INFORMS Journal on Computing
Vol. 12, No. 3, Summer 2000

0899-1499/100/1203-0237 $05.00
© 2000 INFORMS

An Ant Colony System Hybridized with a New Local Search for the Sequential Ordering Problem

LUCA MARIA GAMBARDELLA / *IDSIA, Galleria 2, CH-6928 Manno-Lugano, Switzerland, Email: luca@idsia.ch, Web: http://www.idsia.ch/~luca*

MARCO DORIGO / *IRIDIA, CP 194/6, Université Libre de Bruxelles, Ave. F. Roosevelt 50, B-1050 Brussels, Belgium, Email: mdorigo@ulb.ac.be, Web: http://iridia.ulb.ac.be/~mdorigo*

(Received: December 1997; revised: June 1999; accepted: March 2000)

We present a new local optimizer called SOP-3-exchange for the sequential ordering problem that extends a local search for the traveling salesman problem to handle multiple constraints directly without increasing computational complexity. An algorithm that combines the SOP-3-exchange with an Ant Colony Optimization algorithm is described, and we present experimental evidence that the resulting algorithm is more effective than existing methods for the problem. The best-known results for many of a standard test set of 22 problems are improved using the SOP-3-exchange with our Ant Colony Optimization algorithm or in combination with the MPO/AI algorithm (Chen and Smith 1996).

There are many NP-hard combinatorial optimization problems for which it is impractical to find an optimal solution. Among them is the sequential ordering problem (SOP). For such problems the only reasonable enterprise is to look for heuristic algorithms that quickly produce good, although not necessarily optimal, solutions. These algorithms often use some problem-specific knowledge either to build or to improve solutions. Recently, many researchers have focused their attention on a new class of algorithms called *metaheuristics*. Metaheuristics are rather general algorithmic frameworks that can be applied to several different optimization problems with few modifications. Examples of metaheuristics are simulated annealing (Kirkpatrick et al. 1983), evolutionary computation (Holland 1975; Fogel 1994), and tabu search (Glover 1989a,b). Metaheuristics are often inspired by natural processes. In fact, the above-cited metaheuristics were inspired, respectively, by the physical annealing process, the Darwinian evolutionary process, and the clever management of memory structures. One of the most recent nature-inspired metaheuristics is Ant Colony Optimization (Dorigo and Di Caro 1999; Dorigo et al. 1999; Dorigo et al. 1991; Dorigo et al. 1996; Dorigo 1992). There, the inspiring natural process is the foraging behavior of ants. Ant Colony System (ACS), a particular instance of Ant Colony Optimization (ACO), has recently been shown (Gambardella and Dorigo 1996; Dorigo and Gambardella 1997) to be competitive with other metaheuristics on the symmetric and asymmetric traveling salesman problems (TSP and ATSP). Although this is an interesting and promising result, it remains clear that Ant Colony Optimization, as well as other metaheuristics, in many cases cannot compete with specialized local search methods. A current trend (Johnson and McGeoch 1997) is therefore to associate with the metaheuristic a local optimizer, giving birth to so-called *hybrid methods*. This is an interesting marriage since local optimizers often suffer from the *initialization problem*. That is, the performance of a local optimizer is often a function of the initial solution to which it is applied. For example, multistart, that is, the application of local search to different, randomly generated, initial solutions has been found to be a poor choice, since the local search procedure spends most of its time improving the initial low-quality solution (Aarts and Lenstra 1997). Therefore, it becomes interesting to find good metaheuristic-local optimizer couplings, where a coupling is good if the metaheuristic generates initial solutions that can be carried to very good local optima by the local optimizer.

In previous work we have shown that local search plays an important role in ACO. For example, Dorigo and Gambardella (1997) have applied ACS with an extended version of the 3-opt local search to symmetric and asymmetric TSPs obtaining very good results. Also, Gambardella et al. (1999a) have proposed MACS-VRPTW (Multiple ACS for the Vehicle Routing Problem with Time Windows), a hierarchically organized ACS in which two different colonies successively optimize a multiple objective function exploiting, among other things, local search. MACS-VRPTW has been shown to be competitive with the best-known existing methods in terms of both solution quality and computation and has been able to improve some of the best-known solutions for a number of problem instances in the literature.

Finally, Gambardella et al. (1999b) have recently applied HAS-QAP, an ACO-related algorithm that uses a simple form of local search, to the quadratic assignment problem (QAP), obtaining solutions that are better than those obtained by the best-known algorithms on structured, real-world problem instances; comparison includes reactive tabu search (Battiti and Tecchiolli 1994), robust tabu search (Taillard 1991), simulated annealing (Connolly 1990), and genetic hybrid search (Fleurent and Ferland 1994).

Subject classifications: Metaheuristics, combinatorial optimization, artificial intelligence, production planning.
Other key words: Ant colony optimization, ant algorithms, metaheuristics, sequential ordering problem, swarm intelligence.

In this paper we attack the SOP by an ACO algorithm coupled with SOP-3-exchange, a novel local search procedure specifically designed for the SOP. The resulting Hybrid Ant System for the SOP (HAS-SOP; Gambardella and Dorigo 1997) outperforms all known heuristic approaches to the SOP (code and up-to-date information are maintained at http://www.idsia.ch/luca/has-sop.html). Also, we have been able to improve many of the best results published in TSPLIB (the TSPLIB can be accessed at http://softlib.rice.edu/softlib/tsplib/), one of the most important databases of difficult TSP-related optimization problems available on the Internet.

1. The Sequential Ordering Problem

The sequential ordering problem with precedence constraints (SOP) was first formulated by Escudero (1988) to design heuristics for a production planning system. It consists of finding a minimum weight Hamiltonian path on a directed graph with weights on the arcs and the nodes, subject to precedence constraints among nodes.

1.1 Problem Definition

Consider a complete graph G = (V, A) with node set V and arc set A, where nodes correspond to jobs $0, \ldots, i, \ldots, n$ ($n + 1 = |V|$). A cost $t_{ij} \in \Re$, with $t_{ij} \geqslant 0$, is associated to each arc (i, j). This cost represents the waiting time between the end of job i and the beginning of job j. A cost $p_i \in \Re$, $p_i \geqslant 0$, representing the processing time of job i, is associated with each node i. The set of nodes V includes a starting node (node 0) and a final node (node n) connected with all the other nodes. The costs between node 0 and the other nodes are equal to the setup time of node i, $t_{0i} = p_i\ \forall i$, and $t_{in} = 0$ $\forall i$. Precedence constraints are given by an additional acyclic digraph P = (V, R) defined on the same node set V. An arc $(i, j) \in$ R if job i has to precede job j in any feasible solution. R has the transitive property (that is, if $(i, j) \in$ R and $(j, k) \in$ R then $(i, k) \in$ R). Since a sequence always starts at node 0 and ends at node n, $(0, i) \in$ R $\forall i \in$ V\{0}, and $(i, n) \in$ R $\forall i \in$ V\{n}. In the following we will indicate with predecessor[i] and successor[i] the sets of nodes that have to precede/succeed node i in any feasible solution.

Given the above definitions, the SOP can be stated as the problem of finding a job sequence that minimizes the total makespan subject to the precedence constraints. This is therefore equivalent to the problem of finding a feasible Hamiltonian path with minimal cost in G under precedence constraints given by P.

The SOP can also be formulated as a general case of the ATSP by giving only the weights on the edges (in the SOP a solution connects the first and the last node by a path that visits all nodes once, as opposed to the ATSP in which a solution is a closed tour that visits all nodes once). This formulation is equivalent to the previous: it suffices to remove weights from nodes and to redefine the weight c_{ij} of arc (i, j) by adding the weight p_j of node j to each t_{ij}. In this representation c_{ij} is an arc weight (where c_{ij} may be different from c_{ji}), which can either represent the cost of arc (i, j) when $c_{ij} \geq 0$, or an ordering constraint when $c_{ij} = -1$ ($c_{ij} = -1$ means that element j must precede, not necessarily immediately, element i). In this paper we will use this last formulation.

1.2 Heuristic Methods for the SOP

The SOP models real-world problems like production planning (Escudero 1988), single-vehicle routing problems with pick-up and delivery constraints (Pulleyblank and Timlin 1991; Savelsbergh 1990), and transportation problems in flexible manufacturing systems (Ascheuer 1995).

The SOP can be seen as a general case of both the asymmetric TSP and the pick-up and delivery problem. It differs from ATSP because the first and the last nodes are fixed, and in the additional set of precedence constraints on the order in which nodes must be visited. It differs from the pick-up and delivery problem because this is usually based on symmetric TSPs and because the pick-up and delivery problem includes a set of constraints between nodes with a unique predecessor defined for each node, in contrast to the SOP where multiple precedences can be defined.

1.2.1 Approaches Based on the ATSP

Sequential ordering problems were initially solved as constrained versions of the ATSP. The main effort has been put into extending the mathematical definition of the ATSP by introducing new equations to model the additional constraints. The first mathematical model for the SOP was introduced in Ascheuer et al. (1993) where a cutting-plane approach was proposed to compute lower bounds on the optimal solution. In Escudero et al. (1994), a Lagrangian relax-and-cut method was described, and new valid cuts to obtain strong lower bounds were defined. More recently, Ascheuer (1995) has proposed a new class of valid inequalities and has described a branch-and-cut algorithm for a broad class of SOP instances based on the polyhedral investigation carried out on ATSP problems with precedence constraints by Balas et al. (1995). His approach also investigates the possibility to compute and improve sub-optimal feasible problem solutions starting from the upper bound computed by the polyhedral investigation. The upper bound is the initial solution of a heuristic phase based on well-known ATSP heuristics that are iteratively applied in order to improve feasible solutions. These heuristics do not handle constraints directly: infeasible solutions are simply rejected. With this approach Ascheuer was able to compute new upper bounds for the SOP instances in TSPLIB, although a genetic algorithm called Maximum Partial Order/Arbitrary Insertion (MPO/AI), recently proposed by Chen and Smith (1996), seems to work better on the same class of problems. MPO/AI always works in the space of feasible solutions by introducing a sophisticated crossover operator that preserves the common schema of two parents by identifying their maximum partial order through matrix operations. The new solution is completed using a constructive heuristic.

1.2.2 Approaches Based on the Pick-up and Delivery Problem

Heuristic approaches to pick-up and delivery problems are based on particular extensions of TSP heuristics able to handle precedence constraints while improving feasible solutions without any increase in computation times. Psaraftis

```
procedure ACO metaheuristic (for static combinatorial problems)
   Initialize the pheromone trail, set parameters
   while (end_condition = false)
      Build_solutions
      Apply_local_optimizer  \* Optional *\
      Update_pheromone_trails
   end-while
```

Figure 1. The ACO metaheuristic for static combinatorial optimization problems.

(1983) has introduced a preprocessing technique to ensure feasibility checking in constant time by starting the algorithm with a screening procedure that, at an initial cost of $O(n^2)$, produces a feasibility matrix that contains information about feasible edge exchanges. Subsequently, Solomon (1987) proposed a search procedure based on a tailored updating mechanism, while Savelsbergh (1990), Van der Bruggen et al. (1993), and Kindervater and Savelsbergh (1997) presented a lexicographic search strategy, a variation of traditional edge-exchange TSP heuristics, that reduces the number of visited nodes without losing any feasible exchange. In order to ensure constraint checking in constant time, the lexicographic search strategy has been combined with a labeling procedure where nodes in the sequence are labeled with information related to their unique predecessor/successor, and a set of global variables are updated to keep this information valid. Savelsbergh (1990) presented a lexicographic search based on 2-opt and 3-opt strategies that exchanges a fixed number of edges, while Van der Bruggen et al. (1993) proposed a variable-depth search based on the Lin and Kernighan (1973) approach. Unfortunately, this labeling procedure is not applicable in the case of multiple precedence constraints because it requires that nodes in the sequence have a unique predecessor/successor. On the other hand, the lexicographic search strategy itself is independent of the number of precedence constraints and can therefore be used to solve sequential ordering problems where multiple precedence constraints are allowed.

The approach to the SOP presented in this paper is the first in the literature that uses an extension of a TSP heuristic to handle directly multiple constraints without any increase in computational time. Our approach combines a constructive phase based on the ACS algorithm (Dorigo and Gambardella 1997) with a new local search procedure called SOP-3-exchange. SOP-3-exchange is based on a lexicographic search heuristic due to Savelsbergh (1990) and a new labeling procedure able to handle multiple precedence constraints. In addition, we test and compare different methods to select nodes during the search and different stopping criteria. In particular we test two different selection heuristics: one based on the *don't look bit* data structure introduced by Bentley (1992), and the other based on a new data structure called *don't push stack* introduced by the authors.

2. Ant Colony Optimization

The Ant Colony Optimization metaheuristic (Dorigo and Di Caro 1999) is a population-based approach to the solution of discrete optimization problems. It has been applied to both static and dynamic combinatorial optimization problems (static problems are those whose topology and costs do not change while the problems is being solved, while in dynamic problems the topology and the costs can change while solutions are built; for an up-to-date list of ACO papers and applications see http://iridia.ulb.ac.be/~mdorigo/ACO/ACO.html). When applied to static combinatorial optimization problems, it takes the form shown in Fig. 1, where three procedures are iterated until some end condition is verified. First, a set of agents (artificial ants) builds solutions. Then these solutions can be taken to their local optima by the application of a local search procedure. Finally, pheromone trails are updated. Solution construction uses a probabilistic nearest-neighbor algorithm that uses a special distance measure. Once the artificial ants have built a solution, they use the quality of the generated solutions to update the distance information. Let us consider, for presentation purposes, the symmetric TSP, defined as follows: a graph $G = (V, A)$ with node set V and arc set A is given; arcs have a cost associated (e.g., their length), and the problem is to find a minimal-length closed tour that visits all the nodes once and only once. In the ACO approach each edge of the graph has two associated measures: the heuristic desirability η_{ij}, and the pheromone trail τ_{ij}. In the TSP application the heuristic desirability is defined as the inverse of the edge length and never changes for a given problem instance, while the pheromone trail is modified at runtime by ants. Each ant has a starting node and its goal is to build a solution, that is, a complete tour. A tour is built node by node (nodes are the vertices of the graph): when ant k is in node i it chooses to move to node j using a probabilistic rule that favors nodes that are close and connected by edges with a high pheromone trail value. Nodes are always chosen among those not yet visited in order to enforce the construction of feasible solutions. Once all ants have built a complete tour, the local search procedure can be applied (in fact, in most applications the use of local search greatly improves the metaheuristic performance). Then pheromone trail is updated on the edges of the (possibly locally optimized) solutions. The guiding principle is to increase pheromone trail on the edges that belong to short tours. Pheromone trails also evaporate so that memory of the past is gradually lost (this prevents bad initial choices from having a lasting effect on the search process). The metaheuristic, here informally described, can be implemented in many different ways, and details about specific implementation choices for the TSP can be found in

Dorigo et al. (1991, 1996), Dorigo (1992), and Dorigo and Gambardella (1997). The ACO metaheuristic can be adapted to the SOP by letting ants build a path from source to destination while respecting the ordering constraints (this can be achieved by having ants choose not-yet-visited nodes that do not violate any ordering precedence).

The most distinctive feature of ACO is the management of pheromone trails that are used, in conjunction with the objective function, to construct new solutions. Informally, the intensity of pheromone gives a measure of how desirable it is to insert a given element in a solution. Pheromone trails are used for *exploration* and *exploitation*. Exploration concerns the probabilistic choice of the components used to construct a solution: a higher probability is given to elements with a strong pheromone trail. Exploitation is based on the choice of the component that maximizes a blend of pheromone-trail values and partial objective function evaluations.

3. Ant Colony Optimization for the SOP

As said in the previous section, application of an ACO algorithm to a combinatorial optimization problem requires definition of a constructive algorithm and possibly a local search. Accordingly, we have designed a constructive algorithm called ACS-SOP in which a set of artificial ants builds feasible solutions to the SOP, and a local search specialized for the SOP (discussed in the next section) that takes these solutions to their local optimum. The resulting algorithm is called a Hybrid Ant System for the SOP (HAS-SOP).

3.1 ACS-SOP

ACS-SOP is strongly based on Ant Colony System (Gambardella and Dorigo 1996; Dorigo and Gambardella 1997). It differs from ACS in the way the set of feasible nodes is computed and in the setting of one of the algorithm's parameters that is made dependent on the problem dimensions. ACS-SOP implements the constructive phase of HAS-SOP, and its goal is to build feasible solutions for the SOP. It generates feasible solutions with a computational cost of order $O(n^2)$.

Informally, ACS-SOP works as follows. Each ant iteratively starts from node 0 and adds new nodes until all nodes have been visited and node n is reached. When in node i, an ant applies a so-called *transition rule*, that is, it probabilistically chooses the next node j from the set $F(i)$ of feasible nodes. $F(i)$ contains all the nodes j still to be visited and such that all nodes that have to precede j, according to precedence constraints, have already been inserted in the sequence.

The ant chooses, with probability q_0, the node j, $j \in F(i)$, for which the product $\tau_{ij} \cdot \eta_{ij}$ is highest (deterministic rule), while with probability $1 - q_0$ the node j is chosen with a probability given by

$$\underset{j\in F(i)}{p_{ij}} = \tau_{ij} \cdot \eta_{ij} \Big/ \sum_{l\in F(i)} \tau_{il} \cdot \eta_{il}$$

(i.e., nodes connected by edges with higher values of $\tau_{ij} \cdot \eta_{ij}$, $j \in F(i)$, have higher probability of being chosen).

The value q_0 is given by $q_0 = 1 - s/n$; q_0 is based on a parameter s that represents the number of nodes we would like to choose using the probabilistic transition rule. The parameter s allows the system to define q_0 independently of the problem size, so that the expected number of nodes selected with the probabilistic rule is s.

In ACS-SOP only the best ant, that is the ant that built the shortest tour, is allowed to deposit pheromone trail. The rationale is that in this way a "preferred route" is memorized in the pheromone trail matrix, and future ants will use this information to generate new solutions in a neighborhood of this preferred route. The formula used is:

$$\tau_{ij} = (1 - \rho) \cdot \tau_{ij} + \rho / L_{\text{best}} \qquad (1)$$

where L_{best} is the length of the path built by the best ant, that is, the length of the shortest path generated since the beginning of the computation.

Pheromone is also updated during solution building. In this case, however, it is removed from visited edges. In other words, each ant, when moving from node i to node j, applies a pheromone updating rule that causes the amount of pheromone trail on edge (i, j) to decrease.

The rule is:

$$\tau_{ij} = (1 - \varphi) \cdot \tau_{ij} + \varphi \cdot \tau_0 \qquad (2)$$

where τ_0 is the initial value of trails. We found that good values for the algorithm's parameters are $\tau_0 = (\text{FirstSolution} \cdot n)^{-1}$, $\rho = \varphi = 0.1$, $s = 10$, where FirstSolution is the length of the shortest solution generated by the ant colony following the ACS-SOP algorithm without using the pheromone trails. These values are rather robust. Values in the following ranges didn't cause any appreciable change in performance: $0.05 \leqslant \rho, \varphi \leqslant 0.3$, $5 \leqslant s \leqslant 15$. The number of ants in the population was set to 10. The rationale for using formula (2) is that it causes ants to eat away pheromone trail while they build solutions so that a certain variety in generated solutions is assured (if pheromone trail was not consumed by ants, they would tend to generate very similar tours).

The algorithm stops when one of the following conditions becomes true: a fixed number of solutions has been generated, a fixed CPU time has elapsed, or no improvement has been observed during a fixed last number of iterations.

3.2 HAS-SOP

HAS-SOP is ACS-SOP plus local search. Local search is an optional component of ACO algorithms, although it has been shown since early implementations that it can greatly improve the overall performance of the ACO metaheuristic when static combinatorial optimization problems are considered (the first example of ACO algorithm with local search was Ant-Q (Gambardella and Dorigo 1995), which was followed by the more performing ACS (Gambardella and Dorigo 1996; Dorigo and Gambardella 1997)).

In HAS-SOP local search is applied once ants have built their solutions: each solution is carried to its local optimum by an application of the local search routine described in Section 4. Locally optimal solutions are then used to update pheromone trails on arcs, according to the pheromone trail update rule (1).

```
1./* Initialization phase */
   For each pair (r,s) τ(r,s):= τ0 End-for
2./* First step of the iteration */
     For k:=1 to m do
         Let rk be the node where agent k is located
         rk ← 0 /* All ants start from node 0 */
     End-for
3. /* This is the step in which agents build their Hamiltonian paths. The path of agent k is stored in Pathk. */
   For k:=1 to m do
     For i:=1 to n-1 do
         Starting from rk compute the set F(rk) of feasible nodes
         /* F(rk) contains all the nodes j still to be visited and such that
            all nodes that have to precede j have already been inserted in the sequence */
         Choose the next node sk according to the ant transition rule
         Pathk(i) ← (rk,sk)
         τ(rk,sk) ← (1-φ)·τ(rk,sk)+φ·τ0 /* This is (2) */
         rk ← sk /* New node for agent k */
     End-for
   End-for
4. /* In this step, present only in HAS-SOP, the local optimizer is applied to the solutions built by each ant */
   For k:=1 to m do
     Optmized_Pathk ← local_opt_routine(Pathk)
   End-for
5. /* In this step pheromone trails are updated using (1) */
   For k:=1 to m do
     Compute Lk   /*Lk is the length of the path Optmized_Pathk */
   End-for
   Let Lbest be the shortest Lk from beginning and Optmized_Pathbest the corresponding path
   For each edge (r,s) ∈ Optmized_Pathbest
     τ(r,s) ← (1-ρ)·τ(r,s)+ρ/ Lbest /* This is (1) */
   End-for
6. If (End_condition = True)
     then Print Lbest and Optmized_Pathbest
     else goto Step 2
    end-if
```

Figure 2. The ACS-SOP/HAS-SOP algorithm. ACS-SOP differs from HAS-SOP in step 4 (local search) which is present only in HAS-SOP.

In Fig. 2 we give a commented Pascal-like description of the algorithm; the local optimization routine, the optional step 4 of the algorithm in Fig. 2, is described in the next section.

4. Local Search: SOP-3-Exchange

4.1 Edge-Exchange Heuristics

In recent years much research went into defining ad-hoc TSP heuristics (see Johnson and McGeoch 1997 for an overview). They can be classified as *tour-constructive* heuristics and *tour-improvement* heuristics (the latter are also called *local optimization* heuristics). Tour-constructive heuristics usually start by selecting a random node (city) from the set of nodes and then incrementally building a feasible TSP solution by adding new nodes chosen according to some heuristic rule (see Bentley 1992 for an overview). For example, the nearest-neighbor heuristic builds a tour by adding the closest node in terms of distance to the last node inserted in the path. On the other hand, tour-improvement heuristics start from a given tour and attempt to reduce its length by exchanging edges chosen according to some heuristic rule until a local optimum is found (i.e., until no further improvement is possible using the heuristic rule). It has been experimentally shown (Reinelt 1994) that, in general, tour-improvement heuristics produce better quality results than tour-constructive heuristics. Still, a tour-constructive heuristic is necessary at least to build the initial solution for the tour-improvement heuristic.

Starting from an initial solution, an edge-exchange procedure generates a new solution by replacing k edges with another set of k edges. This operation is usually called a *k-exchange* and is iteratively executed until no additional improving *k-exchange* is possible. When this is the case the final solution is said to be *k-optimal*; the verification of *k-optimality* requires $O(n^k)$ time. For a *k-exchange* procedure to be efficient, it is necessary that the improving criterion for new solutions can be computed in constant time.

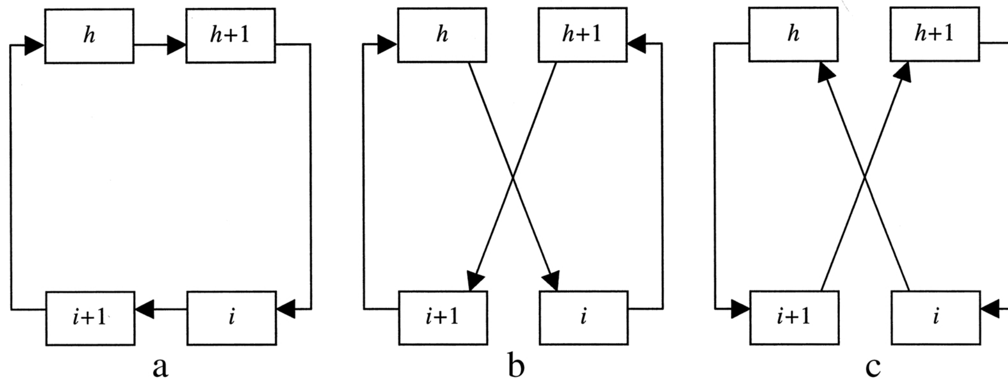

Figure 3. A 2-exchange always inverts a path.

It has been shown that increasing k produces solutions of increasing quality, but the computational effort to test completely the *k-exchange* set for a given solution usually restricts our attention to *k-exchange* with $k \leqslant 3$. The most widely used edge-exchange procedures set k to 2 or 3 (2-opt and 3-opt edge-exchange procedures; Lin 1965) or to a variable value (Lin and Kernighan 1973), in which case a variable-depth edge-exchange search is performed.

In this section we first make some observations about edge-exchange techniques for TSP/ATSP problems. Then, we concentrate our attention on *path-preserving-edge-exchanges* for ATSPs, that is, edge exchanges that do not invert the order in which paths are visited. Next, we discuss *lexicographic-path-preserving-edge-exchange*, a path-preserving-edge-exchange procedure that searches only in the space of feasible exchanges. We then add to the lexicographic-path-preserving-edge-exchange a labeling procedure whose function is to check feasibility in constant time. Finally, we present different possible strategies to select nodes during the search, as well as different search stopping criteria.

4.2 Path-Preserving Edge-Exchange Heuristics

We remind the reader that the SOP can be formulated as a general case of the asymmetric traveling salesman problem (ATSP) in which a solution connects the first and the last node by a path that visits all nodes once, as opposed to the ATSP in which a solution is a closed tour that visits all nodes once. Edge-exchange techniques for TSP/ATSP problems are therefore directly relevant for the SOP. A *k-exchange* deletes k edges from the initial solution creating k disjointed paths that are reconnected with k new edges. In some situations this operation requires an inversion in the order in which nodes are visited within one of the paths (*path-inverting-edge-exchange*), while in other situations this inversion is not required (*path-preserving-edge-exchange*).

Consider a *2-exchange* (Fig. 3) where two edges to be removed, $(h, h+1)$ and $(i, i+1)$, have been selected. In this situation there are only two ways to perform the exchange: in the first case (Fig. 3b), edges (h, i) and $(h+1, i+1)$ are inserted and the traveling direction for path $\langle i, \ldots, h+1\rangle$ is inverted; in the second case (Fig. 3c), edges (i, h) and $(i+1, h+1)$ are inserted inverting the traveling direction for path $\langle h, \ldots, i+1\rangle$.

In the case of a *3-exchange*, however, there are several possibilities to build a new solution when edges $(h, h+1)$, $(i, i+1)$, and $(j, j+1)$ are selected to be removed (Fig. 4). In Figure 4, a path preserving *3-exchange* (Fig. 4*b*) and a path inverting *3-exchange* (Fig. 4*c*) are shown.

It is then clear that any *2-exchange* procedure determines the inversion of one of the involved paths, while for $k = 3$ this inversion is caused only by particular choices of the inserted edges.

In the case of TSP problems, where arc costs $\eta_{ij} = \eta_{ji}$ $\forall(i, j)$, inverting a path does not modify its length. Therefore, the quality of the new solution depends only on the length of the inserted and deleted edges. On the other hand, for ATSP problems, where $\eta_{ij} \neq \eta_{ji}$ for at least one (i, j), inverting a path can modify the length of the path itself, and therefore the length of the new solution does not depend only on the inserted and deleted edges. This situation contrasts with the requirement that the improving criterion be verifiable in constant time. Therefore, the only suitable *edge-exchange* procedures for sequential ordering problems, which are a constrained version of ATSP problems, are *path-preserving-edge-exchange* heuristics. In the following, we concentrate on *path-preserving-k-exchange* (*pp-k-exchange* for short) with $k = 3$, that is, the smallest k that allows a path preserving edge exchange.

4.3 Handling Precedence Constraints

Starting from a feasible SOP sequence H, a *pp-3-exchange* tries to reduce the length of H by replacing edges $(h, h+1)$, $(i, i+1)$, and $(j, j+1)$ with edges $(h, i+1)$, $(i, j+1)$, and $(j, h+1)$ (Fig. 5*a*). The result of a *pp-3-exchange* is a new sequence H_1 (Fig. 5*b*) where, while walking from node 0 to node n, the order we visit path_left $= \langle h+1, \ldots, i\rangle$ and path_right $= \langle i+1, \ldots, j\rangle$ is swapped.

In this situation, the new sequence H_1 is feasible only if in the initial solution H there were no precedence constraints between a generic node $l \in$ path_left and a generic node $r \in$ path_right.

Given two generic paths *path_left* and *path_right*, to test

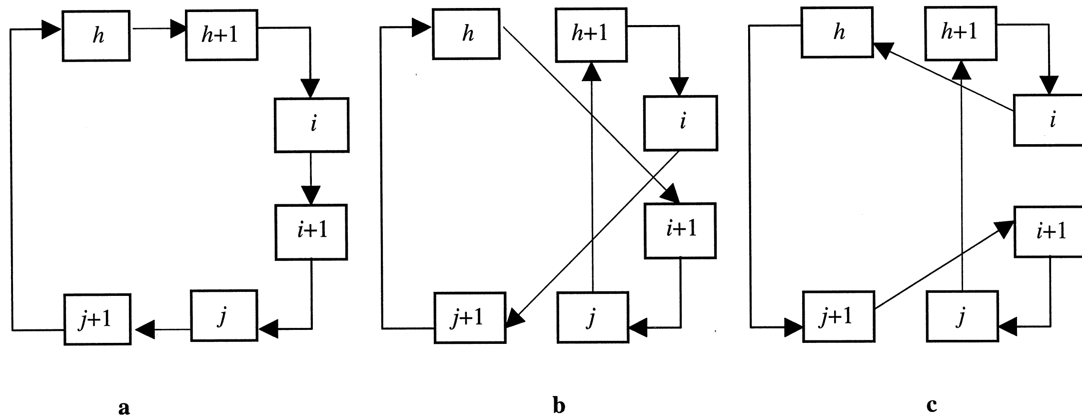

Figure 4. A 3-exchange without (*b*) and with (*c*) path inversion.

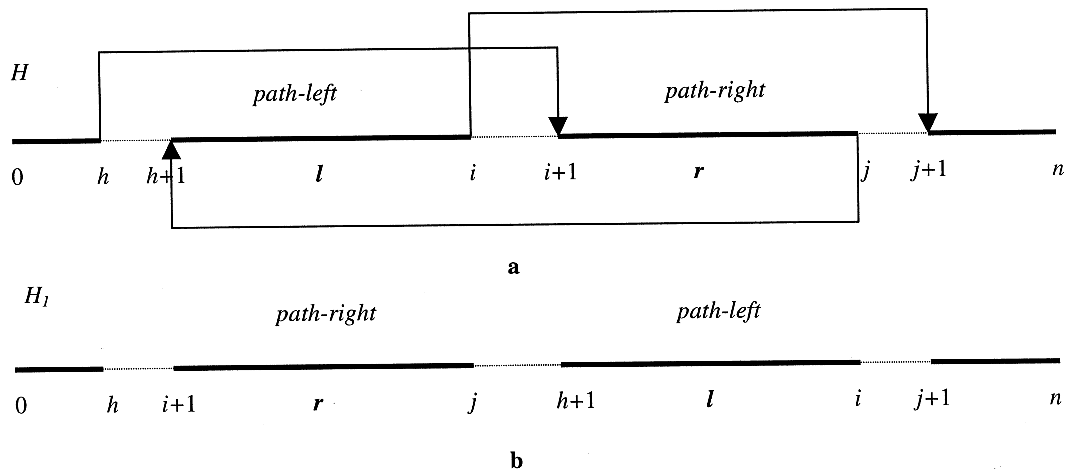

Figure 5. A path-preserving-3-exchange.

the feasibility of the *pp-3-exchange* requires computational effort of order $O(n^2)$ (precedence constraints must be checked between each pair of nodes in the two paths).

Savelsbergh (1990) studied how to limit the computational effort needed for checking solution feasibility in the case of precedence constraints for dial-a-ride problems. He introduced a particular exploration strategy called *lexicographic search strategy* that allows for generating and exploring only feasible exchanges. Savelsbergh presents a combination of the lexicographic search strategy with a labeling procedure where a set of global variables is updated so that precedence-constraint checking can be performed in constant time.

The lexicographic search strategy was introduced to solve dial-a-ride problems where only one precedence constraint for each node is allowed. Nevertheless, it is independent of the number of constraints. We have applied a version of Savelsbergh's lexicographic search strategy restricted to the case $k = 3$, *lpp-3-exchange*, to sequential ordering problems with multiple constraints for each node.

However, Savelsbergh's labeling procedure was designed to handle unique precedence constraints under particular search conditions and cannot be extended to sequential ordering problems. Before explaining our new labeling procedure for the SOP, we present the *lpp-3-exchange*.

4.4 Lexicographic Search Strategy in the Case of Precedence Constraints

The *lpp-3-exchange* procedure identifies two paths, *path_left* and *path_right*, which once swapped give rise to a new feasible solution. These two paths are initially composed of one single node and are incrementally expanded, adding one node at each step. This feature makes it possible to test feasibility easily because precedence conditions must be checked only for the new added node.

To explain how an *lpp-3-exchange* works, let us consider a feasible solution H in which nodes are ordered from 0 to n. Then we consider three indexes h, i, and j, that point to nodes in the sequence. As explained below, *lpp-3-exchange* is composed of two procedures that differ in the order nodes in the sequence H are explored. We start by explaining the *forward-lpp-3-exchange* procedure, *f-lpp-3-exchange* for short.

The *f-lpp-3-exchange* procedure starts by setting the value

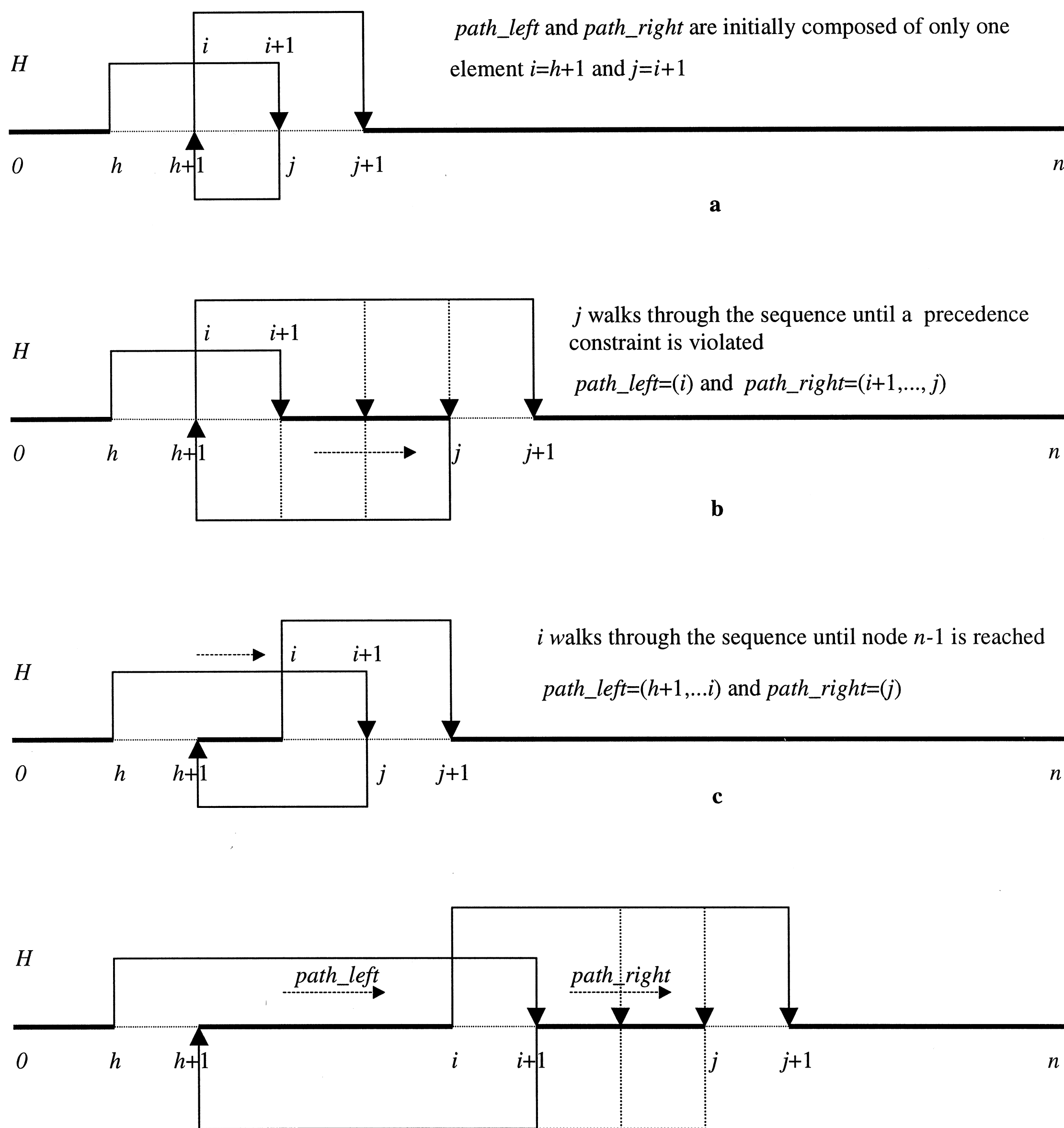

Figure 6. Lexicographic forward path-preserving-3-exchange.

of h to 0 (that is, h points to node 0 in the sequence H). Then it sets the value of i, which identifies the rightmost node of path_left, to $h + 1$ and performs a loop on the value of j, which identifies the rightmost node of path_right (Fig. 6, *a* and *b*). In other words, *path_right* $= \langle i + 1, \ldots, j\rangle$ is iteratively expanded by adding new edges $(j, j + 1)$. Once all available nodes have been added to *path_right* (that is, until a precedence constraint is violated or when $j + 1$ points to node n; see Fig. 6*b*), *path_left* is expanded by adding the new edge $(i, i + 1)$ (Fig. 6*c*), and then *path_right* is searched again. *Path_left* $= \langle h + 1, \ldots, i\rangle$ is expanded until $i + 1$ points to node $n - 1$. Then h is set to $h + 1$ and the process is repeated. The *f-lpp-3-exchange* procedure stops when h points to node $n - 2$.

As we said, *f-lpp-3-exchange* considers only forward exchanges, that is, exchanges obtained considering indexes i and j such that $j > i > h$. The *backward-lpp-3-exchange* procedure (*b-lpp-3-exchange* for short) considers *backward exchanges*, that is, exchanges obtained considering indexes j and i such that $j < i < h$ (with $2 \leq h < n$). In *b-lpp-3-exchange* (Fig. 7), *path_left* is identified by $\langle j + 1, \ldots, i\rangle$ and *path_right* by $\langle i + 1, \ldots, h\rangle$. After fixing h, i is set to $h - 1$ and j to $i - 1$ (Fig. 7*a*). Then *path_left* is expanded backward (Fig. 7*b*), moving j till the beginning of the sequence, that is, iteratively setting j to the values $i - 2, i - 3, \ldots, 0$ (i.e., each backward expansion adds a new node to the left of the path: $\langle j + 1, \ldots, i\rangle$ is expanded to $\langle j, j + 1, \ldots, i\rangle$). Then, *path_right* is iteratively expanded in a backward direction with the new edge $(i, i + 1)$, and the loop on *path_left* is repeated.

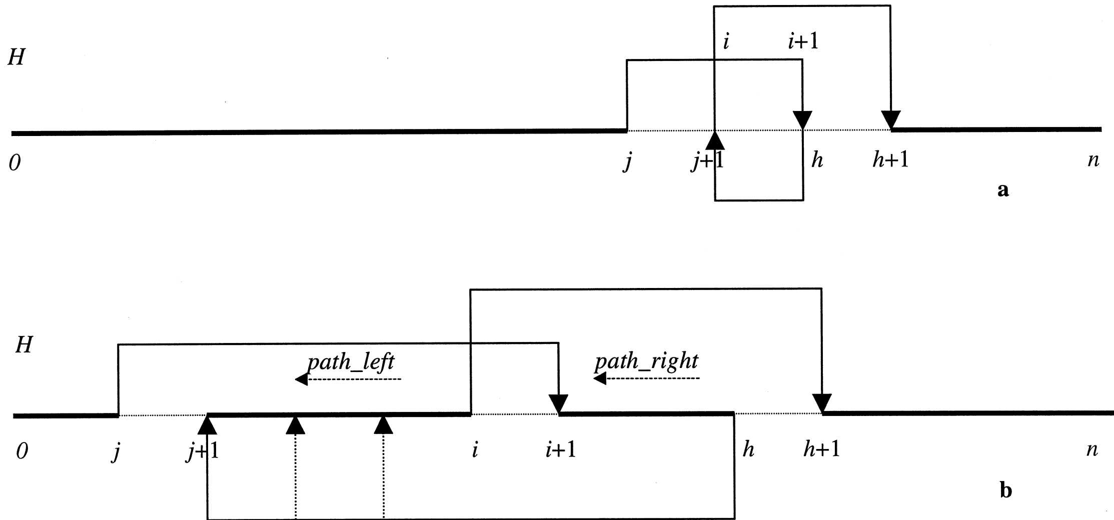

Figure 7. Lexicographic backward path-preserving-3-exchange.

The complete SOP-3-exchange procedure performs a forward and a backward lexicographic search for each value h, visiting in this way all the possible nodes in the sequence (just like any other *3-exchange* procedure).

The important point is that the method for defining *path_left* and *path_right* permits an easy solution of the feasibility-checking problem: the search is restricted to feasible exchanges only, since it can be stopped as soon as an infeasible exchange is found. Consider for example an *f-lpp-3-exchange*: once *path_left* $= \langle h + 1, \ldots, i \rangle$ has been fixed, we set *path_right* to $j = i + 1$. In this situation it is possible to check exchange feasibility by testing whether there is a precedence relation between node j and nodes in *path_left*. Before expanding *path_right* with the new edge $(j, j + 1)$, we check whether the resulting paths are still feasible by testing again the precedence relations between the new node $j + 1$ and nodes in *path_left*. If the test is not feasible, we stop the search. In fact, any further expansion of $j + 1$ in $\langle j + 2, j + 3, \ldots, n \rangle$ will always generate an infeasible exchange because it still violates at least the precedence constraint between $j + 1$ and *path_left*.

Note that expanding *path_left* with edge $(i, i + 1)$ does not induce any precedence constraint violations because the order of nodes inside *path_left* is not modified and the search for a profitable *f-lpp-3-exchange* always starts by setting *path_right* equal to element $j = i + 1$.

Without considering any additional labeling procedure, the feasibility test in this situation has a computational cost of $O(n)$: each time a new j is selected we test if there is a precedence relation between j and the nodes in *path_left*. In the case of the SOP, this test should check whether $c_{jl} \neq -1$ $\forall l \in$ *path_left* (recall that for sequential ordering problems, $c_{jl} = -1$ if l has to precede j, and in the final solution H_1 the order in which we visit *path_left* and *path_right* is swapped and therefore l will follow j; see Fig. 5*b*).

Similar considerations should be made in the case of *b-lpp-3-exchange*, where the feasibility test checks if $c_{r,j+1} \neq -1$ $\forall r \in$ *path_right*.

The previous *complete lexicographic search procedure* requires a check of all predecessors/successors of node j. This procedure increases the computational effort to check *3-optimality* from $O(n^3)$ to $O(n^4)$.

In order to keep the cost at $O(n^3)$, we introduce the *SOP labeling procedure* to handle multiple constraints.

4.5 The SOP Labeling Procedure

The *SOP labeling procedure* is used to mark nodes in the sequence with a label that allows for feasibility checking for each selected j in constant time. The basic idea is to associate with each node a label that indicates, given *path_left* and *path_right*, whether or not it is feasible to expand *path_right* with the following node $j + 1$.

We have implemented and tested different SOP labeling procedures that set and update nodes in different phases of the search. In the following, we will present a combination of the best-performing SOP labeling procedure with the *lexicographic search strategy*, with different selection criteria for node h and with different search-stopping criteria.

Our SOP labeling procedure is based on a set of global variables that are updated during the *lexicographic search* procedure. As in the previous subsection, we will distinguish between forward and backward search.

First we introduce a global variable *count_h* that is set to 0 at the beginning of the search, and which is increased by 1 each time a new node h is selected. Second, we associate a global variable f-mark(v) to each node $v \in H$ in the case of *f-lpp-3-exchange*, and a global variable b-mark(v) in the case of *b-lpp-3-exchange*. These global variables are initially set to 0 for each v.

An *f-lpp-3-exchange* starts by fixing h, $i = h + 1$, and *path_left* $= (i)$. At this point, for all nodes $s \in$ successor[i] we set f_mark(s) = count_h. We repeat this operation each time

path_left is expanded with a new node *i*. Therefore, the labeling procedure marks with the value *count_h* all the nodes in the sequence that must follow one of the nodes belonging to *path_left*. When *path_right* is expanded moving *j* in $\langle i+2, \ldots, n \rangle$, if $f_mark(j) = count_h$ we stop the search because the label indicates that *j* must follow a node in *path_left*. At this point, if no other search-termination condition is met, the procedure restarts expanding again *path_left*. In this situation all the previous defined labels remain valid and the search continues by labeling all the successors of the new node *i*.

On the other hand, when we move *h* forward into the sequence we invalidate all previously set labels by setting count_h = count_h + 1.

The same type of reasoning holds for *b-lpp-3-exchange*. Each time node *i* is selected, we identify a new path_right = $\langle i+1, \ldots, h \rangle$, and for all nodes $s \in$ predecessor$[i+1]$ we set b_mark(s) = count_h.

When expanding *path_left* by iteratively adding a new edge $(j, j+1)$, the expansion is not accepted if b_mark(j) = count_h.

4.6 Heuristics for the Selection of Node *h* and Search Stopping Criteria

This search procedure for sequential ordering problems is a general description of how the *lexicographic search* works in combination with the SOP labeling procedure. Although the SOP labeling procedure reduces the complexity of the lexicographic search to $O(n^3)$, this is still too expensive from a practical point of view; in fact, the exploration of all the feasible exchanges is still required. There are different ways to reduce this effort: for example, heuristic criteria can be introduced to reduce the number of visited nodes, or the search can be stopped and the exchange executed as soon as some improving condition is met.

Heuristic Selection of Node *h*. In order to reduce the number of explored nodes, Savelsbergh (1990) and Van der Bruggen et al. (1993) proposed to use a particular type of *k-exchange* called *OR-exchange* (Or 1976) that limits the choice of *i* among the three closest nodes of *h*. In practice, *i* is selected among $\{h+1, h+2, h+3\}$ in the case of a forward exchange, and among $\{h-1, h-2, h-3\}$ in the case of a backward exchange.

Alternatives decrease the number of visited nodes, introducing two heuristics that influence how node *h* is chosen: one is based on the *don't look bit* data structure introduced by Bentley (1992), while the other is based on a new data structure called *don't push stack* introduced by the authors.

The *don't look bit* is a data structure in which a bit is associated with each node of the sequence. At the beginning of the search all bits are turned off. The bit associated with node *h* is turned on when a search for an improving move starts from node *h*. If a profitable exchange is executed, the bits of the six nodes involved in the exchange (that is, $j+1$, $i+1$, $h+1$, j, i, h) are turned off. The use of *don't look bits* favors the exploration of nodes that have been involved in a profitable exchange. The search procedure visits all the nodes in the sequence, moving from the first node 0 to the last node *n*, but only nodes with the *don't look bit* turned off are taken into consideration as candidates for node *h*. The search procedure is repeatedly applied until all nodes have their *don't look bit* turned on.

The *don't push stack* is a data structure based on a *stack*, which contains the set of nodes *h* to be selected, associated with a particular push operation. At the beginning of the search the *stack* is initialized with all the nodes (that is, it contains $n+1$ elements). During the search, node *h* is popped off the stack and feasible 3-exchange moves starting from *h* are investigated. If a profitable exchange is executed, the six nodes involved in this exchange (that is, $j+1$, $i+1$, $h+1$, j, i, h) are pushed onto the stack (if they do not already belong to it). Using this heuristic, once a profitable exchange is executed starting from node *h*, the top node in the *don't push stack* remains node *h*. In addition, the maximum size of the stack is limited to $n+1$ elements. The use of the *don't push stack* gives the following benefits. First, the search is focused on the neighborhood of the most recent exchange: this has been experimentally shown to result in better performance than that obtained using the *don't look bit* (see Section 5.1). Second, the selection of node *h* is not constrained to be a sequential walk through the sequence H. This is an important feature given the fact that the SOP labeling procedure is designed to work with independent and random choices of *h*, where independent means that the choice of the new *h* is not constrained by the choice of the old *h*. In fact, it does not require, as is the case of Savelsbergh's labeling procedure (Savelsbergh 1990), retention of valid labeling information while walking through the sequence from one *h* to the next. In our case a new labeling is started as soon as a new *h* is chosen; this allows for selecting *h* in any sequence position without introducing additional computational costs.

Stopping Criteria. The number of visited nodes can be decreased by stopping the search once an improving exchange is found. In our experiments we have tested three different stopping conditions: (ExchangeFirstCriterion = *h*, *i*, *j*) stops the search as soon as the first feasible exchange is found in the *h*, *i* or *j* loops, respectively. We have also tested the standard exploration strategy where the most profitable exchange is selected among all the possible exchanges, but this method is not presented here because the results obtained are much worse given the same amount of computation time.

4.7 The SOP-3-Exchange Procedure: An Example

In this section we discuss briefly the local search procedure using a simple example. Fig. 8 presents the pseudo-code of the SOP-3-exchange procedure with all the possible options. The example we consider is the following: let a sequence $\langle 0, a, b, c, d, e, f, g, n \rangle$ represent a feasible solution of a SOP in which node *e* is constrained to follow *a* in any feasible solution. The forward SOP-3-exchange procedure works as follows. Initially, *h* is set to point to node 0 (i.e., $h = 0$), variable *count_h* is set to zero, *direction* is set to forward, *i* is set to *a*, and *j* to *b*. In this state of the computation *path_left* (from node $h+1$ to node *i*) and *path_right* (from node $i+1$ to node *j*) consist of the sequences $\langle a \rangle$ and $\langle b \rangle$, respectively. In the following, the notation $[\langle a \rangle \langle b \rangle]$ will be used to indicate the pair *path_left* and *path_right*.

```
/* input:
       a feasible solution given as a sequence (0.......n)
       a sequential ordering problem G
       a SelectionCriterion for h in (sequential,dont_look_bit,dont_push_stack)
       a WalkingCriterion for i in (3-exchange,OR-exchange)
       an ExchangeFirstCriterion in (h,i,j)
   output:
       a new feasible solution that is 3-optimal */
REPEAT
h←0 /* h is set to the first node in the sequence */
while there is an available h /* h loop */
/* Selects node h according to SelectionCriterion.
   In case SelectionCriterion=sequential, h is the next node in the sequence.
   In case SelectionCriterion=dont_look_bit, h is the next node in the sequence with dont_look_bit[h]=off.
   In case SelectionCriterion=dont_push_stack, h is popped from the stack */
h ← SelectAccordingCriterion(SelectionCriterion,G);
direction ← forward; /* the search starts in forward direction */
gain ← 0; i ← h+1; j ← i+1; SearchTerminated ← false
   while SearchTerminated   /* i loop */
   /* When i has reached the end of the sequence during a forward search we start a new search in backward
        direction starting from the same h. In case WalkingCriterion=OR-exchange the direction is inverted
        after three selections of i */
        feasible ← true
        if direction=forward and EndOfSequence(i,WalkingCriterion,G)
            direction ← backward; i ← h-1; j ← i-1
        end-if
        /* in case direction=forward we update labeling information for successor[i];
           in case direction=backward we update labeling information for predecessor[i+1] */
        UpdateGlobalVariables(h,i,direction,G)
        while feasible /*j loop */
            /* Using labeling information we test if the 3-exchange involving h,i,j is feasible */
            feasible←FeasibleExchange(h,i,j,direction,G)
            /* Checks if the new 3-exchange is better then the previous one; if the case, saves it */
            gain ← ComputeBestExchange(h,i,j,direction,G,feasible,gain)
            if gain>0 and (ExchangeFirstCriterion=j) goto PERFORM-EXCHANGE
            /* j is moved through the sequence according to direction
               in case direction=forward j←j+1, in case direction=backward j←j-1 */
            j ← jWalkThroughTheSequence(h,i,j,direction,G)
            SearchTerminated ← f(j,direction,WalkingCriterion)
        end-while
        if gain>0 and (ExchangeFirstCriterion =i) goto PERFORM-EXCHANGE
        /* i is moved through the sequence according to direction;
           in case direction=forward i←i+1, in case direction=backward i←i-1 */
        i ← iWalkThroughTheSequence(h,i,direction,G)
        SearchTerminated ← f(i,direction,WalkingCriterion)
   end-while
PERFORM-EXCHANGE
   if gain>0
        /* the best exchange found until now is executed and the search starts again */
        PerformExchange(h,i,j,direction,G)
        goto REPEAT
   end-if
end-while
```

Figure 8. The SOP-3-exchange procedure.

Inside the i loop, the successor of node a, node e, is labeled by setting $f_mark(e) = count_h = 0$. In the j loop, *path_right* is expanded by adding nodes of the sequence until either the end of the sequence is reached or a precedence constraint is violated. The first expansions are $[\langle a\rangle\ \langle b, c\rangle]$ and $[\langle a\rangle\ \langle b, c, d\rangle]$. At this point, *path_right* should not be extended to $\langle b, c, d, e\rangle$ because e is labeled with a value equal to *count_h*. In fact, the new sequence generated by using $[\langle a\rangle\ \langle b, c, d, e\rangle]$ would be $\langle 0, b, c, d, e, a, f, g, n\rangle$, where node a follows node e, in contrast with the precedence constraint. Therefore, the j loop is terminated and the i loop is resumed. Node i is moved through the sequence by setting i equal to node b, and the two paths are set to $[\langle a, b\rangle\ \langle c\rangle]$. Node b does not have any successor node to label; therefore, the j loop is executed again. Paths are expanded to $[\langle a, b\rangle\ \langle c, d\rangle]$ but, as before, they should not be extended to $[\langle a, b\rangle\ \langle c, d, e\rangle]$ due to the precedence constraint. The procedure continues generating the paths $[\langle a, b, c\rangle\ \langle d\rangle]$, while the following paths $[\langle a, b, c\rangle\ \langle d, e\rangle]$ and $[\langle a, b, c,$

Table I. Ranking of Median Rank on 22 SOP Test Problems for Different Combinations of Selection and Stopping Criteria*

SelectionCriterion	ExchangeFirstCriterion	WalkingCriterion	Median
don't push stack	*i*	3-exchange	4
don't push stack	*j*	3-exchange	5
don't push stack	*h*	3-exchange	6
don't push stack	*j*	OR_exchange	6
don't look bit	*h*	3-exchange	7
don't push stack	*h*	OR_exchange	8
don't look bit	*j*	3-exchange	8
don't look bit	*i*	3-exchange	8
sequential	*h*	OR_exchange	9
don't push stack	*i*	OR_exchange	10
don't look bit	*h*	OR_exchange	10
sequential	*i*	OR_exchange	11
don't look bit	*j*	OR_exchange	12
sequential	*j*	OR_exchange	12
sequential	*h*	3-exchange	13
sequential	*j*	3-exchange	13
sequential	*i*	3-exchange	13
don't look bit	*i*	OR_exchange	14

*Results are obtained running five experiments for each problem (CPU time was set to 100 seconds for the ft53.x, ft70.x, and ESCxx problems, to 300 seconds for the kro124p.x problems, and to 600 seconds for the other problems).

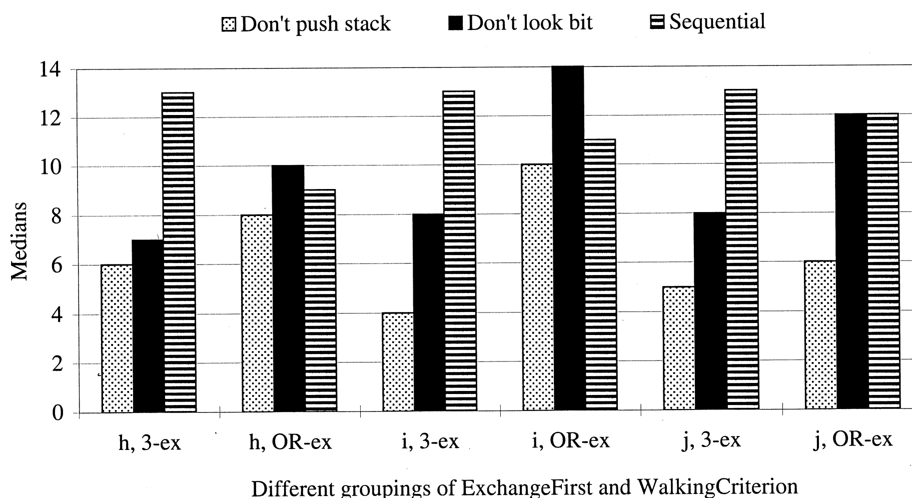

Figure 9. Comparisons of selection criteria on median ranks. Each comparison involves the same value of ExchangeFirstCriterion and WalkingCriterion. Results are obtained running five experiments for each problem (CPU time was set to 100 seconds for the ft53.x, ft70.x, and ESCxx problems, to 300 seconds for the kro124p.x problems, and to 600 seconds for the other problems).

d⟩ ⟨*e*⟩] are not feasible because of the constraint between *a* and *e*. The next feasible steps are [⟨*a*, *b*, *c*, *d*, *e*⟩ ⟨*f*⟩], [⟨*a*, *b*, *c*, *d*, *e*⟩ ⟨*f*, *g*⟩], and [⟨*a*, *b*, *c*, *d*, *e*, *f*⟩ ⟨*g*⟩].

5. Computational Results

Our experiments were aimed at (i) finding the best parameters for the SOP-3-exchange procedure, (ii) comparing ACS-SOP and HAS-SOP with a set of competing methods over a significant set of test problems, and (iii) evaluating the relative contribution to overall performance of the SOP-3-exchange local search with respect to the constructive methods. The results obtained are presented and discussed in this section. Experiments were run on a SUN Ultra1 SPARC Station (167 MHz). The code was written in C++. Before presenting and discussing the computational results, we briefly describe the experimental setting.

Table II. Small Problems (≤100 Nodes)*

	RND	MPO/AI	ACS-SOP	RND+LS	MPO/AI+LS	HAS-SOP
ESC78	49.81%	0.86%	2.15%	**0.00%**	**0.00%**	**0.00%**
ft53.1	167.93%	0.49%	13.11%	0.10%	**0.00%**	**0.00%**
ft53.2	154.94%	0.72%	12.27%	0.36%	**0.00%**	**0.00%**
ft53.3	100.51%	0.59%	18.51%	**0.00%**	**0.00%**	**0.00%**
ft53.4	40.99%	0.00%	5.03%	**0.00%**	**0.00%**	**0.00%**
ft70.1	64.94%	0.76%	11.65%	0.37%	0.10%	**0.00%**
ft70.2	59.18%	0.03%	11.63%	0.85%	**0.00%**	0.02%
ft70.3	52.22%	0.03%	13.22%	0.49%	**0.00%**	**0.00%**
ft70.4	24.62%	0.09%	3.92%	0.08%	**0.02%**	0.05%
kro124p.1	301.69%	4.17%	28.81%	2.65%	0.68%	**0.00%**
kro124p.2	278.99%	3.00%	27.90%	2.90%	**0.19%**	0.26%
kro124p.3	215.49%	3.20%	24.49%	3.75%	1.40%	**0.31%**
kro124p.4	94.07%	0.00%	8.66%	1.23%	**0.00%**	**0.00%**
Average	123.49%	1.07%	13.95%	0.98%	0.18%	**0.05%**

*Shown are the average percentages of deviation from the best-known solution. Results are obtained over five runs of 120 seconds. Best results are in boldface.

Table III. Big Problems (>100 Nodes)*

	RND	MPO/AI	ACS-SOP	RND+LS	MPO/AI+LS	HAS-SOP
prob.100	1440.17%	134.66%	40.62%	50.07%	47.58%	**17.46%**
rbg109a	64.57%	0.33%	1.93%	0.08%	0.06%	**0.00%**
rbg150a	37.85%	0.19%	2.54%	0.08%	0.13%	**0.00%**
rbg174a	40.86%	0.01%	2.16%	0.15%	**0.00%**	0.08%
rbg253a	45.85%	0.03%	2.68%	0.21%	**0.00%**	**0.00%**
rbg323a	80.14%	1.08%	9.60%	1.27%	**0.08%**	0.21%
rbg341a	125.46%	3.02%	12.64%	4.41%	**0.96%**	1.54%
rbg358a	151.92%	7.83%	20.20%	4.98%	2.51%	**1.37%**
rbg378a	131.58%	5.95%	22.02%	4.17%	1.40%	**0.88%**
Average	235.38%	17.01%	12.71%	7.27%	5.86%	**2.39%**

*Shown are the average percentages of deviation from the best-known solution. Results are obtained over five runs of 600 seconds. Best results are in boldface.

5.1 Experimental Settings: Test Problems

We tested our algorithms on the set of problems available in the TSPLIB (http://www.iwr.uni-heidelberg.de/iwr/comopt/software/TSPLIB95/).

Sequential ordering problems in TSPLIB can be classified as follows: a set of problems (rbgxxxa) are real-life problems derived from a stacker crane application (Ascheuer 1995). These problems were originally defined as ATSPs with time windows: to obtain SOP instances, time window precedences are relaxed to generate SOP precedences. Prob.100 (Ascheuer 1995) is a randomly generated problem, and problems (ftxx.x and kroxxxp.x) have been generated (Ascheuer 1995) starting from ATSP instances in TSPLIB by adding a number $\leqslant k$ of random precedence constraints, where $k = (n/4, n/2, 2, 2n)$ corresponds to the problem extension (.1, .2, .3, .4). ESC78 is taken from Escudero (1988).

5.2 Experimental Settings: Competing Methods

The algorithms with which we compared HAS-SOP are the following:

- MPO/AI: This was previously the best known algorithm for the SOP (Chen and Smith 1996). MPO/AI is a genetic algorithm explicitly designed to solve sequencing problems. Each individual is a feasible sequence represented by an $n \times n$ Boolean matrix. An element (i, j) of the matrix is set to 1 if node j follows (not necessarily immediately) node i in the sequence, and is set to 0 otherwise. New individuals are generated by a specialized crossover operation. First, the two matrices are intersected; the intersection generates a new matrix where, in general, only partial subsequences (with fewer than n elements) are present. Next, the longest subsequence MPO in the new matrix is selected and is completed by using an AI pro-

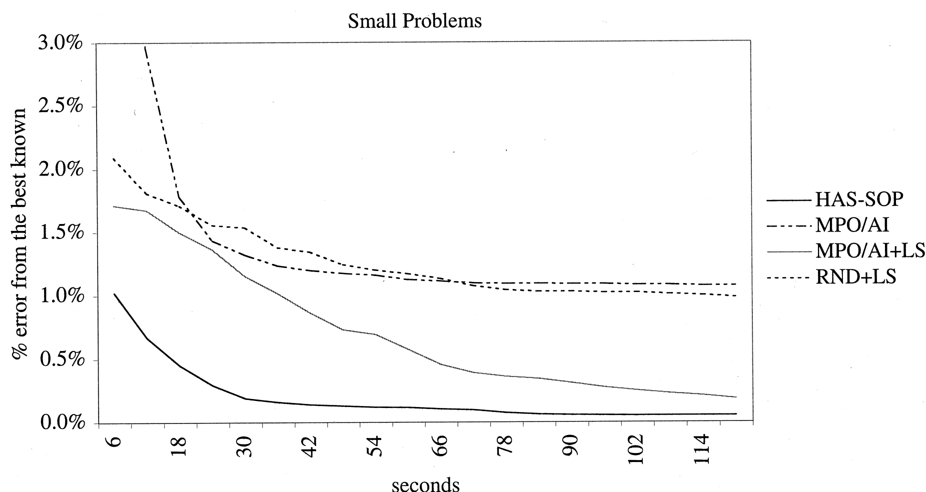

Figure 10. Comparison across algorithms over small problems. Results are obtained over five runs of 120 seconds.

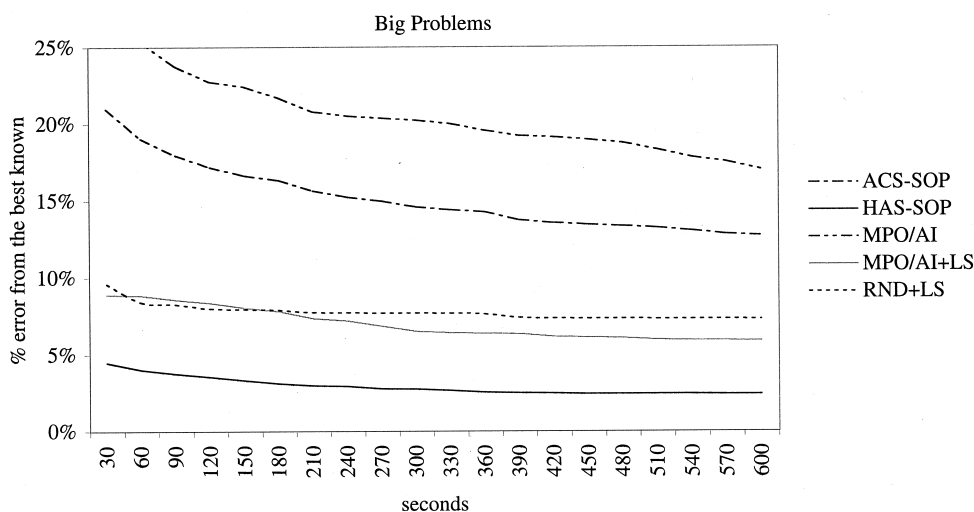

Figure 11. Comparison across algorithms over big problems. Results are obtained over five runs of 600 seconds.

cedure. AI starts from a sub-tour, picks an arbitrary node not already included, and inserts it in the feasible position with minimum cost. This simple local search procedure is applied until no further elements are available. The code has been implemented by Chen in C++ and is available on the Web at: http://www.cs.cmu.edu/afs/cs.cmu.edu/user/chens/WWW/MPOAI_SOP.tar.gz.

Experiments were run setting the population to many different dimensions. Using 500 individuals, the same population dimension as proposed by Chen and Smith (1996) resulted in the best performance, and this value was used in all the experiments presented in the following.

- MPO/AI+LS: This is MPO/AI to which we added the SOP-3-exchange local search. The hybridization is similar to what was done with ACS-SOP: each time a new individual is created by the MPO/AI crossover operation, it is optimized by the SOP-3-exchange local search (with the main structure of the genetic algorithm remaining unchanged).
- RND: This algorithm generates random feasible solutions. The constructive procedure is the same as in ACS-SOP except that pheromone trail and distance information are not used.
- RND+LS: This is RND plus local search. As with the other hybrid algorithms considered, each time a new individual is created, it is optimized by the SOP-3-exchange local search.

5.3 Computational Results: Selection Criteria for Node *i* and Search Stopping Criteria

In this section we test different selection criteria for node *i* and different search stopping criteria. We ran five experiments for each problem, setting the computational time to

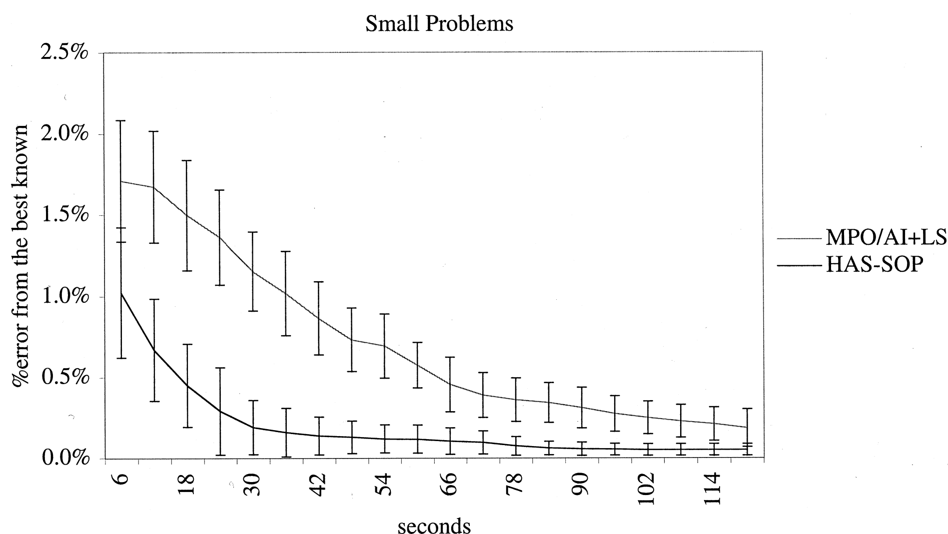

Figure 12. Comparison between MPO/AI+LS and HAS-SOP over small problems. Results are obtained over five runs of 120 seconds. Error bars (one standard deviation) are shown.

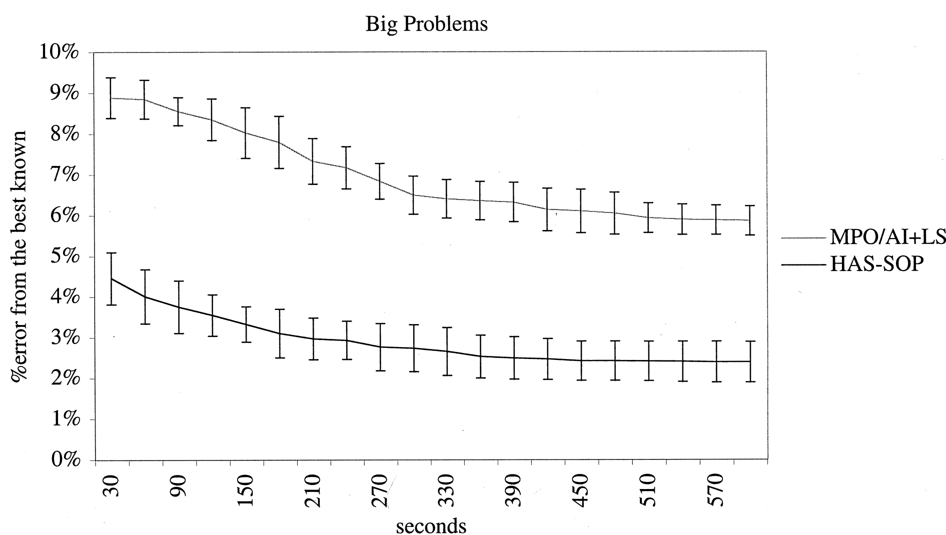

Figure 13. Comparison between MPO/AI+LS and HAS-SOP over big problems. Results are obtained over five runs of 600 seconds. Error bars (one standard deviation) are shown.

100 seconds for the ft53.x, ft70.x, and ESCxx problems, to 300 seconds for the kro124p.x problems, and to 600 seconds for the other problems.

The stopping criteria tested are ExchangeFirstCriterion = *j*, *i*, *h*. The selection criteria tested are *sequential*, *don't look bit*, and *don't push stack*, coupled with either the *3-exchange* or the *OR-exchange* walking criterion.

For each test problem (the problems are reported in Tables II and III), we ranked results computed by the different combinations of selection and stopping criteria according to the average results obtained. In Table I the methods are ranked by the median of each method over the set of test problems.

Results indicate that the *don't push stack* is the best selection criterion, followed by the *don't look bit* and finally by the *sequential* selection criterion. Fig. 9 compares the three selection criteria for the same values of ExchangeFirstCriterion and WalkingCriterion. Again, it is clear that *don't push stack* performs better than the other two criteria.

These results were obtained using HAS-SOP. That is, the SOP-3-exchange local search was applied to feasible solutions generated by ACS-SOP. We ran the same experiment using the other solution generation methods (i.e., MPO/AI and RND), and we found that also in these cases the best performance was obtained by setting *SelectionCriterion* = *don't push stack*, *ExchangeFirstCriterion* = *i*, and *WalkingCriterion* = *3-exchange*. These parameters are therefore used in

Table IV. Percentage of Improvement Due to Local Search*

	Δ% RND	Δ% MPO/AI	Δ% ACS-SOP
ESC78	49.81%	0.86%	2.15%
ft53.1	167.83%	0.49%	13.11%
ft53.2	154.58%	0.72%	12.27%
ft53.3	100.51%	0.59%	18.51%
ft53.4	40.99%	0.00%	5.03%
ft70.1	64.57%	0.66%	11.65%
ft70.2	58.33%	0.03%	11.61%
ft70.3	51.73%	0.03%	13.22%
ft70.4	24.55%	0.06%	3.88%
kro124p.1	299.04%	3.50%	28.81%
kro124p.2	276.09%	2.81%	27.65%
kro124p.3	211.74%	1.81%	24.18%
kro124p.4	92.84%	0.00%	8.66%
prob.100	1390.10%	87.08%	23.16%
rbg109a	64.49%	0.27%	1.93%
rbg150a	37.77%	0.07%	2.54%
rbg174a	40.71%	0.01%	2.09%
rbg253a	45.64%	0.03%	2.68%
rbg323a	78.87%	1.00%	9.39%
rbg341a	121.05%	2.06%	11.10%
rbg358a	146.94%	5.31%	18.83%
rbg378a	127.41%	4.55%	21.14%
Small Average	122.51%	0.89%	13.90%
Big Average	228.11%	11.15%	10.32%
All Average	165.71%	5.09%	12.44%

*The last three rows report, respectively, the average over the small, big, and all problems. Results are obtained over five runs of 120 seconds (small problems, that is, ft53.x, ft70.x, ESCxx, and kro124p.x problems) and 600 seconds for the others.

all the experiments involving local search presented in the following sections. (It should otherwise be noted that, for MPO/AI and RND, although the best parameter settings remained the same, the ordering of the other possible combinations of parameter values was different.)

5.4 Computational Results and Comparisons with Other Methods

In this section, we compare the ACS-SOP, RND, and MPO/AI algorithms and their hybrid versions (using the local search with the best parameters experimentally found as explained in Section 5.3). To run the comparisons we divided the set of test problems into two sets: smaller easier problems and larger more difficult problems. The separation point was set to be 100 nodes: small problems have 100 or fewer nodes, big problems have more than 100 nodes (with the exception of prob.100 that, because of its difficulty, although having 100 nodes was assigned to the set of big problems). Experiments were run giving a fixed amount of CPU time to the algorithms. The CPU time was fixed to be the same for all algorithms running on the same set of problems: 120 seconds for each small problem, 600 seconds for each big problem.

Results at the end of the experiment are reported in Tables II and III for small and big problems, respectively, while the runtime behavior of the various algorithms is shown in Figs. 10 and 11.

If we observe the average performance of the algorithms on the set of small problems (Table II) we can make the following observations: (i) RND is, as it was expected, the worst performing algorithm; (ii) ACS-SOP performs better than RND, which means that the additional use of pheromone trails and local heuristic information (i.e., distance between nodes) is useful, (iii) MPO/AI is the best of the algorithms not using our local search (in fact, MPO/AI uses a simple form of local search, which can explain its better performance), and (iv) when the SOP-3-exchange local search is added, all the algorithms, as expected, increase their performance, and HAS-SOP with an average 0.05% deviation from the best-known solutions is the best performing algorithm.

Similar observations can be done for the set of big problems (Table III). The only difference is that, on the average, ACS-SOP performs better than MPO/AI. This is mainly due to problem prob.100, a difficult problem that ACS-SOP solves much better than the competing methods. Also, in the case of big problems, HAS-SOP is the best performing algorithm, with an average error of 2.39% from the best-known solutions.

Fig. 10 shows the runtime behavior of HAS-SOP, MPO/AI, MPO/AI+LS, and RND+LS on small problems (RND and ACS-SOP are not plotted because they are out of scale). It is clear that, besides reaching slightly better results than MPO/AI+LS, HAS-SOP has also a better convergence speed: it reaches after 12 seconds the same performance level reached by MPO/AI+LS after approximately 60 seconds. Similar considerations can be done for big problems (Fig. 11), where all algorithms are plotted (with the exception of RND, which is out of scale). Note the small difference in behavior between RND+LS and MPO/AI+LS.

A more detailed version of Figs. 10 and 11 showing the performance of the two best algorithms, HAS-SOP and MPO/AI+LS, with error bars is given in Figs. 12 and 13.

Table IV shows the percentage improvement due to local search (this is computed as the difference between the performance of the basic algorithm and the performance of the corresponding hybrid algorithm reported in Tables II and III). Data show that MPO/AI profits from local search less than ACS-SOP and RND. This is probably due to the fact that MPO/AI generates solutions that are already close to local optima, and therefore the SOP-3-exchange procedure quickly gets stuck. On the contrary, RND is the algorithm that best exploits local search. Unfortunately, this is due to the very poor quality of the solution given as a starting point to the local search: notwithstanding the great improvement caused by the local search, the final result is not competitive with that produced by HAS-SOP. In some sense it seems that solutions generated by ACS-SOP are good enough to let local search work fruitfully, yet they are not so "good" as to

Table V. Results Obtained by HAS-SOP and MPO/AI+LS on a Set of 22 Test Problems*

	MPO/AI+LS				HAS-SOP			
PROB	Best Result	Avg. Result	Std. Dev.	Avg. Time (sec)	Best Result	Avg. Result	Std. Dev.	Avg. Time (sec)
ESC78	18,230	18,230.0	0.0	12.4	18,230	18,230.0	0.0	**3.5**
ft53.1	7,531	7,531.0	0.0	16.6	7,531	7,531.0	0.0	**16.3**
ft53.2	8,026	8,026.0	0.0	**14.0**	8,026	8,026.0	0.0	17.0
ft53.3	10,262	10,262.0	0.0	7.8	10,262	10,262.0	0.0	**3.8**
ft53.4	14,425	14,425.0	0.0	11.4	14,425	14,425.0	0.0	**0.5**
ft70.1	39,313	39,352.4	33.2	81.4	39,313	**39,313.0**	0.0	20.9
ft70.2	40,419	**40,419.6**	1.2	81.6	40,419	40,428.6	12.0	41.0
ft70.3	42,535	42,535.0	0.0	**27.0**	42,535	42,535.0	0.0	36.8
ft70.4	53,530	**53,542.8**	15.7	42.2	53,530	53,554.6	20.5	58.3
kro124p.1	39,502	39,686.2	214.0	97.4	**39,420**	**39,420.0**	0.0	60.8
kro124p.2	41,336	**41,415.4**	114.2	95.2	41,336	41,442.8	127.8	53.2
kro124p.3	49,835	50,189.6	298.0	97.4	**49,499**	**49,653.2**	66.3	24.2
kro124p.4	76,103	76,103.0	0.0	47.8	76,103	76,103.0	0.0	**34.2**
prob.100	1,722	1,756.2	30.6	333.0	**1,344**	**1,397.8**	38.5	404.4
rbg109a	1,038	1,038.6	0.5	75.8	1,038	**1,038.0**	0.0	27.5
rbg150a	1,751	1,752.2	0.7	17.2	**1,750**	**1,750.0**	0.0	128.1
rbg174a	2,033	**2,033.0**	0.0	82.8	2,033	2,034.6	1.1	189.4
rbg253a	2,950	2,950.0	0.0	**68.6**	2,950	2,950.0	0.0	145.0
rbg323a	**3,143**	**3,143.6**	0.4	458.8	3,146	3,147.6	2.2	271.1
rbg341a	**2,588**	**2,598.8**	2.0	553.6	2,609	2,613.6	18.1	421.3
rbg358a	2,602	2,609.0	6.2	482.8	**2,574**	**2,579.8**	4.8	454.1
rbg378a	2,841	2,856.4	8.1	516.0	**2,831**	**2,841.8**	6.8	500.6
Total wins	2	6	—	3	6	8	—	5

*See text for explanation of boldface and "Total wins" row. Results are obtained over five runs of 120 seconds (small problems, that is, ft53.x, ft70.x, ESCxx, and kro124p.x problems) and 600 seconds for the others.

impede the work of the local search, as is the case for MPO/AI.

In Table V we compare HAS-SOP with MPO/AI+LS. As in the previous experiment, runs lasted different amounts of CPU time: 120 seconds for small problems and 600 seconds for big problems. Each experiment was run five times. For both algorithms in Table V we report:

- Best Result: the best result obtained over five experiments.
- Average Result: average of the best results obtained in each experiment.
- Std. Dev.: standard deviation of the best results obtained in each experiment.
- Average Time: average time (in seconds) needed to reach the best result in each experiment.

In the table we have marked in boldface the results according to the following criteria. First we consider the Best Result columns, and for each problem we mark in boldface the best of the best results obtained by the two algorithms. Similarly we compare and mark in boldface the best average results. Then, only for those problems on which the two algorithms obtained the same average result, we mark with boldface the lowest average time.

In the last row of Table V we report the number of wins, that is, the number of times one algorithm was better than the other, one for each of the considered criteria (this corresponds to the number of boldface entries in each column). The "Total wins" row synthetically shows that HAS-SOP has a better performance than MPO/AI+LS on all the measured criteria.

In conclusion, in Table VI we report the new upper bounds obtained by HAS-SOP and by MPO/AI+LS, as well as new lower bounds obtained by a branch-and-cut program run by Ascheuer (1997) starting from HAS-SOP solutions. The first column gives the problem names, the second column gives the size of the problem in terms of the number n of nodes, the third columns gives the number $|R|$ of constraints, and the fourth column the bounds reported in TSPLIB. The other columns report the new upper and lower bounds we computed, and finally the "All Best" columns report the best solutions computed by the HAS-SOP and MPO/AI+LS algorithms. In parentheses are the results obtained applying a post-optimization consisting of rerunning the algorithm (HAS-SOP or MPO/AI), starting from the best found solution but using as local search one of the variants presented in Table I (the post-optimization was run for all

Table VI. New Bounds for Sequential Ordering Problems*

PROB	n	|R|	TSPLIB Bounds	NEW Lower Bounds	NEW Upper Bounds	All Best HAS-SOP	All Best MPO/AI+LS
ESC63	65	95	62			62	62
ESC78	80	77	18230			18230	18230
ft53.1	54	12	[7438, 7570]		7531	7531	7531
ft53.2	54	25	[7630, 8335]		8026	8026	8026
ft53.3	54	48	[9473, 10935]		10262	10262	10262
ft53.4	54	63	14425			14425	14425
ft70.1	71	17	39313			39313	39313
ft70.2	71	35	[39739, 40422]	39803	40419	40419	40419
ft70.3	71	68	[41305, 42535]			42535	42535
ft70.4	71	86	[52269, 53562]	53072	53530	53530	53530
kro124p.1	101	25	[37722, 40186]	37761	39420	39420	39420
kro124p.2	101	49	[38534, 41677]	38719	41336	41336	41336
kro124p.3	101	97	[40967, 50876]	41578	49499	49499	49519
kro124p.4	101	131	[64858, 76103]			76103	76103
prob.100	100	41	[1024, 1385]	1027	1190	1219 (1190)	1573
rbg109a	111	622	1038			1038	1038
rbg150a	152	952	[1748, 1750]			1750	1750
rbg174a	176	1113	2033			2033	2033
rbg253a	255	1721	[2928, 2987]	2940	2950	2950	2950
rbg323a	325	2412	[3136, 3157]	3137	3141	3141	3141
rbg341a	343	2542	[2543, 2597]		2570	2576 (2574)	2572 (2570)
rbg358a	360	3239	[2518, 2599]	2529	2545	2549 (2545)	2555
rbg378a	380	3069	[2761, 2833]		2816	2817	2816

*New upper bounds were obtained by HAS-SOP and MPO/AI+LS, while new lower bounds were obtained by a branch-and-cut program starting from HAS-SOP solutions. "All Best" columns report the best solutions computed by the two algorithms. In parentheses are the results obtained by applying the post-optimization.

the problems, but only in four cases was it able to improve the best solution found in the first optimization phase).

6. Conclusions

The contribution of this paper is two-fold. First, we have introduced a novel local search procedure for the SOP called SOP-3-exchange. This procedure has been shown to produce solutions of quality higher than those produced by MPO/AI, the previous best-known algorithm for the SOP. This has been shown to be the case even when the local search is applied to poor-quality, randomly generated initial solutions. Second, we have shown that the performance of the algorithm obtained by coupling MPO/AI with SOP-3-exchange can still be improved by coupling the local search with ACS-SOP, a straightforward extension of Ant Colony System (Dorigo and Gambardella 1997). The resulting algorithm, called HAS-SOP, is currently the best available algorithm for the SOP. Both HAS-SOP and MPO/AI with local search were able to improve the upper bounds for most of the standard test problems used (and available in TSPLIB).

Acknowledgments

We wish to thank Norbert Ascheuer for checking the optimality of our results with his branch-and-cut program, as well as the Associate Editor, the anonymous referees, and Andrea Rizzoli for their many useful comments. Marco Dorigo acknowledges support from the Belgian FNRS, of which he is a Research Associate.

References

Aarts, E.H., J.K. Lenstra. 1997. Introduction. E.H. Aarts, J.K. Lenstra, eds. *Local Search in Combinatorial Optimization*. John Wiley & Sons, Chichester, U.K. 1–17.

Ascheuer, N. 1995. Hamiltonian path problems in the on-line optimization of flexible manufacturing systems. Ph.D. thesis, Technische Universität, Berlin, Germany.

Ascheuer, N. 1997. Personal communication.

Ascheuer, N., L.F. Escudero, M. Grotschel, M. Stoer. 1993. A cutting plane approach to the sequential ordering problem (with applications to job scheduling in manufacturing). *SIAM Journal on Optimization 3*, 25–42.

Balas, E., M. Fischetti, W.R. Pulleyblank. 1995. The precedence-constrained asymmetric traveling salesman polytope. *Mathematical Programming 65*, 241–265.

Battiti, R., G. Tecchiolli. 1994. The reactive tabu search. *ORSA Journal on Computing 6*, 126–140.

Bentley, J.L. 1992. Fast algorithms for geometric traveling salesman problem. *ORSA Journal on Computing 4*, 387–411.

Chen, S., S. Smith. 1996. Commonality and genetic algorithms. Technical Report CMU-RI-TR-96-27, The Robotic Institute, Carnegie Mellon University, Pittsburgh, PA.

Connolly, D.T. 1990. An improved annealing scheme for the QAP. *European Journal of Operational Research 46*, 93–100.

Dorigo, M. 1992. Ottimizzazione, apprendimento automatico, ed algoritmi basati su metafora naturale. [Optimization, learning and natural algorithms.] Doctoral dissertation, Dipartimento di Elettronica e Informazione, Politecnico di Milano, Italy.

Dorigo, M., G. Di Caro. 1999. The Ant Colony Optimization metaheuristic. D. Corne, M. Dorigo, F. Glover, eds. *New Ideas in Optimization*. McGraw-Hill, London, U.K. 11–32.

Dorigo, M., G. Di Caro, L.M. Gambardella. 1999. Ant algorithms for discrete optimization. *Artificial Life 5*, 137–172.

Dorigo, M., L.M. Gambardella. 1997. Ant Colony System: a cooperative learning approach to the traveling salesman problem. *IEEE Transactions on Evolutionary Computation 1*, 53–66.

Dorigo, M., V. Maniezzo, A. Colorni. 1991. The Ant System: an autocatalytic optimizing process. Technical Report No. 91-016, Dipartimento di Elettronica, Politecnico di Milano, Italy.

Dorigo, M., V. Maniezzo, A. Colorni. 1996. The Ant System: optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics—Part B 26*, 29–41.

Escudero, L.F. 1988. An inexact algorithm for the sequential ordering problem. *European Journal of Operational Research 37*, 232–253.

Escudero, L.F., M. Guignard, K. Malik. 1994. A Lagrangian relax-and-cut approach for the sequential ordering problem with precedence relationships. *Annals of Operations Research 50*, 219–237.

Fleurent, C., J. Ferland. 1994. Genetic hybrids for the quadratic assignment problem. *DIMACS Series in Mathematical Theoretical Computer Science 16*, 190–206.

Fogel, D.B. 1994. *Evolutionary Computation: Toward a New Philosophy of Machine Intelligence*. IEEE Press, New York.

Gambardella, L.M., M. Dorigo. 1995. Ant-Q: a reinforcement learning approach to the traveling salesman problem. A. Prieditis, S. Russell, eds. *Proceedings of ML-95, Twelfth International Conference on Machine Learning*, *Tahoe City, CA, July 1995,* Morgan Kaufmann, Palo Alto, CA. 252–260.

Gambardella, L.M., M. Dorigo. 1996. Solving symmetric and asymmetric TSPs by ant colonies. *Proceedings of IEEE International Conference on Evolutionary Computation, IEEE-EC 96, Nagoya, Japan, May 1996,* IEEE Press, Piscataway, NJ. 622–627.

Gambardella, L.M., M. Dorigo. 1997. HAS-SOP: an Hybrid Ant System for the sequential ordering problem. Technical Report, IDSIA/11-97, IDSIA, Lugano, Switzerland.

Gambardella, L.M., É.D. Taillard, G. Agazzi. 1999a. MACS-VRPTW: a multiple Ant Colony System for vehicle routing problems with time windows. D. Corne, M. Dorigo, F. Glover, eds. *New Ideas in Optimization*. McGraw-Hill, London, U.K. 63–76.

Gambardella, L.M., É.D. Taillard, M. Dorigo. 1999b. Ant colonies for the quadratic assignment problem. *Journal of the Operational Research Society 50*, 167–176.

Glover, F. 1989a. Tabu search, Part I. *ORSA Journal on Computing 1*, 190–206.

Glover, F. 1989b. Tabu search, Part II. *ORSA Journal on Computing 2*, 4–32.

Holland, J.H. 1975. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor, MI.

Johnson, D.S., L.A. McGeoch. 1997. The traveling salesman problem: a case study. E.H. Aarts, J.K. Lenstra, eds. *Local Search in Combinatorial Optimization*. John Wiley & Sons, Chichester, U.K. 215–310.

Kindervater, G.A.P., M.W.P. Savelsbergh. 1997. Vehicle routing: handling edge exchanges. E.H. Aarts, J.K. Lenstra, eds. *Local Search in Combinatorial Optimization*. John Wiley & Sons, Chichester, U.K. 311–336.

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

Lin, S. 1965. Computer solutions of the traveling salesman problem. *Bell Systems Journal 44*, 2245–2269.

Lin, S., B.W. Kernighan. 1973. An effective heuristic algorithm for the traveling salesman problem. *Operations Research 21*, 498–516.

Or, I. 1976. Traveling salesman-type combinatorial problems and their relation to the logistics of blood banking. Ph.D. thesis, Department of Industrial and Engineering and Management Science, Northwestern University, Evanston, IL.

Psaraftis, H.N. 1983. K-interchange procedures for local search in a precedence-constrained routing problem. *European Journal of Operational Research 13*, 341–402.

Pulleyblank, W., M. Timlin. 1991. Precedence constrained routing and helicopter scheduling: heuristic design. IBM Technical Report RC17154 (#76032), IBM T.J. Watson Research Center, Yorktown Heights, NY.

Reinelt, G. 1994. *The Traveling Salesman: Computational Solutions for TSP Applications*. LNCS 840, Springer-Verlag, Berlin, Germany.

Savelsbergh, M.W.P. 1990. An efficient implementation of local search algorithms for constrained routing problems. *European Journal of Operational Research 47*, 75–85.

Solomon, M.M. 1987. Algorithms for the vehicle routing and scheduling problems with time windows constraints. *Journal of Operational Research 35*, 254–265.

Taillard, É.D. 1991. Robust tabu search for the quadratic assignment problem. *Parallel Computing 17*, 443–455.

Van der Bruggen, L.J.J., L.K. Lenstra, P.C. Schuur. 1993. Variable-depth search for the single-vehicle pickup and delivery problem with time windows. *Transportation Science 27*, 298–311.