Computers and Chemical Engineering 37 (2012) 248–261

Contents lists available at SciVerse ScienceDirect

Computers and Chemical Engineering

journal homepage: www.elsevier.com/locate/compchemeng

# Global Search Metaheuristics for planning transportation of multiple petroleum products in a multi-pipeline system

A. Herrán[a,*], J.M. de la Cruz[b], B. de Andrés[b]

[a] *Department of Computer Science Engineering (CES Felipe II), Complutense University, 28300 Aranjuez, Spain*
[b] *Department of Computer Architecture and Automatic Control, Complutense University, 28040 Madrid, Spain*

ARTICLE INFO

*Article history:*
Received 1 December 2010
Received in revised form 10 October 2011
Accepted 11 October 2011
Available online 4 November 2011

*Keywords:*
Multi-product pipeline network
Mixed-integer linear program
Global Search Metaheuristics
Taboo Search
Simulated Annealing
Multiple Markov Chain

ABSTRACT

The objective of this work is to develop several metaheuristic algorithms to improve the efficiency of the MILP algorithm used for planning transportation of multiple petroleum products in a multi-pipeline system. The problem involves planning the optimal sequence of products assigned to each new package pumped through each polyduct of the network in order to meet product demands at each destination node before the end of the planning horizon. All the proposed metaheuristics are combinations of improvement methods applied to solutions resulting from different construction heuristics. These improvements are performed by searching the neighborhoods generated around the current solution by different Global Search Metaheuristics: Multi-Start Search, Variable Neighborhood Search, Taboo Search and Simulated Annealing. Numerical examples are solved in order to show the performance of these metaheuristics against a standard commercial solver using MILP. Results demonstrate how these metaheuristics are able to reach better solutions in much lower computational time.

## 1. Introduction

Pipelines have been a widely used mode of transportation for petroleum products and their derivatives for the last 40 years. The annual transportation cost in the Petroleum Industry usually surpasses billions of dollars since large volumes have to be transported over long distances. Evidently, pipeline systems play an important role in the industry. Although the initial capital investment required to setup these transportation systems is high, the operating costs are very low compared to other transportation modes such as rail and highway. Even so, the final price of the product depends on its transportation cost, making the optimization of the transportation process a problem of extreme relevance. Consequently, the related scheduling activities for product distribution using pipeline systems have been a focus for at least 30 years.

Several authors have been concentrating on solving the pumping schedule of multiple products for a single multi-product pipeline during the past decade. In such a pipeline, different products are pumped back-to-back without any separation devices between them as shown in Fig. 1. Since there is no physical separation between different products as they move through the pipeline, some mixing and consequent contamination at product interface is unavoidable. These product mixtures are called *transmixes* and they cannot be simply discarded. They must pass through a special treatment that usually involves sending them back to a refinery for reprocessing. The degree of these interface losses depends on the products that come in contact inside the pipeline segment. Moreover, if two products are known to generate high interface losses, the pumping schedule must avoid pumping them back-to-back into the pipeline. Sometimes these pairs of products are considered as forbidden sequences. Moreover, the pumping schedule must take into account the product availability at the refinery and the consumption of different products at each depot.

All the different aspects mentioned above make the pumping schedule of multiple petroleum products from a single refinery to multiple destinations a complex activity. A few papers have been published on this subject in the last decade. Rejowski and Pinto (2003) developed a discrete-time MILP model for the scheduling of a real world multi-product pipeline system with multiple destinations. Later, Rejowski and Pinto (2004) developed a model to improve the computational efficiency from their previous work (Rejowski & Pinto, 2003). Additionally, Rejowski and Pinto (2008) developed a novel continuous-time representation to model the same process considered in their previous papers. On the other hand, Cafaro and Cerdá (2004) developed a continuous-time MILP model for the scheduling of a single pipeline transporting several refined petroleum products from an oil refinery to several distribution terminals. In a subsequent work, Cafaro and Cerdá (2008) extended their formulation considering multiple delivery due dates. Recently, other authors, i.e. Mirhassani and Ghorbanalizadeh

* Corresponding author.
*E-mail address:* aherrang@fis.ucm.es (A. Herrán).

## Nomenclature

*Sets*

| | |
|---|---|
| $C$ | set of connections indexed by $c = 1,\ldots,\lvert C\rvert$ |
| $P$ | set of different petroleum derivatives indexed by $p = 1,\ldots,\lvert P\rvert$ |
| $S$ | set of product pairs $\{(p,p'),\ldots\}$ representing forbidden pumping sequences |
| $T$ | set of time periods in the planning horizon indexed by $t = 1,\ldots,\lvert T\rvert$ |
| $N$ | set of nodes indexed by $n = 1,\ldots,\lvert N\rvert$ |

*Parameters*

| | |
|---|---|
| $CA_{n,p}$ | unit inventory cost for product $p$ at node $n$ |
| $CI_{c,p}$ | unit pumping cost to deliver a package containing product $p$ from its source to its destination through connection $c$ |
| $CT_c$ | start/stop cost at each polyduct (connection) $c$ of the network |
| $CR_{p,p'}$ | unit reprocessing cost of interface material involving different products $p$ and $p'$ |
| $\Delta$ | period length in hours |
| $VC_{p,p'}$ | interface volume between two packages pumped consecutively through the same polyduct containing products $p$ and $p'$ |
| $VP$ | unit package volume in cubic meters |
| $xo_{c,b,p}$ | binary parameter denoting if portion $b$ of connection $c$ is occupied by a package containing product $p$ at the beginning of the planning horizon |

*Variables*

| | |
|---|---|
| $a_{t,n,p}$ | inventory level of product $p$ at node $n$ at period $t$ |
| $qm_{t,n,p}$ | amount of product $p$ transferred at period $t$ from destination node $n$ to its local market |
| $s_{t,c}$ | binary variable denoting if a package is pumped through connection $c$ at period $t$ |
| $v_{t,c,p,p'}$ | interface volume between the package pumped at period $t$ through connection $c$ and the package occupying the first portion of the same connection if the packages contain products $p$ and $p'$, respectively |
| $y_{t,c,p}$ | binary variable denoting if the package pumped through connection $c$ at period $t$ contains product $p$ |

(2008) developed an integer programming formulation to deal with the same problem.

All the papers reviewed above consider the pumping schedule of multiple products for a single pipeline system. However, the polyducts in a specific geographical area (region, country, etc. . .) are connected together, resulting in a more complex system commonly named multi-pipeline system (Cruz, Andrés, Herrán, Besada, & Fernández, 2003; Cruz, Risco, Herrán, & Fernández, 2004; Cruz, Herrán, Risco, & Andrés, 2005). Fig. 2 shows an example of a multi-pipeline system. This network has two source nodes (N1 and N2), two intermediate nodes (N3 and N4) and three destination nodes (N5, N6 and N7). Moreover, the product can flow in both directions through the reversible polyduct joining intermediate nodes N3 and N4. Source nodes could be refineries or other supply systems, for example ports. Destination nodes are the final distribution centers with a specific demand that has to be fulfilled at the end of each planning horizon. On a logistic level, the problem is in planning the way in which different products taken from source nodes are temporally transported to destination nodes, passing through intermediate nodes in order to meet product demands at all pipeline depots before the end of the planning horizon. Furthermore, constraints related to the maximum/minimum inventory levels at each node must be satisfied, and some forbidden product sequences must be avoided due to high product contamination.

Since real world planning and scheduling of complex oil supply chains appears as a challenging problem, several authors have tried to apply different heuristic algorithms in order to aid the decision making process. Despite the fact that heuristic approaches may eliminate the optimal system solution, they are becoming widely used so as to reduce problem complexity and to obtain results in real time. Additionally, decision makers and schedulers usually seek good solutions, close to the optimal, rather than a time consuming optimal solution with little margin of improvement when compared to the others (Relvas, Barbosa-Póvoa, & Matos, 2010). All these facts have motivated the usage of heuristic procedures for solving the pumping schedule of multiple products

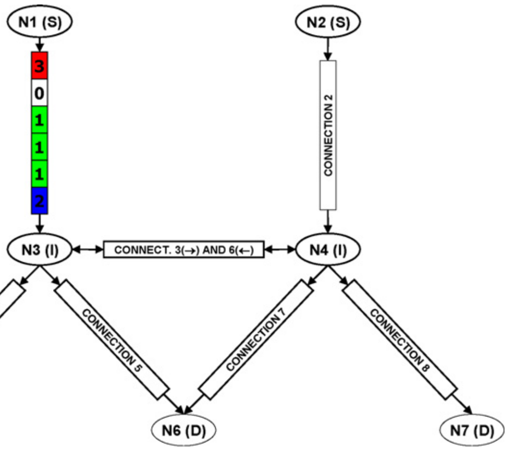


**Fig. 2.** Multi-pipeline system model.

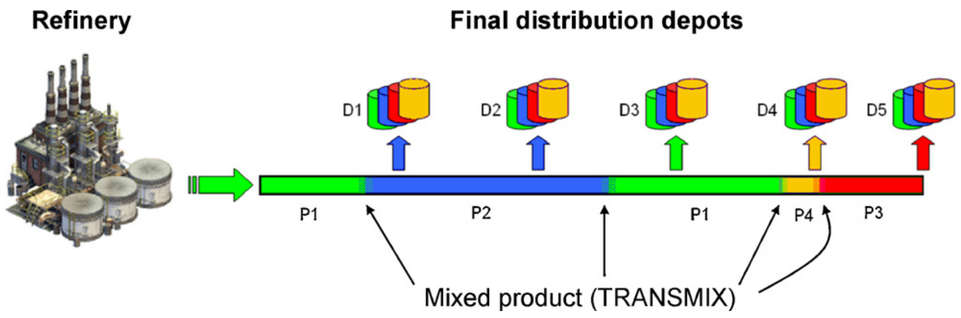


**Fig. 1.** Typical operation of a polyduct system.

for both single and multi-pipeline systems during the past decade. Sasikumar, Prakash, Patil, and Ramani (1997) presented a knowledge-based heuristic search technique providing a monthly pumping schedule to minimize interface and pumping costs. Cruz et al. (2003, 2004, 2005) developed several heuristic algorithms to solve a simplified formulation of the problem presented in Herrán, de la Cruz, and de Andrés (2010). Altiparmak, Gen, Lin, and Paksoy (2006) developed a genetic algorithm for multi-objective optimization of supply chain networks. Other heuristic rules were also used integrated in the short-term scheduling hierarchical approach for refinery operations proposed by Luo and Rong (2007). More recently, Relvas et al. (2010) developed sequencing heuristic to be used prior to the MILP model implementation to provide a set of information on the most desirable sequences of products to be pumped to a multiproduct oil distribution system composed of a single pipeline that connects one refinery to one distribution center. Another issue which increases the complexity in planning and scheduling problems is the existence of uncertainty in the parameters of the problem; i.e. market demands. In these cases, several robust optimization methodologies have been developed to produce "robust" solutions which are in a sense immune against bounded uncertainty (Chen & Lee, 2004; Janak, Lin, & Floudas, 2007). These robust optimization models could be computationally expensive when using classical methods, motivating the usage of heuristics approaches. Summarizing, heuristics can support exact solution methods mainly to provide fast solutions or reduce search space, diminishing the complexity of formulations representing real world systems.

Recently, Herrán et al. (2010) developed a MILP model able to solve the planning of the optimal sequence of products assigned to each new package pumped through each polyduct of the network in order to meet product demands at each destination node before the end of the planning horizon. The criteria used to get this optimal planning were to minimize the pumping and start/stop costs, interface losses, and inventory carrying costs. In order to keep inventory levels in each node within its permissible range, the inventory levels in each node must be tracked over the planning period. In this sense, product balance at intermediate nodes is the most difficult process to model; however, Herrán et al. (2010) easily modeled it assuming a discrete transport approach that divides both the planning horizon into time intervals of equal duration and the individual polyducts into packages of equal volume containing a single product. This method leads to a MILP problem, which could be computationally expensive for large networks when using classical methods (Bazaraa, Jarvis, & Sherali, 1990; Schrijver, 1986). The complexity of such a preeminent problem was also analyzed by Milidiú, Pessoa, and Laber (2003). This problem falls into the commonly named DLSP, Discrete Lot Sizing and Scheduling Problems (Drexl, 1997), which are, in general, NP-hard. In these cases, an alternative to classical methods are heuristic methods (Pereira, Carvalho, Pedroso, & Constantino, 2003) which are especially well suited to solve combinatorial problems, Aarts and Lenstra (2003). This paper proposes different Global Search Metaheuristics to efficiently solve the transportation problem presented in Herrán et al. (2010) without any simplification. The remainder of this paper is organized as follows. Section 2 shows the problem description together with the solution representation and some useful heuristic operators to generate initial solutions and modify the solutions during the search. Section 3 provides a detailed description of all the Global Search Metaheuristics proposed in this paper. Section 4 shows how to improve the efficiency of all the metaheuristics shown in Section 3 by using Multiple Markov Chains. In Section 5, several numerical examples are presented to show the utility of these metaheuristics against a standard commercial solver using MILP. Finally, conclusions are presented in Section 6.

## 2. Model development

### 2.1. Problem description

The pipeline network under study can be initially represented by a set of nodes ($N$), connections ($C$) and products ($P$), of which the activity is determined by a time interval $T$ (planning horizon) in which the demand must be fulfilled. For the network components $N = NS \cup NI \cup ND$ is the set of nodes, where $NS$, $NI$ and $ND$ are the subsets of source, intermediate and destination nodes, respectively. $C$ is the set of the network connections and $CB$ is the set of bidirectional ones, whose elements are ordered pairs of elements of $C$. As an example, for the network shown in Fig. 2, $C = \{c_1, c_2, c_3, c_4, c_5, c_6, c_7, c_8\}$ and $CB$ is the two-dimensional set $CB = \{(c_3, c_6), (c_6, c_3)\}$. It is also defined $CI_n$ as the subset of incoming connections to node $n$ and $CO_n$ as the outgoing ones. An example for the third node of this network is $CI_3 = \{c_1, c_6\}$and $CO_3 = \{c_3, c_4, c_5\}$.

To summarize the problem description, fully detailed in Herrán et al. (2010), given the following information:

- network configuration: number of source, intermediate and destination nodes; number of polyducts and their length; connection topology and number of products,
- the initial state of the network and the length of the planning horizon,
- scheduled production at each refinery according to the planning horizon,
- maximum/minimum allowed product inventory for each depot at each node,
- product inventory at each node at the beginning of the planning horizon,
- demand to be satisfied at each consumer node at the end of the planning horizon,
- cost associated to each operation process in the network;

together with the following assumptions:

- each individual pipeline is composed of an integral number of packages – all of which feature the same volume and contain a single product,
- each package is injected at the same pump rate regardless of the product that it contains; thus, the planning horizon can be divided into an integral number of time intervals featuring the same length,
- direct transfer of product packages between consecutive pipelines is not allowed,
- every package of product pumped into an individual pipeline comes from the tank farm of a depot located at the pipeline inlet,
- simultaneous package injections into several pipelines from the same source or intermediate depot located at their common inlet are permitted,
- every product demand at all pipeline depots must be satisfied before the end of the planning horizon,
- shutdown and re-starting operations for each individual pipeline have a finite cost that can change with the polyduct;

the objective is to establish the optimal sequence of products assigned to each package pumped through each polyduct of the network in order to meet product demands at each destination node before the end of the planning horizon with the minimum total cost $z$. This objective can be expressed using the nomenclature in Herrán et al. (2010) as it is shown in Eq. (1). The first term is the pumping cost with a different cost factor $CI_{c,p}$ for each connection and product. The second term is the start/stop cost of each polyduct with a different cost factor $CT_c$ for each connection. The third term is the reprocessing cost of the interface volume between

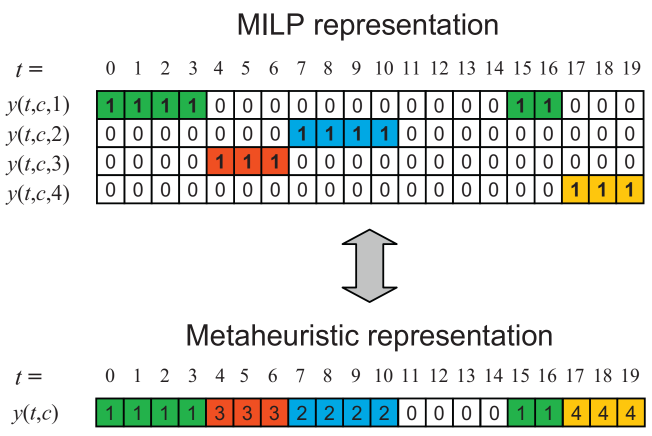

MILP representation
t = 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
y(t,c,1) 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0
y(t,c,2) 0 0 0 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0 0
y(t,c,3) 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0
y(t,c,4) 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1
Metaheuristic representation
t = 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
y(t,c) 1 1 1 1 3 3 3 2 2 2 2 0 0 0 0 1 1 4 4 4

**Fig. 3.** Equivalence between MILP and metaheuristic representations for a solution.

adjacent packages with different products $p$ and $p'$. It has a different cost factor depending on the products $p$ and $p'$, of which, the interface volume could also vary. Finally, the last term stands for the cost of holding product inventory in each node of the network with a different cost factor $CA_{n,p}$ for each node and product.

$$\min z = \sum_t \sum_c \sum_p CI_{c,p} \cdot VP \cdot y_{t,c,p} + \sum_c CT_c \cdot |so_c - s_{1,c}| + \sum_{t>1} \sum_c CT_c \cdot |s_{t-1,c} - s_{t,c}| + \sum_t \sum_c \sum_p \sum_{p'} CR_{p,p'} \cdot v_{t,c,p,p'} + \sum_t \sum_n \sum_p CA_{n,p} \cdot \Delta \cdot VP \cdot a_{t,n,p} \tag{1}$$

The MILP model also has a set of constraints to fix the value of all the variables included in Eq. (1), namely: (1) the value of $y_{t,c,p}$ to ensure that only one package entering a polyduct per period is allowed; (2) the value of $s_{t,c}$ to denote if a polyduct is active or not at each period; (3) the value of the interface volume between adjacent packages with different products, $v_{t,c,p,p'}$; and (4) the value of all the inventory levels at each period of the planning horizon, $a_{t,n,p}$. Moreover, an additional set of constraints is included in order to:

(a) avoid collisions in reversible polyducts,
(b) avoid forbidden pumping product sequences,
(c) avoid violations of maximum/minimum inventory levels,
(d) fulfill market demands.

## 2.2. Solution representation for metaheuristics

In metaheuristics it is convenient to represent the solution with a different set of variables than those used in the MILP formulation. In this paper, however, the solution representation also uses the transport variable $y$, but only indexed in time and connection $\langle t,c \rangle$ with values in the range $[0, |P|]$. Now, $y_{t,c} = p$ means that product $p$ leaves the origin of connection $c$ at period $t$, using $y_{t,c} = 0$ for the case in which no product is sent; i.e. the connection $c$ is stopped at period $t$. Hence, a solution $y = [y_{t,c}]$ can be represented by a $|C| \times |T|$ matrix where rows correspond to connections and columns to periods. The search space $\Omega$ is the set of all solutions considered for this problem (feasible and infeasible), and it is given by Eq. (2). Fig. 3 shows a comparison between both MILP and metaheuristic representations.

$$\Omega = \{y \in [0, 1, \ldots, |P|]^{|C| \times |T|}\} \tag{2}$$

Since metaheuristic methods can handle non-feasible solutions, an additional term is needed to measure their deviation from feasibility in order to decide which one is the best when comparing two solutions. This term, denoted by $Infeas(y)$ for a solution $y \in \Omega$, is calculated measuring the infeasibility degree of a solution by the four terms shown in Eq. (3). A feasible solution must have $Infeas(y) = 0$. The first term, $Inf_a(y)$, is the number of collisions in reversible polyducts, and it is measured by Eq. (4). The second term, $Inf_b(y)$, is the number of forbidden pumping product sequences, and it is measured by Eq. (5). The third term, $Inf_c(y)$, is the amount of product below the minimum or over the maximum inventory levels, and it is measured by Eq. (6). Finally, the fourth term, $Inf_d(y)$, is the amount of product not delivered to the consumer nodes to meet market demands, which is measured by Eq. (7). Notice that all the variables $y_{t,c,p}$, $s_{t,c}$, $v_{t,c,p,p'}$, $a_{t,n,p}$ and $qm_{t,n,p}$ appearing in Eqs. (1) and (4)–(7) can be calculated by the model proposed in Herrán et al. (2010). Now, given two solutions $y_1, y_2 \in \Omega$, we say that $y_1$ is better than $y_2$ ($y_1 < y_2$) if the condition shown in Eq. (8) is fulfilled.

$$Infeas(y) = Inf_a(y) + Inf_b(y) + Inf_c(y) + Inf_d(y) \tag{3}$$

$$Inf_a(y) = \sum_t \sum_{(c,c') \in CB} r^{(a)}_{t,c,c'} \quad \text{with} \quad r^{(a)}_{t,c,c'} = \begin{cases} 1 & \text{if } y_{t,c} \cdot y_{t,c'} \neq 0 \\ 0 & \text{otherwise} \end{cases} \tag{4}$$

$$Inf_b(y) = \sum_t \sum_c r^{(b)}_{t,c} \quad \text{with} \quad \begin{array}{l} r^{(b)}_{1,c} = \begin{cases} 1 & \text{if } (xo_c, y_{1,c}) \in S \\ 0 & \text{otherwise} \end{cases} \\ r^{(b)}_{t,c} = \begin{cases} 1 & \text{if } (y_{t-1,c}, y_{t,c}) \in S \\ 0 & \text{otherwise} \end{cases} \quad : t > 1 \end{array} \tag{5}$$

$$Inf_c(y) = \sum_t \sum_n r^{(c)}_{t,n} \quad \text{with} \quad r^{(c)}_{t,n} = \begin{cases} Amin_{n,p} - a_{t,n,p} & \text{if } a_{t,n,p} < Amin_{n,p} \\ a_{t,n,p} - Amax_{n,p} & \text{if } a_{t,n,p} > Amax_{n,p} \end{cases} \tag{6}$$

$$Inf_d(y) = \sum_n \sum_p r^{(d)}_{n,p} \quad \text{with} \quad r^{(d)}_{n,p} = \begin{cases} DM_{n,p} - \sum_t qm_{t,n,p} & \text{if } \sum_t qm_{t,n,p} < DM_{n,p} \\ 0 & \text{otherwise} \end{cases} \tag{7}$$

$$[Infeas(y_1) < Infeas(y_2)] \text{ OR } [(Infeas(y_1) = Infeas(y_2)) \text{ AND } (z(y_1) < z(y_2))] \tag{8}$$

## 2.3. Construction heuristics

Construction heuristics are used to generate an initial solution for search metaheuristic procedures. Here, three different construction heuristics are developed. The first one is a random construction, which requires a very short computational time, but it fails to obtain a feasible solution. Therefore, two other construction heuristics are developed based on the solution of two different relaxed versions of the MILP formulation proposed in Herrán et al. (2010).

- *Random* (*Cr*). This procedure generates a solution in the most random way. It randomly chooses the product to be sent through each polyduct at each period of the planning horizon taking into account the product availability at the sources from which the products are taken. It is also possible to choose the value of zero

$$y=\begin{bmatrix} p_1 & * & * \\ * & p_2 & * \end{bmatrix} \Rightarrow N(y)=\left\{\begin{bmatrix} 0 & * & * \\ * & 0 & * \end{bmatrix},\begin{bmatrix} 0 & * & * \\ * & 1 & * \end{bmatrix},\begin{bmatrix} 0 & * & * \\ * & 2 & * \end{bmatrix},\ldots,\begin{bmatrix} |P| & * & * \\ * & |P| & * \end{bmatrix}\right\}$$

**Fig. 4.** Effect of the *Replace* neighborhood operator for $|T| = 3$ and $M = 2$.

$$y=\begin{bmatrix} p_1 & p_2 & p_3 \\ * & * & * \end{bmatrix} \Rightarrow N(y)=\left\{\begin{bmatrix} 0 & 0 & p_3 \\ * & * & * \end{bmatrix},\begin{bmatrix} p_1 & 0 & 0 \\ * & * & * \end{bmatrix},\begin{bmatrix} 1 & 1 & p_3 \\ * & * & * \end{bmatrix},\ldots,\begin{bmatrix} p_1 & P & P \\ * & * & * \end{bmatrix}\right\}$$

**Fig. 5.** Effect of the *Batch* neighborhood operator for $|T| = 3$ and $M = 2$.

$$y=\begin{bmatrix} * & * & * \\ p_1 & p_2 & p_3 \end{bmatrix} \Rightarrow N(y)=\left\{\begin{bmatrix} * & * & * \\ p_2 & p_1 & p_3 \end{bmatrix},\begin{bmatrix} * & * & * \\ p_3 & p_2 & p_1 \end{bmatrix},\begin{bmatrix} * & * & * \\ p_1 & p_3 & p_2 \end{bmatrix}\right\}$$

**Fig. 6.** Effect of the *Swap* neighborhood operator for $|T| = 3$.

for $y_{t,c}$ denoting a stoppage in the polyduct associated to connection $c$ at period $t$.

- *Linear relaxation of MILP* (*Cl*). The information provided by the solution given by a linear relaxation of the MILP model can be used in a construction heuristic to obtain an integer solution. An initial application of this idea was used in Lengauer (1990, Chapter 8). This relaxation consists of allowing the values of the variable $y$ in the MILP formulation to be real and subjected to the constraint shown in Eq. (9), leading to a LP problem. Hence, for each $t \in T$, $c \in C$ and $p \in P$ the solution of the relaxed problem can be considered as the probability of sending product $p$ through connection $c$ at period $t$. This heuristic provides solutions that, although still infeasible in general, are better than the ones given by a purely random construction.

$$0 \le y_{t,c,p} \le 1; \quad \forall (t, c, p) \tag{9}$$

- *Quadratic relaxation of MILP* (*Cq*). The main problem of *Cl* construction method is that the probability of sending a product for a given $c$ and $t$ tends to be the same for all the products. This fact makes the solutions generated by *Cl* to be unfeasible regarding product collisions on reversible polyducts, Eq. (5). To solve this problem a novel approach has been developed. In order to force the solution of the linear relaxed version (LP) to be next to the integer one (MILP), the quadratic function shown in Eq. (10) is added to the objective function shown in Eq. (1), leading to a QP problem. Since this function is zero for integer solutions and greater than zero for non-integer solutions, by minimizing this term the solution of this QP problem would have a higher integrality degree than the solutions of the LP problem. In this way, an initial solution built with *Cq* would satisfy more constraints than the one provided by the *Cl* method.

$$\begin{aligned} z_5 &= \sum_t \sum_c \left(1 - \sum_p y_{t,c,p} \cdot y_{t,c,p}\right) \\ &= |T| \cdot |C| - \sum_t \sum_c \sum_p y_{t,c,p} \cdot y_{t,c,p} \end{aligned} \tag{10}$$

## 2.4. Neighborhood generation

The optimization methods based on search metaheuristics work generating an initial solution by some method, like the ones proposed in the previous section, and modifying the current solution in search for a better one. In this work, four heuristic methods have been developed to generate a neighborhood of solutions around the current one.

- *Replace* (*Nr*). This operator takes $M$ elements from the solution matrix $y$, and it replaces them by all the possible combinations of these $M$ elements with a value between $[0,|P|]$. Hence, the size of this neighborhood is $(|P| + 1)^M - 1$, since we have to remove the original solution. An example of this neighborhood operator is shown in Fig. 4 for $|T| = 3$ and $M = 2$. This operator is particularly efficient in those cases where the initial solution provides some residual value.
- *Batch* (*Nb*). This operator chooses a random number between 0 and $|T| - 1$, namely $M$, and it generates the neighborhood by replacing all the $(|T| - M + 1)$ sets of consecutive elements of a randomly selected connection from the solution matrix by a batch of $M$ elements of product $p$. This operator is repeated for all the products included 0 (pumping stoppage). Hence, the maximum size of this neighborhood is $(|T| - M + 1)\cdot(|P| + 1)$. Fig. 5 shows an example of this neighborhood operator for $|T| = 3$ and $M = 2$. This operator is especially effective when the solution matrix $y$ needs to change the product type of some batch. Note that this operator is different from the *Replace* one since it is applied to batches.
- *Swap* (*Ns*). This operator generates the neighborhood by swapping all the $(|T| - 1)$ pairs of elements of a randomly selected connection from the solution matrix. Hence, the size of this neighborhood should be $\sum n$, however, it is generally lower because the solutions generated by swapping two elements with the same value are not added to the neighborhood. Fig. 6 shows an example of this neighborhood operator for $|T| = 3$. The swap operator is especially effective when the solution matrix contains all the products needed to meet demand but product relocation is needed.
- *Move* (*Nm*). This operator generates the neighborhood by moving from 1 to $|T|$ periods all the elements of a randomly selected connection from the solution matrix. If all the elements of the row associated to the selected connection are equal, a different connection is again randomly selected to perform the move neighborhood procedure. Hence, the size of this neighborhood is $(|T| - 1)$. Fig. 7 shows an example of this neighborhood operator

$$y=\begin{bmatrix} * & * & * \\ p_1 & p_2 & p_3 \end{bmatrix} \Rightarrow N(y)=\left\{\begin{bmatrix} * & * & * \\ p_3 & p_1 & p_2 \end{bmatrix},\begin{bmatrix} * & * & * \\ p_2 & p_3 & p_1 \end{bmatrix}\right\}$$

**Fig. 7.** Effect of the *Move* neighborhood operator for $|T| = 3$.

for $|T| = 3$. This operator has the same advantages and disadvantages as the swap operator.

## 3. Global Search Metaheuristics

As it was said above, search metaheuristics are those that provide strategies for exploring the space of candidate solutions to a problem by moving iteratively to a neighborhood solution in search for a better one. Such processes are known as monotonous searches, hill climbing or local searches. Roughly speaking, a local search is based on the study of solutions of the neighborhood or environment of the solution runs. It consists of iteratively choosing the best of such solutions as long as there is some possible improvement. The main drawback of local searches is that local optimal solutions used to be trapped in their environment (Yagiura & Ibaraki, 2002). One possible solution is to extend the local search beyond the local optimum by a global search. Global Search Metaheuristics (GSMs) incorporate guidelines for three basic ways to escape from a poor quality local optimum: (a) to restart the search from another boot solution by a Multi-Start Search, (b) to modify the structure of neighborhoods by a Variable Neighborhood Search; and (c) to apply non-monotonous movements during the search by accepting worse solutions than the current one. Non-monotonous searches keep control over the possible worsening movements by using some memory structures in the search process, whose most representative technique is Taboo Search; or some stochastic acceptance criteria, whose most representative technique is Simulated Annealing. A more detailed description (pseudocode) for the procedures required to implement all GSMs presented in this section can be found in Appendix A, supplied as online supplementary information.

### 3.1. *Multi-Start Search*

Multi-Start Search (MSS) is the simplest approach for global search to escape from local minima that trap a local search (Martí, 2003). It is based on restarting the search each time the algorithm is trapped in a local minimum. Local search is trapped in a local minimum when none of the neighbor solutions improves the current one. However, this is not a “blockage” situation, since there are other neighborhoods to explore (i.e. the ones generated by selecting different elements as the *M* elements chosen in *Nr* or the connection selected in *Nb*, *Ns* or *Nm*). For this reason, each time the algorithm is trapped in a local minimum, the local search is restarted from the best current solution instead from a new one. Now, if this situation continues after *K* iterations, the local search is restarted from a new solution constructed by any of the construction methods explained above.

### 3.2. *Variable Neighborhood Search*

Even with the MSS algorithm, once all the possible neighborhoods around the current solution are explored without reaching a better solution than the current one, the algorithm is trapped on a local minimum. In these cases, Variable Neighborhood Search (VNS) offers a good alternative to MSS to escape from local minima by modifying the structure of neighborhoods each time the algorithm is trapped on one of these minima. In fact, modifying the structure of neighborhoods at each iteration allows the algorithm to improve the quality of solutions along more iterations before restarting the algorithm. Hence, the procedures used in this metaheuristic are similar to those used in the MSS algorithm except in the strategy used to generate neighborhoods (Hansen & Mladenovic, 1999). In VNS the method used to generate the neighborhoods is modified at each iteration before the local search is called. The simplest strategy to do this consists of selecting one of the available methods to generate such neighborhoods (*Nr*, *Nb*, *Ns* and *Nm*) in a sequential way; this method is denoted as *Nc*. A better approach consists of selecting the neighborhood method according to some empirical probability of success when using one of the aforementioned methods or another; this method is denoted as *Np*. This probability, calculated at each iteration, is given by the expression shown in Eq. (11) where $S(Ni)$ is the number successful attempt with neighborhood $Ni \in \{Nr, Nb, Ns, Nm\}$, and $T(Ni)$ is the total number of attempt with this neighborhood.

$$P(Ni) = \frac{S(Ni)}{T(Ni)} \cdot \left( \sum_{i \in \{r,b,s,m\}} \frac{S(Ni)}{T(Ni)} \right)^{-1} \quad ; \quad i \in \{r, b, s, m\} \qquad (11)$$

### 3.3. *Taboo Search*

A different approach to escape from local minima is non-monotonous searches. These search metaheuristics are able to accept worse solutions than the current one each time the algorithm is trapped on a local minimum. Obviously, accepting worse solutions could lead the algorithm back to a previously visited solution, and the search could be trapped in a cycle. Taboo Search (TS) avoids this problem by using memory structures (Glover & Laguna, 1997). Once a potential solution has been determined it is stored into a “taboo” list, so that the set of solutions that the algorithm can use in each worsening movement excludes all the previously accepted solutions in order to avoid cycles. The main difference between monotonous searches (as MSS or VNS) and TS is that, while in MSS or VNS the search is directed towards a new solution built any time the algorithm is trapped on a local minimum, in TS the search is directed towards a non-taboo solution randomly chosen within the neighborhood.

### 3.4. *Simulated Annealing*

Like TS, Simulated Annealing (SA) is a kind of non-monotonous search able to escape from a local minimum by accepting, with some probability, worse solutions than the current one during the search. The first SA algorithm was first introduced by Kirkpatrick, Gelatt, and Vecchi (1983) based on the work of Metropolis, Rosenbluth, Rosenbluth, Teller, and Teller (1953). SA can be seen as a stochastic algorithm generating a sequence $y_0, y_1, \ldots, y_n$, of solutions approaching the set of optimal solutions as $n \to \infty$. This sequence is generated by iteratively moving to a neighbor solution, which is accepted by following Eq. (12), where $y'_n$ is the neighbor solution of $y_n$ and $z(y_n)$ is the value of its objective function. In this equation *rand*(0,1) is a random number generator between (0,1) for making a stochastic decision on the acceptance of the new solution. Finally, $T_n$ (>0) is the temperature at the *n*th iteration, such that $T_{n+1} \le T_n$, which decreases the probability of accepting worse solutions than the current one as $n \to \infty$. The SA algorithm starts building an initial solution according to some construction method. Then, each cycle performs *K* attempt movements to improve the current solution at a constant temperature *T*, starting at $T_o$. Each attempt only uses the acceptance criteria for new solutions given by Eq. (12) if both solutions (current and new) have the same feasibility degree. Otherwise, a new solution is only accepted if it has a lower *Infeas*() value than the current one. After that, the temperature is modified according to the cooling schedule given by the equation $T = \alpha \times T$, where $\alpha$ is a value between [0,1] generally chosen to be close to 1.

$$y_{n+1} = \begin{cases} y'_n & \text{if } \exp(z(y_n) - z(y'_n)/T_n) > r \text{ and } (0, 1) \\ y_n & \text{otherwise} \end{cases} \qquad (12)$$

## 4. Multiple Markov Chain based algorithms

### 4.1. Sequential Multiple Markov Chain based algorithms

All the GSMs described above can be seen as stochastic algorithms generating a sequence $y_0, y_1, \ldots, y_n$ of solutions approaching the set of optimal solutions as $n \to \infty$. This sequence is generated by moving iteratively to a neighbor solution which is accepted under some stochastic criteria. Since the choice of $y_{n+1}$ depends only on the current solution $y_n$ but not on the previously visited ones, the search path of such algorithms follows a first order Markov chain. These algorithms used to be referenced as Sequential Single Markov Chain (SSMC) algorithms. In SSMC algorithms, the probability of not getting an optimal solution after $n$ iterations is characterized by Eq. (13), where $S_{min}$ is a set of optimal solution points and $C > 0$ and $\theta > 0$ are constant values associated to a given objective function and neighborhood generation method respectively. Thus as $n \to \infty$, the solution converges to one of the optimal points in $S_{min}$ with a probability of 1.

$$P(y_n \notin S_{\min}) \approx \left(\frac{C}{n}\right)^{\theta} \tag{13}$$

Although most schemes in the literature follow a single Markov chain, it could be inefficient from a performance point of view (Defersha & Chen, 2008). To solve this problem, a Sequential Multiple Markov Chain algorithm (SMMC) performs $J$ independent versions of SSMC algorithm on a single processor computer, using the same search space, neighborhood generation, and cooling schedule for SA. Each one of these independent versions is stopped after $n$ iterations to provide $J$ independent terminal solutions $\{y_{n,1}, y_{n,2}, \ldots, y_{n,J}\}$. Then, out of these terminal solutions, the best one is chosen as the final solution $y_n$. Now, the probability of not getting an optimal solution after $n$ iterations shown in Eq. (13) becomes the one shown in Eq. (14). Thus, for $0 < C/n < 1$, this probability decreases exponentially as $J$ increases when using SMMC (Azencott, 1992), while the CPU time increases only linearly as the function of $J$. Hence, given a CPU time $Tcpu$, it is more efficient to move $n/J$ iterations in a SMMC than move $n$ iterations in a SSMC.

$$P(y_n \notin S_{\min}) = \prod_{j} P(y_n \notin S_{\min}) \approx \left(\frac{C}{n}\right)^{\theta \cdot J} < \left(\frac{C}{n}\right)^{\theta} \tag{14}$$

### 4.2. Distributed Multiple Markov Chain based algorithms

Distributed Multiple Markov Chain (DMMC) algorithms are those in which $J$ Markov chains are partitioned into equal subgroups as $J_1, J_2, \ldots, J_n$ and distributed to $N$ concurrently available SMMC algorithms, communicating to each other at every $R$ iteration in order to further improve the solution quality. This communication is based on the migration of the best solutions from one SMMC to another according to different interaction topologies. The most commonly used topologies are *Ring*, *Fully Connected* and *Master–Slave*. In the first one, *Ring*, the best solution found by each SMMC is sent to all its neighbors. The second scheme, *Fully Connected*, is similar to *Ring* but sending the best solution found by each SMMC to all the rest of the available SMMCs. In the *Master–Slave* scheme, the best solution so far found among all the SMMCs determines the master algorithm. Then, this solution is sent to all of the remaining SMMCs, the slaves. Finally, the best solution found by each slave is sent to the master SMMC. Each time a SMMC receives a solution, it is only accepted if it improves the worst solution found by the receiving SMMC. According to Distributed Parallel Genetic Algorithms (Alba & Troya, 2000), the best performance of using $N$ SMMCs with $J/N$ Markov chains (each over a single SMMC with $J$ Markov chains) is attributed to the following characteristics: (1) their decentralized search which allows more speciation; (2) the larger diversity levels, since many search regions are sought at the same time; and (3) the exploitation inside each SMMC by refining the best partial solutions found in each SMMC.

### 4.3. Parallel Multiple Markov Chain based algorithms

As discussed above, SMMC may result in an exponential reduction of the error probability with a linear increasing of computational time as the number of independent SSMC runs increases. A promising technique to achieve this exponential reduction of the error probability with less or no increment of computational time is to use parallel computing. Assume a SMMC with $J$ Markov chains requiring $t_s$ unit time to perform $i$ iterations in each search direction. The value of $J$ can be increased manifold keeping $t_s$ and $i$ unchanged using parallel computing to further reduce the error probability. This can be done by having multiple copies of SMMC run on several concurrently available computers (Azencott, 1992; Lee & Lee, 1996). Let $J$ Markov chains be partitioned into equal subgroups as $J_1, J_2, \ldots, J_p$ and distributed to $P$ concurrently available processors. The computational time for the Parallel Multiple Markov Chain (PMMC), $t_p$ will be equal to $t_s/P$, where $t_s$ is the computational time for the SMMC.

## 5. Numerical examples

All the GSMs presented in this paper will be illustrated by solving the application example developed by Herrán et al. (2010) under several scenarios. This example involves the network shown in Fig. 2 transporting four refined petroleum products ($P1$: gasoline; $P2$: diesel oil; $P3$: LPG; $P4$: jet fuel). Tables with common data for all the scenarios can be found in Appendix A. These data include the lower and upper limits for all depots, the scheduled production at each refinery along the planning horizon, the inventory cost, the interface material cost and contact volume for each ordered pair of products, and the initial state of the network. These data, together with the demand for all the scenarios are selected to force the pumping of the product through the reversible polyduct in both directions. The demand must be fulfilled at the end of the 100 h planning horizon, composed of $|T| = 20$ time periods of length $\Delta = 5$ h each. Consequently, the problem dimensions are $|N| = 7$ nodes, $|C| = 8$ connections, $|P| = 4$ products and $|T| = 20$ periods. The length of all polyducts is $L = 3$, measured in terms of the number of packages that a polyduct is able to store. A value of $VP = 5000$ m$^3$ is used as the unit package volume. The pumping cost, $CI_{c,p}$, is usually proportional to the polyduct length, and in this case it is set to $L$ US\$/m$^3$ for all $c$ and $p$.

The proposed MILP approaches described in Herrán et al. (2010) were illustrated by solving this application example under several scenarios. These MILP approaches are composed by a *complete model* and a *simplified* one which can be used when fulfillment of depot demands requires pump operations during the entire planning horizon. All the scenarios were solved by both models using CPLEX with ILOG OPL-Studio 4.2 (ILOG Inc., 2006) on an Intel Xeon IV 2.8 GHz/2GB RAM processor. The relative MIP gap tolerance and the integrity tolerance were set to $1 \times 10^{-4}$ and $1 \times 10^{-5}$, respectively. In *Scenarios* I and II the demand at all the destinations is 10 $VP$m$^3$ for all the products. However, while in *Scenario* I the start/stop cost factor $CT_c$ is set to 100,000 US\$ for all polyducts, in *Scenario* II this value is set to 500,000 US\$. In *Scenario* III the start/stop cost factor $CT_c$ is reduced again to 100,000 US\$, but the demand is increased to 13 $VP$m$^3$ for all the products at destinations D1 and D3, and to 18 $VP$m$^3$ at D2. Table 1 summarizes the main characteristics of these scenarios. These scenarios were chosen into show the utility of the simplified model under certain circumstances. For example, if there is a high start/stop cost (*Scenario* II) or a high demand pattern

**Table 1**
CPU time (in s) needed by each algorithm to reach the optimal solution in all scenarios.

| | *Scenario* I | *Scenario* II | *Scenario* III |
|---|---|---|---|
| Demand | Low | Low | High |
| Start/stop cost factor | Low | High | Low |
| Model to use | Complete | Complete/simplified | Complete/simplified |
| CPU time (s) – complete model | 19,475 | 9227 | 35,630 |
| CPU time (s) – simplified model | × | 889 | 3685 |
| Optimal cost (US$) | 2,759,400 | 4,312,275 | 3,493,650 |

(*Scenario* III), the *simplified model* is able to reach the same optimal solution as given by the *complete model* with more than one order of magnitude less time. However, in a low demand scenario and when the start/stop cost is very low compared to the pumping cost (*Scenario* I) the *complete model* should be used, since fulfillment of depot demands would not require pump operations during the entire planning horizon, and consequently, the simplified model could give non-optimal solutions. In this case, the problem has 5245 variables and 13,086 constraints, and the MILP algorithm needs almost 20,000 s to reach the optimal solution. Therefore, the GSMs proposed in this paper could be a good alternative to efficiently solve this problem.

Before solving all these scenarios by the metaheuristic algorithms here proposed, the next sections show an analysis of each single GSM for the problem here treated in *Scenario* II. We chose this scenario since, as can be seen from Table 1, it is less time consuming when it is solved by CPLEX. First, Section 5.1 shows an analysis of the best construction and neighborhood generation methods by using monotonous searches (MSS and VNS). Next, Section 5.2 shows the performance of non-monotonous searches (TS and SA) together with an analysis of the robustness of SA algorithm, which seems to be the best among all others when solving the problem here considered. The performance improvement from using SMMC algorithms over SSMCs is presented in Section 5.3 for SA algorithm. This section also shows how to further improve the performance given by SMMCs by using a DMMC algorithm. Section 5.4 shows a comparison of the best GSM against CPLEX for all the scenarios solved in Herrán et al. (2010). Finally, Section 5.5 shows a larger application example than the ones presented in Herrán et al. (2010). All these metaheuristics were implemented in C++ using the Borland C++ Builder 5 compiler and were solved with the same processor as CPLEX.

## 5.1. Monotonous searches: MSS and VNS

All the metaheuristic presented in this paper are combinations of improvement methods applied to solutions resulting from construction heuristics (*Cr*, *Cl* and *Cq*). These improvements are performed by searching the neighborhoods generated around the current solution by four different methods (*Nr*, *Nb*, *Ns* and *Nm*) or a combination of them (*Nc* and *Np*). This means that there are different implementations of each metaheuristic according to the methods selected to build new solutions and to generate neighborhoods. Thus, there are twelve different implementations of the MSS algorithm and six different implementations of the VNS algorithm. Neighborhoods are randomly explored in search of a better solution than the current one. The current solution ($y$) is automatically replaced by the current neighbor explored ($y^*$) whenever $y^*$ is better than $y$. Experience on solving this problem demonstrates that this strategy offers better results that a total exploration of neighborhoods. Moreover, a value of $K = 100$ iterations was used for both MSS and VNS.

In order to compare the convergence among all the algorithms here developed, each one is run 25 times, and the average value of the objective function together with the infeasibility degree is compared for all the instances. In this way, results are more reliable than the obtained ones when trying only one time each metaheuristic. Note that the infeasibility degree, shown in Eq. (3), is different from zero if there are at least one unfeasible solution into the 25 times each algorithm is run. Table 2 shows the results given by all the different implementations of the MSS and VNS algorithms. This table shows the number of feasible and optimal solutions found among the 25 instances together with the average values of the infeasibility degree and cost (in US$) of the best solutions found by each algorithm after 100 s of computation. The last column also shows these values for the best solution found among the 25 times each algorithm is run.

As can be seen from this table the best construction heuristic is *Cq*, followed by *Cl* and finally *Cr*. Looking at the results for the VNS algorithm, which includes all the neighborhood generation methods, it can be seen how the three generation methods are able to reach feasible solutions all 25 times each algorithm is run. However, by using *Cl* instead of *Cr* the results are improved in terms of cost value in all cases. Additionally, by using *Cq* instead of *Cl*, the cost values are further improved and some optimal solutions are reached despite the short time the algorithm is run. Regarding the neighborhood generation, the *Nb* operator gives the best results, followed by *Nr, Ns* and *Nm*. Looking at the results for the MSS algorithm, since it uses different neighborhood generation methods, it can be seen how *Nb* is able to reach feasible solutions all 25 times each algorithm is run even using *Cr* as a construction method. The *Nr* method is also able to reach some feasible solutions for all the construction methods; however, it has worse quality than *Nb* in terms of the average infeasibility degree and cost value. Finally, the worst neighborhood generation method seems to be *Nm*, since it is unable to reach any feasible solutions even when it is combined with *Cq*. Fig. 8 shows the convergence of the average infeasibility degree and cost function value along 100 s of computation for all the construction heuristics used by the MSS–$C^*$–*Nr* algorithm. In this figure, and in all the next ones, the cost function values associated to feasible solutions are marked with a circle. Fig. 9 is similar to Fig. 8, but comparing all the neighborhood generation methods used by the MSS–*Cr*–$N^*$ algorithm. Although Fig. 9 shows how the *Nb* operator is the only one able to reach feasible solutions, it could be trapped in a local minimum due to "premature convergence". To avoid this problem, the VNS algorithm is able to swap to a different neighborhood operator according to *Nc* or *Np* strategies. As can be seen from Table 2, both strategies improve the results reached by MSS–$C^*$–*Nb* for all the construction methods and feasible solutions are always reached for all 25 instances run. Fig. 10 shows the convergence of both algorithms (VNS–*Cr*–*Nc* and VNS–*Cr*–*Np*) against MSS–*Cr*–*Nb*.

## 5.2. Non-monotonous searches: TS and SA

The non-monotonous searches, TS and SA, always use *Np* for the neighbor generation. Therefore, there are only three different implementations of these algorithms. A value of $K = 100$ iterations was used for the TS algorithm and a size of 200 solutions was used for the taboo list. This value was chosen to be greater than the size of the neighborhoods generated by any of the methods used by *Np*. Experience from solving this problem demonstrates that no

**Table 2**
Results of all the implementations of MSS and VNS algorithms after 100 s of computation.

| Implementation | No. of feasible solutions | No. of optimal solutions | Average solution | | Best solution | |
|---|---|---|---|---|---|---|
| | | | Infeas | Cost (US$) | Infeas | Cost (US$) |
| MSS–*Cr*–*Nr* | 19 | 0 | 0.80 | 10,368,911 | 0.00 | 7,966,075 |
| MSS–*Cl*–*Nr* | 12 | 0 | 3.04 | 7,819,847 | 0.00 | 6,482,900 |
| MSS–*Cq*–*Nr* | 21 | 0 | 0.60 | 8,054,168 | 0.00 | 6,372,100 |
| MSS–*Cr*–*Nb* | 25 | 0 | 0.00 | 4,837,878 | 0.00 | 4,817,775 |
| MSS–*Cl*–*Nb* | 25 | 0 | 0.00 | 4,822,392 | 0.00 | 4,812,275 |
| MSS–*Cq*–*Nb* | 25 | 0 | 0.00 | 4,820,960 | 0.00 | 4,813,650 |
| MSS–*Cr*–*Ns* | 0 | 0 | 25.84 | 15,659,627 | 14.00 | 13,123,950 |
| MSS–*Cl*–*Ns* | 0 | 0 | 22.36 | 6,166,704 | 11.00 | 4,469,525 |
| MSS–*Cq*–*Ns* | 21 | 0 | 0.16 | 8,072,756 | 0.00 | 5,454,200 |
| MSS–*Cr*–*Nm* | 0 | 0 | 72.48 | 31,928,342 | 56.00 | 29,898,275 |
| MSS–*Cl*–*Nm* | 0 | 0 | 34.84 | 7,949,719 | 22.00 | 5,507,425 |
| MSS–*Cq*–*Nm* | 0 | 0 | 15.80 | 15,484,336 | 14.00 | 15,430,700 |
| VNS–*Cr*–*Nc* | 25 | 0 | 0.00 | 4,494,576 | 0.00 | 4,316,350 |
| VNS–*Cl*–*Nc* | 25 | 0 | 0.00 | 4,335,498 | 0.00 | 4,315,275 |
| VNS–*Cq*–*Nc* | 25 | 0 | 0.00 | 4,323,837 | 0.00 | 4,314,000 |
| VNS–*Cr*–*Np* | 25 | 0 | 0.00 | 4,389,946 | 0.00 | 4,314,900 |
| VNS–*Cl*–*Np* | 25 | 0 | 0.00 | 4,335,222 | 0.00 | 4,314,425 |
| VNS–*Cq*–*Np* | 25 | 0 | 0.00 | 4,322,810 | 0.00 | 4,313,475 |

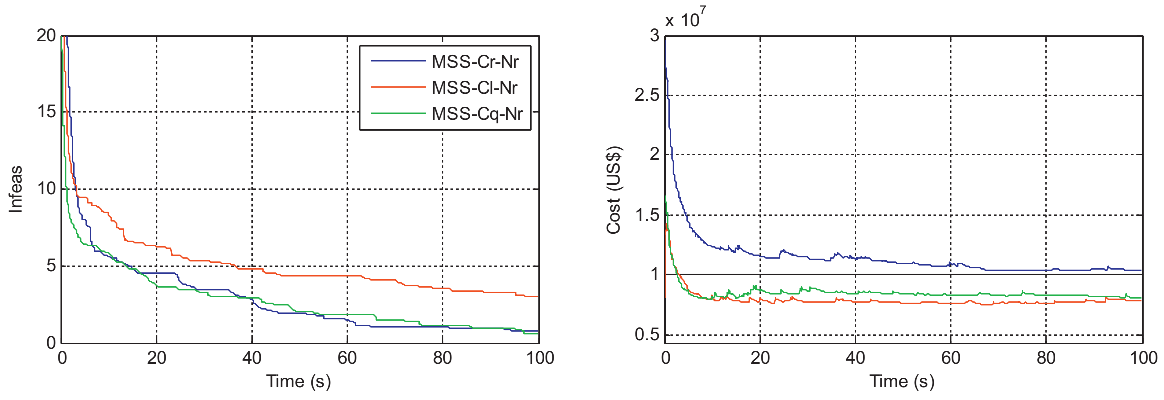


**Fig. 8.** Comparison of all the construction heuristics for the MSS–*C**–*Nr* algorithm.

improvement is achieved on the quality of solutions by increasing the size of the taboo list over this value. Regarding SA, a preliminary study was made in order to fix an adequate order of magnitude for *K*, $\alpha$ and $T_0$, resulting in the values $K = 1000$, $\alpha = 0.990$ and $T_0 = 5 \times 10^6$. Table 3 shows the results given after 100 s of computation by all the different implementations of TS and SA algorithms. These results show how both algorithms are able to reach some optimal solutions for all the construction methods. Moreover, the SA algorithm improves the performance of TS. These algorithms are able to avoid the premature convergence to local minima by including the possibility to accept worse solutions than the current one during the search. Fig. 11 shows the convergence of both algorithms (TS and SA) together with the best implementations of the monotonous searches when using *Cr* as a construction method; i.e. MSS–*Nb*–*Cr* and VNS–*Np*–*Cr*. As can be seen, the SA algorithm is the first one able to reach feasible solutions. Moreover, after 70 s of computation it provides the best feasible solutions out of all of them. Regarding TS, notice how it gives worse results than VNS. It could be due to the fact that VNS restarts from new solutions (sometimes very good solutions) every *K* iteration, while TS only applies an

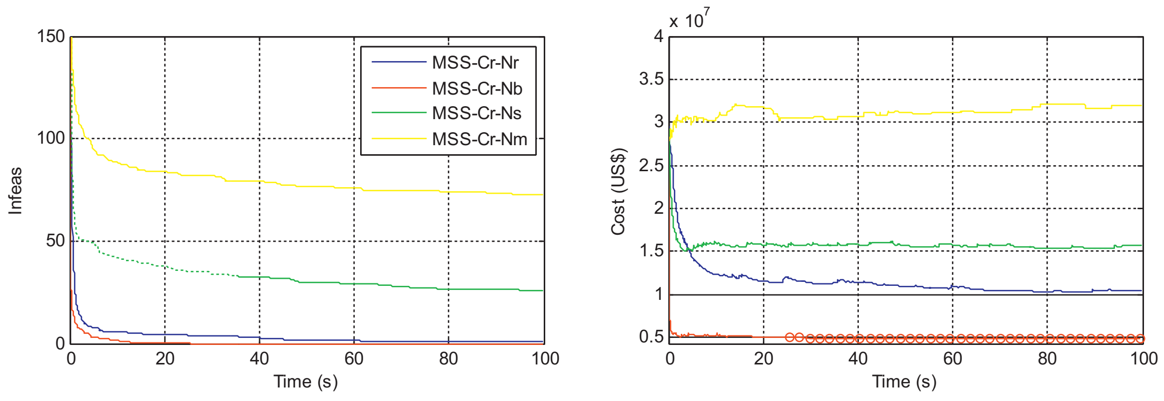


**Fig. 9.** Comparison of all the neighborhood generation methods for the MSS–*Cr*–*N** algorithm.

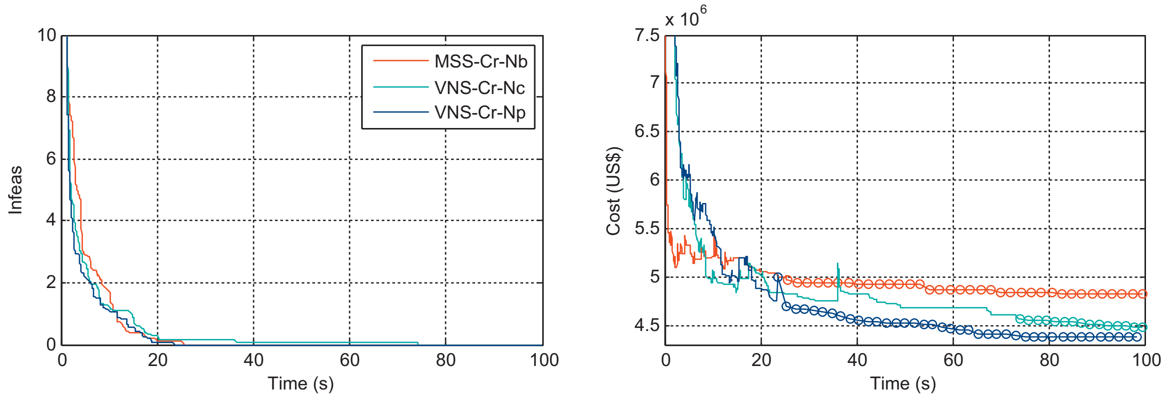


**Fig. 10.** Improvement achieved by using VNS-*Cr*–*Nc*/*Np* instead of MSS–*Cr*–*Nb*.

**Table 3**
Results of all the implementations of TS and SA algorithms after 100 s of computation.

| Implementation | No. of feasible solutions | No. of optimal solutions | Average solution | | Best solution | |
|---|---|---|---|---|---|---|
| | | | Infeas | Cost (US$) | Infeas | Cost (US$) |
| TS–*Cr*–*Np* | 25 | 0 | 0.00 | 4,537,090 | 0.00 | 4,312,950 |
| TS–*Cl*–*Np* | 25 | 2 | 0.00 | 4,365,985 | 0.00 | 4,312,275 |
| TS–*Cq*–*Np* | 25 | 4 | 0.00 | 4,325,143 | 0.00 | 4,312,275 |
| SA–*Cr*–*Np* | 25 | 6 | 0.00 | 4,352,833 | 0.00 | 4,312,275 |
| SA–*Cl*–*Np* | 25 | 12 | 0.00 | 4,312,275 | 0.00 | 4,312,275 |
| SA–*Cq*–*Np* | 25 | 16 | 0.00 | 4,312,595 | 0.00 | 4,312,275 |

escaping movement, which used to be closer to the previous solution. Another cause could be that VNS has a faster implementation than TS since VNS does not have to check the taboo list, thus, being able to explore more solutions than TS along the same CPU time.

Finally, we tested the robustness of the SA algorithm (the best GSM found among all others) by evaluating its performance for five different parameter settings randomly generated around the reference values. Results are shown in Table 4, where the first instance corresponds to these reference values and *Cr* was the selected construction method. The average cost function value among all the instances for all the tests, including the reference one, is 4,416,313 US$ with a standard deviation of 18,511.74 US$ (0.42%). Moreover, several optimal solutions are reached in each case. Fig. 12 also shows the convergence of both terms of the objective function. It can be seen how all the instances are able to reach feasible solutions after 10 s of computation and how the cost function value converges with the optimal one in each case.

### 5.3. Multiple Markov Chain algorithms

As it was discussed in Section 4.1, the probability of not achieving an optimal solution when running SSMC decreases exponentially as the number of Markov chains ($J$) increases, with only a linear increase of the CPU time. Hence SMMC algorithms are a good approach to improve the quality of the solutions reached by the corresponding SSMC algorithms. In order to test the influence of parameter $J$ over the algorithm performance, different SMMC have been run with values from $J=2$ to $J=6$. This analysis was only done for SA, since it was the best SSMC algorithm found among all those proposed in this paper. Table 5 shows the best solutions found by each instance of SMMC–SA after 500 s of computation using the reference parameters $K=1000$, $\alpha=0.990$ and $T_o=5\times10^6$. As can be seen, the number of instances which reach the optimal solution improves as $J$ increases; however, as it is shown in Fig. 13(a), it results in a slower convergence since each SSMC runs less number of iterations on the same computational time. These results show how the convergence behavior and the robustness of the algorithm are improved by using multiple short runs instead of one long single run with the same total computational time. However, it is necessary to have enough CPU time to let the SMMC–SA converge to good quality solutions. Fig. 13(b) shows how the SMMC–SA algorithm needs more than 100 s of computation to reach better results than SSMC–SA for $J=2$, more than 200 s if $J=3$, more than 300 if $J=4$, etc. Notice how the SMMC–SA algorithm is able to reach 25 optimal

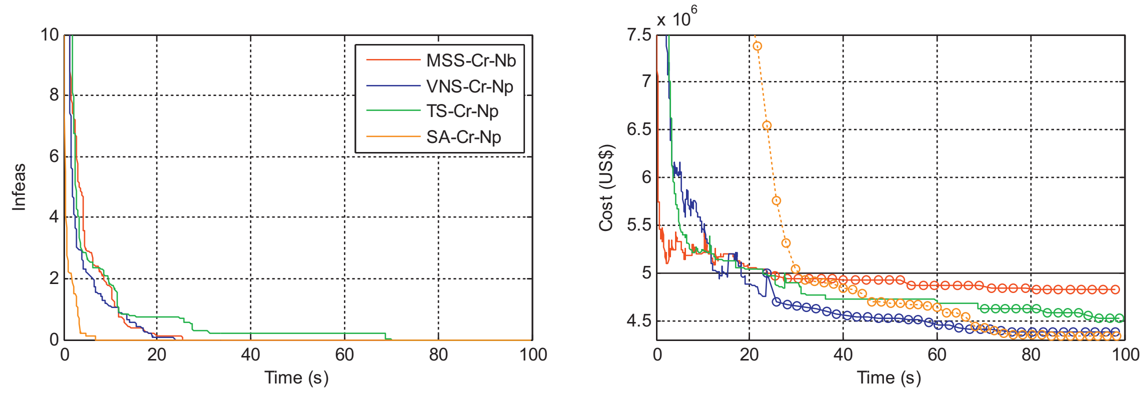


**Fig. 11.** Convergence of all the proposed GSMs when using *Cr* as construction method.

**Table 4**
Results of five different test cases of SA algorithm after 100 s of computation.

| Parameter | Reference | Test no. 1 | Test no. 2 | Test no. 3 | Test no. 4 | Test no. 5 |
|---|---|---|---|---|---|---|
| $K$ | 1000 | 119% | 80% | 134% | 77% | 145% |
| $T_o$ | 1,000,000 | 70% | 107% | 121% | 140% | 80% |
| $\alpha$ | 0.990 | 99% | 98% | 102% | 95% | 102% |
| Average cost (US$) | 4,352,833 | 4,393,563 | 4,361,061 | 4,392,505 | 4,356,250 | 4,361,175 |
| No. of optimal solutions | 6 | 6 | 5 | 7 | 8 | 6 |

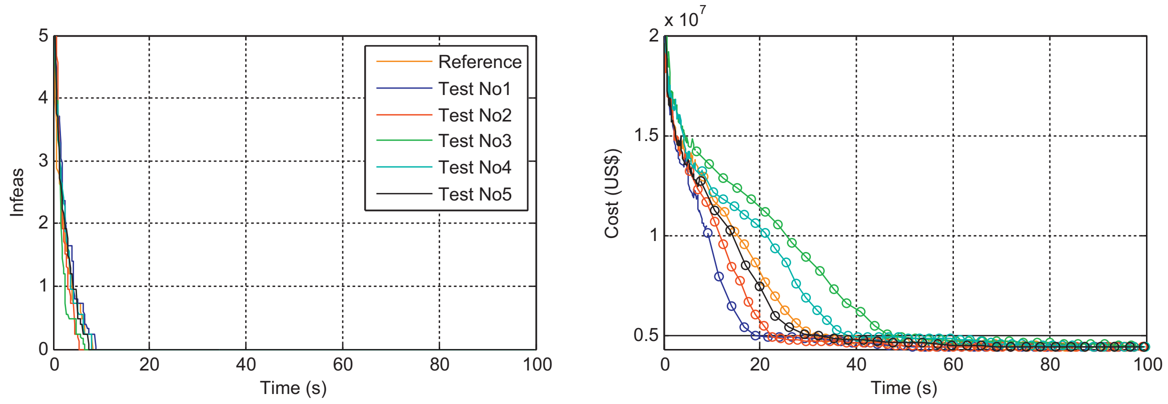


**Fig. 12.** Robustness of the developed SA algorithm.

**Table 5**
Results of SMMC–SA for different values of $J$ after 500 s of computation.

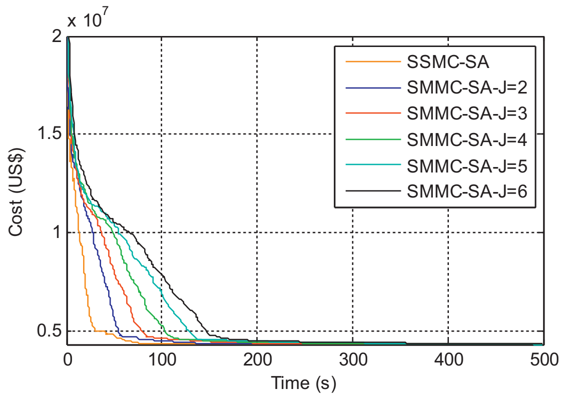
| Implementation | No. of feasible solutions | No. of optimal solutions | Average solution | | Best solution | |
|---|---|---|---|---|---|---|
| | | | Infeas | Cost (US$) | Infeas | Cost (US$) |
| SSMC–SA | 25 | 10 | 0.00 | 4,351,813 | 0.00 | 4,312,275 |
| SMMC–SA–$J=2$ | 25 | 15 | 0.00 | 4,337,540 | 0.00 | 4,312,275 |
| SMMC–SA–$J=3$ | 25 | 19 | 0.00 | 4,327,275 | 0.00 | 4,312,275 |
| SMMC–SA–$J=4$ | 25 | 23 | 0.00 | 4,317,315 | 0.00 | 4,312,275 |
| SMMC–SA–$J=5$ | 25 | 25 | 0.00 | 4,312,275 | 0.00 | 4,312,275 |
| SMMC–SA–$J=6$ | 25 | 3 | 0.00 | 4,313,362 | 0.00 | 4,312,275 |

solutions for $J=5$ but not for $J=6$. Thus, if $J \times$ *time* is very large for better results, the time consumption can be a problem limiting the use of SMMC–SA. This problem may constrain SMMC–SA to work on a rather small number of Markov chains and/or require shortening the run length of each individual Markov chain. However, using fewer number of Markov chains or shortening the run length of the individual Markov chain can decrease the effectiveness of the algorithm. This limitation can be solved using parallel computing by distributing the $J$ Markov chains to $P$ concurrently available processors running $J/P$ Markov chains each in a PMMC–SA algorithm, and where each one uses a separate pseudorandom number generator to explore different areas of the search space.

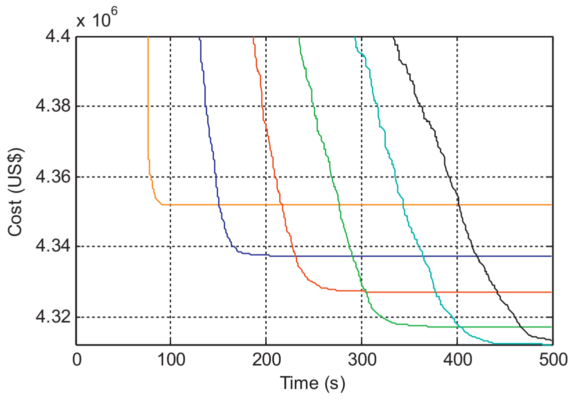
Finally, the performance of DMMC–SA has been evaluated for $J=6$ with $N=3$, leading to $J_n=2$ Markov chains running on each SMMC–SA algorithm. Notice that there are two other different configurations; i.e. $(N=2, J_n=3)$ and $(N=6, J_n=1)$, but after testing them we saw that they gave worse results than the selected one. Again, the cooling schedule is given by a temperature change performed every $K=1000$ iterations, linearly decreasing from $T_o=5\times10^6$ with $\alpha=0.990$. The performance of this distributed SA was evaluated and compared with a unique SMMC–SA running $J=6$ Markov chains independently. Results are shown in Table 6, where the DMMC–SA was run for a value of $R$ (number of iterations without communication among the $N$ SMMC–SA algorithms) equal to 50 and for the three different topologies described in Section 4.2. As can be seen, the number of optimal solutions increases by allowing the communication between $N=3$ different SMMC–SA with $J_n=2$ running in parallel instead of having a single SMMC–SA with $J=6$. Moreover, the best migration strategy seems to be *Master Slave* since it is able to not only increase the number of optimal solutions, but also decrease the average cost function value. Finally, notice how having $P=3$ available processors, the parallel version of this algorithm (PDMMC–SA) would decrease the CPU time needed to reach the same solution to $T_{\text{DMMC–SA}}/3$.

**Table 6**
Results of DMMC–SA–($N=3$,$J_n=2$) for different migration strategies after 500 s of computation.

| Implementation | No. of feasible solutions | No. of optimal solutions | Average solution | | Best solution | |
|---|---|---|---|---|---|---|
| | | | Infeas | Cost (US$) | Infeas | Cost (US$) |
| SMMC–SA–$J=6$ | 25 | 3 | 0.00 | 4,313,362 | 0.00 | 4,312,275 |
| DMMC–SA–RI | 25 | 15 | 0.00 | 4,393,010 | 0.00 | 4,312,275 |
| DMMC–SA–FC | 25 | 10 | 0.00 | 4,313,233 | 0.00 | 4,312,275 |
| DMMC–SA–MS | 25 | 15 | 0.00 | 4,313,102 | 0.00 | 4,312,275 |



**(a)** Convergence



**(b)** Expanded graph of (a)

**Fig. 13.** Convergence of SMMC–SA for different values of *J*.

**Table 7**
CPU time (s) needed by each algorithm to reach the optimal solution in all scenarios.

| Solver | *Scenario* I | *Scenario* II | *Scenario* III |
|---|---|---|---|
| CPLEX–MILP complete model | 19,475 | 9227 | 15,630 |
| CPLEX–MILP simplified model | × | 889 | 1685 |
| DMMC–MSS–*Cq*–*Nb*–($N$ = 5,$J_n$ = 5)–MS | 1245 | 943 | 748 |
| DMMC–VNS–*Cq*–*Np*–($N$ = 5,$J_n$ = 5)–MS | 842 | 854 | 645 |
| DMMC–TS–*Cq*–*Np*–($N$ = 5,$J_n$ = 5)–MS | 547 | 763 | 565 |
| DMMC–SA–*Cq*–*Np*–($N$ = 5,$J_n$ = 5)–MS | 234 | 312 | 258 |

× = Optimal solution was not found.

## 5.4. Comparison against CPLEX under several scenarios

Usually, at the completion time of the current planning horizon it moves forward and a rescheduling process based on updated problem data is triggered again over the new horizon. Obviously, the algorithm employed for this rescheduling process has to be fast enough to reach the optimal solution before the beginning of the new planning horizon. As it was said above, under certain scenarios the *simplified model* proposed in Herrán et al. (2010) can be used instead of the *complete* one in order to reduce the CPU time needed to reach the optimal solution. Such scenarios are those where the fulfillment of depot demands requires pump operations during the entire planning horizon or those where the pumping cost is very high to avoid stoppage operations. However, the *complete model* should always be used, since it is able to reach the optimal solution in any situation, as opposed to the *simplified* one which could lead to sub-optimal solutions in some scenarios. In these cases, the GSMs proposed in this paper, which always evaluate the objective function over the *complete model*, give a good alternative to the MILP algorithm. In order to show this, the performance of each GSM is evaluated against CPLEX for all the scenarios presented in Herrán et al. (2010).

Table 7 shows the CPU time needed to reach the optimal solution in each scenario by solving both models (*complete* and *simplified*) with CPLEX together with CPU time needed by the best implementation of each GSM using the *complete model*. After the analysis made in the previous section, it resulted that SMMC–$J$ = 5 is the best implementation of the SA algorithm for *Scenario* II, since it is always able to reach an optimal solution in less than 500 s for this scenario. However, after doing a similar analysis for all the scenarios and GSMs it resulted that, in general, the best GSM implementation for that problem is DMMC–($N$ = 5,$J_n$ = 5)–MS. As can be seen from Table 7, most of the GSMs reduce the CPU time required by CPLEX in running the *complete model* in more than one order of magnitude. Moreover, even in for *Scenarios* II and III, where the *simplified model* can be used, several GSMs perform better results than CPLEX.

## 5.5. Solving a larger scenario

Finally, the performance of each GSM is evaluated against CPLEX under a larger scale problem than the problems seen in the previous scenarios. This problem (*Scenario* IV) considers the same network as before, but with a larger planning horizon composed by $|T| = 30$ periods of $\Delta = 5$ h length each. Data for this problem are the same as for *Scenario* I but increasing the demand at all the destinations to 13 *VP*m$^3$ for all the products. Moreover, two additional production runs were scheduled at each refinery to cover the additional periods from 21 to 30. They were similar to the scheduled productions from periods 1 to 10 used in previous scenarios. The complete model for this larger scenario has 7971 variables and 19,780 constraints. Fig. 14 shows the convergence of the GSMs implementations shown in Table 7 along 5 h of computation together with the best cost function value found by CPLEX after 40 h of computation. As can be seen, all the GSMs are able to improve the best solution found by CPLEX within the 5 h these GSMs were run.

Since no improvement was met by all the GSMs after 5 h of computation, the solution obtained from these GSMs was used as an initial guess for the MILP formulation. Results can be found in Table 8. This table shows the lower bound and the best cost function value found by CPLEX after 40 h of computation together with the convergence of the implementations shown in Table 7 for all the GSMs. As can be seen, SA is able to reach the optimal solution. This optimality was guaranteed by CPLEX when introducing the best known SA solution as an initial guess for the MILP formulation. Finally, it can also be seen how the solution given by CPLEX

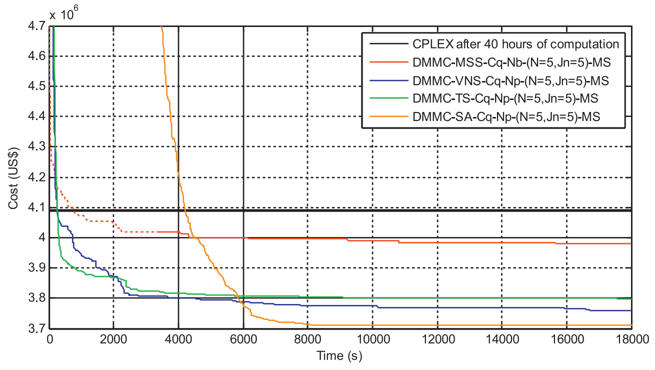


**Fig. 14.** Comparison of all GSMs against CPLEX for *Scenario* IV.

**Table 8**
Performance of all the GSMs against CPLEX for *Scenario* IV.

| Time hh:mm:ss | Lower bound | CPLEX Cost (US$) | MSS Cost (US$) | VNS Cost (US$) | TS Cost (US$) | SA Cost (US$) |
|---|---|---|---|---|---|---|
| 00:00:05 | * | × | 5,520,050 | 7,872,250 | 7,502,200 | 10,546,550 |
| 00:03:00 | 2,920,140.1 | × | 4,235,275 | 4,362,450 | 4,489,750 | 9,453,175 |
| 00:05:00 | 2,943,984.7 | × | 4,180,875 | 4,090,325 | **4,021,375** | 9,000,050 |
| 00:10:00 | 2,962,482.6 | × | 4,115,550 | **4,061,800** | **3,917,825** | 8,765,875 |
| 00:50:00 | 3,084,452.2 | 8,378,150 | **4,040,625** | **3,840,250** | **3,835,475** | 6,018,475 |
| 02:00:00 | 3,135,627.1 | 6,110,500 | **4,002,300** | **3,815,175** | **3,820,425** | **3,723,650** |
| 05:00:00 | 3,215,547.7 | 4,875,525 | ***3,981,675*** | ***3,759,375*** | ***3,797,225*** | ***3,710,575*** |
| 10:00:00 | 3,239,816.8 | 4,768,475 | ***3,961,050*** | ***3,743,875*** | ***3,781,525*** | ● |
| 15:00:00 | 3,263,188.0 | 4,445,750 | ***3,916,175*** | ***3,741,500*** | ***3,770,725*** | ● |
| 20:00:00 | 3,308,159.0 | 4,197,825 | ***3,773,500*** | ***3,740,250*** | ***3,768,025*** | ● |
| 40:00:00 | 3,323,242.7 | 4,090,050 | ***3,762,300*** | ***3,736,550*** | ***3,736,025*** | ● |
| Improvement on solution quality (%) | | | 8.01% | 8.64% | 8.66% | |

× = Feasible solution was not found; ● = optimal solution was found; **Bold** = the solution quality improves the best CPLEX solution after 40 h of computation. ***Italic*** = warm start used by CPLEX as a post-optimization algorithm. ***Italic*** = CPLEX used as a post-optimization algorithm.

after 40 h of computation can be improved in more than 8% by supplying a MSS, VNS or a TS solution as a warm start for the MILP formulation used by CPLEX.

## 6. Conclusions

The metaheuristic optimization methods described in this paper provide an easy way for solving a combinatorial optimization problem; namely, planning transportation of multiple petroleum products in a multi-pipeline system. These methods are an alternative to the MILP algorithm for large scale problems. All the metaheuristics presented in this paper are combinations of improvement methods applied to solutions resulting from different construction heuristics: purely random (*Cr*), based on the solution of a linear relaxation of the original MILP model (*Cl*), and based on the solution of a quadratic relaxation (*Cq*). The improvements are performed by searching the neighborhoods generated around the current solution by four different methods, replace (*Nr*), batch (*Nb*), swap (*Ns*) and move (*Nm*), or a combination of them (*Nc* and *Np*). All these construction and neighborhood generation methods were specifically designed by using the knowledge of the problem to be solved. This makes the proposed metaheuristic optimization methods converge at an acceptable speed for a good quality solution.

The application example proposed in this paper was solved by four different GSMs: MSS, VNS, TS and SA. The best results were achieved by SA using *Cq* as the method to start the search and *Np* for the generation of neighbors around the current solution. This is due to the fact that SA algorithms can accept worse solutions than the current one in a stochastic search, avoiding the premature convergence to local minima. The performance of the basic SA algorithm was improved by a Sequential Multiple Markov Chain SA performing *J* independent versions of SA on a single processor computer, using the same search space, neighborhood generation and cooling schedule. Results showed how the convergence behavior and the robustness of SA can be improved by using multiple short runs instead of one long single run with the same total CPU time. Moreover, a distributed version of this algorithm was implemented to further improve the performance given by the SMMC. After analyzing the performance of each single algorithm, a comparison of the best implementation of each GSM against CPLEX was presented for all the scenarios solved in Herrán et al. (2010). The best GSM implementation for the problem under study was a DMMC with $N = 5$ SMMCs running $J_n = 5$ Markov chains each with a Master Slave migration strategy at every $R = 50$ major iterations. Results showed how all these algorithms are able to reduce the time needed to reach the optimal solution by the MILP algorithm run by CPLEX in more than two orders of magnitude.

Finally, the performance of each metaheuristic was evaluated against CPLEX under a larger scale scenario than those proposed in Herrán et al. (2010), leading to solutions that CPLEX could not find after several hours of computation. For this reason, the authors are currently working on developing other metaheuristic algorithms, as population based, to efficiently solve the same problem here proposed.

## Acknowledgments

The authors would like to thank the Spanish Science and Technology Ministry for their support through project DPI2002-02924 and Madrid Autonomous Community through project S-0505/DPI/0391.

## Appendix A. Supplementary data

Supplementary data associated with this article can be found, in the online version, at doi:10.1016/j.compchemeng.2011.10.003.

## References

Alba, E. & Troya, J. M. (2000). Influence of the migration policy in parallel distributed GAs with structured and panmictic populations. *Applied Intelligence*, *12*, 163–181.

Aarts, E. & Lenstra, J. K. (2003). *Local Search in combinatorial optimization*. Princeton University Press.

Altiparmak, F., Gen, M., Lin, L. & Paksoy, T. (2006). A genetic algorithm approach for multi-objective optimization of supply chain networks. *Computers and Chemical Engineering*, *51*, 196–215.

Azencott, R. (1992). Sequential simulated annealing: speed of convergence and acceleration techniques. In R. Azencott (Ed.), *Simulated annealing: Parallelization techniques* (pp. 1–9). New York: John Wiley and Sons.

Bazaraa, M. S., Jarvis, J. J. & Sherali, H. D. (1990). *Linear programming and network flows*. New York: Wiley., 394–418

Cafaro, D. C. & Cerdá, J. (2004). Optimal scheduling of multiproduct pipeline systems using a nondiscrete MILP formulation. *Computers and Chemical Engineering*, *28*, 2053–2068.

Cafaro, D. C. & Cerdá, J. (2008). Dynamic scheduling of multiproduct pipelines with multiple delivery due dates. *Computers and Chemical Engineering*, *32*, 728–753.

Chen, C.-L. & Lee, W.-C. (2004). Multi-objective optimization of multi-echelon supply chain networks with uncertain product demands and prices. *Computers and Chemical Engineering*, *28*, 1131–1144.

Cruz, J. M., Andrés, B., Herrán, A., Besada, E. & Fernández, P. (2003). Multiobjective optimization of the transport in oil pipelines networks. In *9th IEEE international conference on emerging technologies and factory automation (Vol. 1)* (pp. 566–573).

Cruz, J. M., Risco, J. L., Herrán, A. & Fernández, P. (2004). Hybrid heuristic and mathematical programming in oil pipelines networks. In *Congress on evolutionary computation. CEC2004 (Vol. 1)* Portland Marriot Downtown, Portland, USA, (pp. 1479–1486).

Cruz, J. M., Herrán, A., Risco, J. L. & Andrés, B. (2005). Hybrid heuristic and mathematical programming in oil pipelines networks: Use of immigrants. *Journal of Zhejiang University Science*, *6A*(1), 9–19.

Defersha, F. M. & Chen, M. (2008). A parallel multiple Markov chain simulated annealing for multi-period manufacturing cell formation problems. *International Journal of Advanced Manufacturing Technology*, *37*(1–2), 140–156.

Drexl, A. (1997). Lot sizing and scheduling – Survey and extensions. *European Journal of Operations Research*, *99*, 221–235.

Glover, F. & Laguna, M. (1997). *Tabu search*. Kluwer.

Hansen, P. & Mladenovic, N. (1999). An introduction to variable neighborhood search. In S. Voss, S. Voss, et al. (Eds.), *Metaheuristics: Advances and trends in local search paradigms for optimization* (pp. 433–458). Kluwer.

Herrán, A., de la Cruz, J. M. & de Andrés, B. (2010). A mathematical model for planning transportation of multiple petroleum products in a multi pipeline system. *Computers and Chemical Engineering*, *34*, 401–413.

ILOG Inc. (2006). *ILOG OPL Studio 4.2 users manual*. 1080 Linda Vista Ave. Mountain View, CA 94043. http://www.ilog.com.

Janak, S. L., Lin, X. & Floudas, C. A. (2007). A new robust optimization approach for scheduling under uncertainty: II. Uncertainty with known probability distribution. *Computers and Chemical Engineering*, *31*, 171–195.

Kirkpatrick, S., Gelatt, C. D. & Vecchi, M. P. (1983). Optimization by simulated annealing. *Science*, *220*, 671–680.

Lee, S.-Y. & Lee, K. G. (1996). Synchronous and asynchronous parallel simulated annealing with multiple markov chains. *IEEE Transactions on Parallel and Distributed Systems*, *7*, 903–1007.

Lengauer, T. (1990). *Combinatorial algorithms for integrated circuit layout*. John Wiley and Sons. (pp. 427–446)

Luo, C. & Rong, G. (2007). Hierarchical approach for short-term scheduling in refineries. *Industrial and Engineering Chemical Research*, *46*, 3656–3668.

Martí, R. (2003). Multistart methods. In F. Glover, A. Gary, & Kochenberger (Eds.), *Handbook of metaheuristics* (pp. 355–368). Kluwer Academic.

Metropolis, N., Rosenbluth, A., Rosenbluth, M., Teller, A. & Teller, E. (1953). Equations of state calculations by fast computing machines. *Journal of Chemical Physics*, *21*, 1087–1092.

Milidiú, R. L., Pessoa, A. & Laber, E. (2003). The complexity of makespan minimization for pipeline transportation. *Theoretical Computer Science*, *306*, 339–351.

Mirhassani, S. A. & Ghorbanalizadeh, M. (2008). The multiproduct pipeline scheduling system. *Computers and Mathematics with Applications*, *56*(4), 891–897.

Pereira, A., Carvalho, F., Pedroso, J. & Constantino, M. (2003). Iterated local search and Tabu search for a discrete lot sizing and scheduling problem. In *META-HEURISTICS: Computer decision-making (combinatorial optimization book series)*. The Netherlands: Kluwer Academic Publishers.

Rejowski, R., Jr. & Pinto, J. M. (2003). Scheduling of a multiproduct pipeline system. *Computers and Chemical Engineering*, *27*, 1229–1246.

Rejowski, R., Jr. & Pinto, J. M. (2004). Efficient MILP formulations and valid cuts for multiproduct pipeline scheduling. *Computers and Chemical Engineering*, *27*, 1511–1528.

Rejowski, R., Jr. & Pinto, J. M. (2008). A novel continuous time representation for the scheduling of pipeline systems with pumping yield rate constraints. *Computers and Chemical Engineering*, *32*, 1042–1066.

Relvas, S., Barbosa-Póvoa, A. P. F. D. & Matos, H. A. (2010). Heuristic batch sequencing on a multiproduct oil distribution system. *Computers and Chemical Engineering*, *33*, 712–730.

Sasikumar, M., Prakash, P. R., Patil, S. M. & Ramani, S. (1997). PIPES: A heuristic search model for pipeline schedule generation. *Knowledge-Based Systems*, *10*, 169–175.

Schrijver, A. (1986). *Theory of linear and integer programming*. Chichester: Wiley.

Yagiura, M. & Ibaraki, T. (2002). Local search. In P. M. Pardalos, & M. G. C. Resende (Eds.), *Handbook of applied optimization* (pp. 104–123). Oxford University Press.