A new precedence-based Ant Colony Optimization for permutation problems

Marco Baioletti[1], Alfredo Milani[1,2], and Valentino Santucci[1]

[1] Department of Mathematics and Computer Science, University of Perugia, Italy
[2] Department of Computer Science, Hong Kong Baptist University, Hong Kong
{marco.baioletti,alfredo.milani,valentino.santucci}@unipg.it

Abstract. In this paper we introduce ACOP, a novel ACO algorithm for solving permutation based optimization problems. The main novelty is in how ACOP ants construct a permutation by navigating the space of partial orders and considering precedence relations as solution components. Indeed, a permutation is built up by iteratively adding precedence relations to a, initially empty, partial order of items until it becomes a total order, thus the corresponding permutation is obtained. The pheromone model and the heuristic function assign desirability values to precedence relations. An ACOP implementation for the Linear Ordering Problem (LOP) is proposed. Experiments have been held on a large set of widely adopted LOP benchmark instances. The experimental results show that the approach is very competitive and it clearly outperforms previous ACO proposals for LOP.

Keywords: Ant Colony Optimization, Permutations Representation, Partial Orders, Linear Ordering Problem

1 Introduction

Ant Colony Optimization (ACO) [6] is a popular meta-heuristic scheme for solving hard combinatorial optimization problems inspired by the foraging behavior of natural ant colonies. Since the seminal work of Dorigo in the early '90s [8], ACO has been extensively and successfully applied to permutation-based optimization problems, i.e., problems where a solution is a permutation of items. See for example the several ACO proposals for the traveling salesman problem [8, 7], the quadratic assignment problem [9] or the permutation flowshop scheduling problem [19].

The typical ACO approach to permutation problems is to consider the items (to be ordered) as the solution components. Therefore, an artificial ant constructs a n-length permutation by starting from the empty sequence and iteratively adding up items until all the n items appear in the sequence exactly once. Usually, the items are sequentially inserted from left to right, though simple variations, where the items can be inserted in arbitrary positions of the (partial) permutation, have been proposed [15].

To the best of our knowledge, this construction scheme is applied by all the ACO proposals for permutation problems available in literature. Basically, they all represent a permutation of n items directly as a n-length sequence without repetitions, thus, during the construction process, a partial solution is a sequence containing empty slots.

Though this representation is very natural, it is not always the most suited for the problem at hand. This is the case of the objective functions defined by means of non-local characteristics such as the precedence relations between the pairs of items contained in a permutation. Indeed, precedences are important in a variety of problems such as some scheduling problems [3, 14] and, in particular, in the Linear Ordering Problem (LOP) [16] where the objective function is defined as the sum of the contributions associated to all the precedences encoded by a permutation. It is evident that, in this case, adding a new item to a partial sequence induces a drastic change to the expected objective value of the permutation under construction.

In this paper, we propose the smoother approach of constructing a permutation by iteratively adding precedences to a, initially empty, partial order until a total order, i.e., a permutation, is formed. Basing on this idea, a new precedence-based ACO approach for permutation problems, namely ACOP, is introduced. Therefore, the main novelty is that ACOP ants represent a partial permutation, i.e., a partial order, as a collection of precedence relations (between items) which are consistent each other. When this collection contains $\binom{n}{2}$ precedences, then a n-length permutation is mathematically guaranteed to be built.

ACOP has been implemented and applied to LOP. Experiments have been held on a large set of widely adopted benchmark instances where ACOP performances have been compared to state-of-the-art results. Moreover, a further experiment has been held to compare ACOP with, as far as we know, the only ACO approach to LOP available in literature, i.e., ACS-IM [5, 18].

2 Ant Colony Optimization

2.1 ACO General Scheme

ACO algorithms [8, 9, 15] are inspired by the stigmergic foraging behavior of natural ant colonies. When a real ant discover a food source, it walks back to its nest by also leaving pheromone trails on the way, so other ants can sense the trails and reach the food themselves. Analogously, in ACO, artificial ants build up combinatorial solutions component-by-component using a probabilistic construction procedure biased by the artificial pheromone trails deposited on solution components by the best-performing ants of the previous iterations.

Let $f : S \to \mathbb{R}$ be the objective function to be optimized, where S is the set of solutions, then each $s \in S$ is composed by a certain number of components $c_1, c_2, \ldots, c_n$ taken from the set of possible components C. Clearly, S and C are problem dependent. For example, in the traveling salesman problem, C is the set of cities, while S contains all the permutations of C.

ACO aims to optimize f by iteratively probing S by means of N artificial ants. The ants indirectly communicate through a common data structure, called pheromone, which associates a real value τ_c to each solution component $c \in C$. The main scheme of ACO is depicted in Figure 1, where, without loss of generality, maximization is assumed.

```
function ACO(N, α, β, ρ, Δ_τ^ib, Δ_τ^gb)
    Initialize pheromone values τ_c for all c ∈ C
    s^gb ← null
    while termination condition is not met do
        s^ib ← null
        for i ← 1 to N do
            s_i ← BuildSolution(α, β)
            Evaluate f(s_i)
            if f(s_i) > f(s^ib) then
                s^ib ← s_i
            end if
            if f(s_i) > f(s^gb) then
                s^gb ← s_i
            end if
        end for
        Optionally perform a local search on s^ib (and update s^gb)
        EvaporatePheromone(ρ)
        DepositPheromone(s^ib, s^gb, Δ_τ^ib, Δ_τ^gb)
    end while
    return s^gb
end function
```

Fig. 1. General scheme of ACO

Pheromone values are usually initialized to a constant value. Then, at every ACO iteration, each ant starts from an empty partial solution and builds up a complete solution by iteratively choosing components from C. In many problems, the set C_t of feasible components at construction step t is restricted by the choices done in the previous steps, thus, in general, $C_t \subseteq C$. The choice of a component c from C_t is influenced by its pheromone value $\tau_c \in \mathbb{R}^+$ and a problem dependent heuristic value $\eta_c \in \mathbb{R}^+$ which estimates the contribution of c to the solution quality. Formally, the probability of choosing $c \in C_t$ is

$$p(c) = \frac{\tau_c^\alpha \eta_c^\beta}{\sum_{k \in \mathcal{C}_t} \tau_k^\alpha \eta_k^\beta}, \qquad (1)$$

where the parameters $\alpha, \beta \in \mathbb{R}$ determine the influence of, respectively, pheromone and heuristic values.

The construction process terminates when N complete solutions (one per ant) have been generated. Each solution is evaluated using f and the pheromone is updated. First, for all $c \in C$, the pheromone value τ_c is evaporated as follows

$$\tau_c \leftarrow (1-\rho)\tau_c, \tag{2}$$

where $\rho \in [0,1]$ is the evaporation rate parameter. Then, a given amount of pheromone is deposited on the components belonging to the best solutions. Though various deposition strategies are possible [15], the most common ones consider the iteration and global best solutions, respectively, s^{ib} and s^{gb}. Formally, for all $c \in C$, the pheromone value τ_c is updated as

$$\tau_c \leftarrow \tau_c + I\left(c \in s^{ib}\right) \Delta_\tau^{ib} + I\left(c \in s^{gb}\right) \Delta_\tau^{gb}, \tag{3}$$

where: $I\left(c \in s\right)$ is 1 if component c belongs to solution s and 0 otherwise, while $\Delta_\tau^{ib}, \Delta_\tau^{gb} \in \mathbb{R}^+$ are the "awards" of pheromones for the components of, respectively, the iteration and global best solutions.

Finally, note that, before pheromone update, a local search refinement can be optionally applied to a selected set of solutions (usually, the iteration best).

2.2 Pheromone Models for Permutation Problems

While the typical permutation construction procedure of ACO schemes has been described in Section 1, here we provide a brief overview of the different pheromone models for permutation problems available in literature [3, 17].

One simple approach, denoted as PH_{abs} in [3], is to associate pheromone values to pairs composed by an item and an absolute position, in order to indicate the desirability to have a given item at a given position in the permutation.

Two other relevant approaches are PH_{suc} and PH_{rel} [3], which both assign pheromone values to ordered pairs of items. While PH_{suc} aims to encode the desirability of having the two items in consecutive positions of the permutation, PH_{rel} is less stringent and only indicates the desirability of the precedence relation between the two items independently of their distance in the permutation.

We highlight that, as far as we know, all the ACO proposals in literature using the pheromone model PH_{rel} build up the permutation as seen in Section 1, i.e., by iteratively adding items to a incumbent sequence till it becomes a complete permutation.

3 Permutations, partial and total orders

Here, we provide a brief mathematical background useful to describe the representation of the (partial) solutions in ACOP. In particular: we introduce an encoding for generic partial orders of items, we show under which conditions the partial order is also a total order and how to obtain its corresponding permutation.

Let I be a finite set of items that, without loss of generality, can be taken as $I = \{1, \ldots, n\}$, then a strict partial order relation $\prec$ on I is a binary relation which satisfies the following properties:

– Irreflexivity: $a \not\prec a$, for all $a \in I$;
– Transitivity: if $a \prec b$ and $b \prec c$, then $a \prec c$, for $a, b, c \in I$;
– Anti-symmetry: if $a \prec b$, then $b \not\prec a$, for $a, b \in I$.

As any binary relation, a partial order $\prec$ on I can be represented as the set of pairs $P = \{(a, b) : a, b \in I \text{ and } a \prec b\}$, thus the pair $(a, b) \in P$ indicates the precedence $a \prec b$.

Conversely, given any finite set of precedences $P = \{(a_1, b_1), \ldots, (a_k, b_k)\}$, where $a_i, b_i \in I$ for $i = 1, \ldots, k$, such that the precedences in P do not violate any partial order property, it is possible to find the corresponding partial order $\prec_P$ on I generated by P, i.e., the smallest partial order which respects all the precedences in P. Operatively, $\prec_P$ is the transitive closure P^* of P, which is computed as follows. Let $P_0 = P$, then

$$P_{r+1} = P_r \cup \{(a, b) : \exists c \in I \text{ such that } (a, c), (c, b) \in P_r\}. \tag{4}$$

After a finite number s of steps, the sequence of sets $\langle P_r \rangle_r$ stabilizes (i.e., $P_s = P_{s+i}$ for any integer $i \geq 1$), because the maximum number of "compatible" precedences is finite and equal to $\binom{n}{2}$. Hence, $P^* = P_s$ and $a \prec_P b$ if and only if $(a, b) \in P^*$.

Importantly, the partial order $\prec_P$, represented by a given set of pairs P, can also be seen as the arcs set of the digraph $G_\prec$ whose nodes set is I and such that there is an arc $a \to b$ for each precedence $(a, b) \in P$. Therefore, a partial order P can be encoded by the $n \times n$ incidence matrix A of $G_\prec$, whose entries are

$$A_{ab} = \begin{cases} 1 & \text{if } a \prec b \\ -1 & \text{if } b \prec a \\ 0 & \text{otherwise.} \end{cases} \tag{5}$$

Furthermore, if $\prec$ also satisfies the property that for all $a, b \in I$, with $a \neq b$, either $a \prec b$ or $b \prec a$, then $\prec$ is a strict total order. For a total order, the matrix A does not contain any 0-entry, except in its main diagonal. Moreover, it contains $\binom{n}{2}$ 1-entries and the same number of -1s.

It is easy to see that there is a bijective correspondence between the set of total orders on I and the set $\mathcal{S}_n$ of the permutations of I. Indeed, given $\pi \in \mathcal{S}_n$, its corresponding total order $\prec_\pi$ is defined as all the precedences $a \prec_\pi b$ such that $a, b \in I$ and a appears before b in π. More formally, $a \prec_\pi b$ if and only if $\pi^{-1}(a) < \pi^{-1}(b)$, where π^{-1} is the inverse permutation of π. On the other hand, if $\prec$ is a total order, the corresponding permutation $\pi_\prec$ is recursively defined as: (i) $\pi_\prec(1) = a$ where $a \in I$ is the unique item such that $b \not\prec a$ for every $b \in I$, and (ii) $\pi_\prec(k) = a$ if $b \prec a$ only for all the $b \in \{\pi_\prec(1), \ldots, \pi_\prec(k-1)\}$.

Therefore, given a total order encoded by a matrix A (see equation (5)), the corresponding permutation π can be recovered by observing that $\pi(k) = a$ if and only if the a-th row of A has exactly $n - k$ 1s. Hence, by setting $\sigma(a)$ to n minus the number of 1-entries in the a-th row of A, for all $a = 1, \ldots, n$, and observing that σ is a permutation, then $\pi = \sigma^{-1}$.

As a further interpretation, note that a partial order $\prec$ individuates the set of permutations $Q_\prec \subseteq \mathcal{S}_n$ such that $\pi \in Q_\prec$ if and only if π agrees with $\prec$, i.e., for all $a, b \in I$, if $a \prec b$ then $\pi^{-1}(a) < \pi^{-1}(b)$.

Finally, given a partial order $\prec$ and a new pair (c, d), with $c, d \in I$ and $c \neq d$, such that $d \not\prec c$, it is possible to extend $\prec$ with the precedence $c \prec d$ by simply computing the transitive closure of the set $P_\prec \cup \{(c, d)\}$.

4 ACOP: Ant Colony Optimization on Precedences

ACOP is a new Ant Colony Optimization algorithm for permutation based optimization problems which works on the space of partial orders. Its aim is to optimize an objective function of the form $f : \mathcal{S}_n \to \mathbb{R}$, where $\mathcal{S}_n$ contains all the permutations of a set I of n items.

The main structure of ACOP follows the same ACO general scheme depicted in Figure 1. It handles a colony of N artificial ants and uses the pheromone model PH_{rel} previously described in Section 2.2, i.e., pheromone values are maintained in a $n \times n$ matrix where the entry $\tau_{a,b}$, with $a, b \in I$, is the amount of pheromone assigned to precedence $a \prec b$.

The original parts of ACOP are: (i) the (partial) solution representation, and (ii) the construction procedure performed by the artificial ants, i.e., the implementation of *BuildSolution* (see Figure 1).

Indeed, every ant builds up a permutation by iteratively adding precedence relations to a partial order $\prec$, which is initially empty, until it becomes a total order. The pseudo-code of the procedure is depicted in Figure 2.

procedure BuildSolution(α, β)
  $A \leftarrow \mathbf{0}$ ▷ All 0s in matrix A
  $np \leftarrow 0$ ▷ Number of precedences in A
  while $np < \binom{n}{2}$ **do**
    $C = \{(a, b) : A_{a,b} = 0 \text{ and } a \neq b\}$ ▷ Candidate set of precedences
    $(a, b) \leftarrow \text{ChoosePrec}(C, \tau, \eta, \alpha, \beta)$ ▷ See equation (1)
    $Q \leftarrow \{(a, b)\}$
    while $Q \neq \emptyset$ **do** ▷ Insert (a, b) and compute the transitive closure
      $(a, b) \leftarrow$ remove an element from Q
      $A_{a,b} \leftarrow 1$
      $A_{b,a} \leftarrow -1$
      $np \leftarrow np + 1$
      $Q \leftarrow Q \cup \{(a, c) : A_{a,c} = 0 \text{ and } A_{b,c} = 1\}$
        $\cup \{(c, b) : A_{c,b} = 0 \text{ and } A_{c,a} = 1\}$
    end while
  end while
  Return A
end procedure

Fig. 2. The permutation construction procedure of ACOP

The matrix A encodes the partial order $\prec$ that is initially empty, while the variable np is the number of 1s in A, i.e., the number of precedences of $\prec$. The loop of lines 4–16 iteratively adds 1-entries to A and terminates when A contains exactly $\binom{n}{2}$ 1-entries, i.e., when A encodes a total order which corresponds to a permutation. At each construction step, the ant chooses a precedence from C (line 5). This precedence can be safely added to $\prec$. The choice of line 6 is performed by considering pheromone and heuristic values as in the general ACO scheme provided in equation (1). The inner loop at lines 8–15 inserts the selected precedence in $\prec$, by removing some 0-entries from A and iteratively computing the transitive closure on A as described in Section 3. Finally, though, at the end of the procedure, the matrix A can be converted to a permutation (see Section 3), some objective functions can be directly computed on A, therefore *BuildSolution* returns the matrix A.

5 Application of ACOP to LOP

The Linear Ordering Problem (LOP) is a classical NP-Hard combinatorial optimization problem [16] and has received considerable attention because of its many applications in diverse research fields such as economy [13], graph theory [4], archeology [10] and computational social choice [12].

LOP can be straightforwardly formulated as a matrix triangulation problem [16]. Given a $n \times n$ matrix H, LOP requires to find a permutation $\pi \in \mathcal{S}_n$ of the row and column indices $\{1, \ldots, n\}$ that maximizes the objective function

$$f(\pi) = \sum_{i=1}^{n} \sum_{j=i+1}^{n} H_{\pi(i),\pi(j)} \tag{6}$$

The permutation structure of the LOP solutions allows to apply a variety of meta-heuristics and evolutionary algorithms specifically designed for the permutations search space. See for instance [1, 2, 11, 20]. To the best of our knowledge, the only ACO approach to LOP has been proposed by Pintea et al. in [5] and [18].

An interesting observation is that the objective function of LOP can be directly computed on the permutation representation used by ACOP, i.e., on the matrix A returned by the function *BuildSolution* depicted in Figure 2. Indeed, it is easy to see that equation (6) can rewritten as

$$f(A) = \sum_{a=1}^{n} \sum_{b=1}^{n} H_{a,b} \cdot \max\{A_{a,b}, 0\}. \tag{7}$$

Here, it is evident that the contribution of any single precedence relation (a, b) present in the permutation to be evaluated is exactly $H_{a,b}$. Hence, a simple but effective choice for the heuristic function $\eta_{a,b}$ is to set $\eta_{a,b} = H_{a,b} + \epsilon$, where ϵ is a small positive quantity introduced to avoid null probabilities when $H_{a,b} = 0$.

Further details of the implementation of ACOP for LOP are as follows. Pheromone is deposited using a mix of the iteration-best and global-best strategies as depicted by equation (3). Inspired by [21], the pheromone values are constrained to the interval $[\tau_{min}, \tau_{max}]$, where $\tau_{max} = (\Delta_\tau^{ib}+\Delta_\tau^{gb})/\rho$ and $\tau_{min} = \tau_{max}/(2n^2-n)$. All the pheromones are initialized to τ_{max}. *ChoosePrec* (line 6 of Figure 2) has been implemented as in [7], i.e., with probability q_0 the most probable precedence is chosen, otherwise a tournament is performed. Precedence probabilities are computed as in equation (1).

Finally, an enhanced variant of ACOP, called ACOP$^+$, has been devised. ACOP$^+$ performs a local search refinement on the iteration best solution at the end of every iteration. The local search has been implemented by iteratively applying the best item insertion move till no improvement is observed (see [16]). Moreover, in order to avoid stagnation, ACOP$^+$ reinitializes the pheromone values if no improvement to the global best solution has been observed during the last r iterations.

6 Experiments

The ACOP application to LOP has been experimentally investigated on the three widely known benchmark suites LOLIB, SGB and MB[3]. Therefore, a total of 105 LOP instances, with dimensionalities ranging from 44 to 250, has been considered.

The optima of these instances are known[4] and they have been used to compute two performance measures: the success rate (SR), and the average relative percentage deviation (ARPD). An algorithm is executed k times per instance, thus SR indicates the percentage of executions that reach the known optimum, while ARPD $= \frac{100}{k}\sum_{i=1}^{k}\frac{opt-run_i}{opt}$ is the average percentage deviation from the known optimum.

ACOP parameters have been experimentally tuned on a subset of 10 selected instances: the (lexicographically) first instances for every dimensionality available in the benchmarks. A set of settings have been individuated by some preliminary experiments, then a full factorial experimental design has been considered in order to choose the best setting. The involved parameters and their values are: $N \in \{20, 50, 100\}$, $\alpha, \beta \in \{1, 2\}$, $\rho \in \{0.05, 0.1, 0.2\}$, $q_0 \in \{0, 0.01, 0.1\}$, and $\left(\Delta_\tau^{ib}, \Delta_\tau^{gb}\right) \in \{(10, 0), (7.5, 2.5)\}$. Therefore, a total of 216 settings have been tested by performing 10 executions per instance with a termination criterion of 60 seconds. Then, the average rank of the ARPDs obtained in every instance are computed and the setting with the best average rank is chosen as the reference configuration of ACOP. This setting is $\left(N = 20, \alpha = 2, \beta = 2, \rho = 0.05, q_0 = 0.1, (\Delta_\tau^{ib}, \Delta_\tau^{gb}) = (7.5, 2.5)\right)$.

The tuned setting has been used both for ACOP and ACOP$^+$. The further parameter r of ACOP$^+$ has been set to $r = 50$. Then, ACOP and ACOP$^+$ have

[3] The instances are available from http://www.optsicom.es/lolib.

[4] During the years and using a considerably large amount of computational time, they have been proved to be optima using exact methods [16].

been executed 20 times on every instance. The termination criteria adopted are: 120 seconds for LOLIB instances, 300 seconds for SGB instances, and 600 seconds for the larger MB instances. All the experiments have been run on a homogeneous cluster of computers equipped with Intel Xeon X5650 processors clocking at 2.67GHz. The SR, ARPD and the median time where the global best solution has been found are reported in Tables 1 (LOLIB) and 2 (SGB and MB).

Table 1. Experimental Results on LOLIB instances

Instance		**ACOP**			**ACOP$^+$**			**Instance**		**ACOP**			**ACOP$^+$**		
Name	n	SR	ARPD	Time	SR	ARPD	Time	Name	n	SR	ARPD	Time	SR	ARPD	Time
N-t59b11xx	44	95	0.0004	0.437	100	0	0.133	N-t75d11xx	44	40	0.0009	4.145	100	0	0.089
N-t59d11xx	44	100	0	0.319	100	0	0.038	N-t75e11xx	44	100	0	0.854	100	0	0.046
N-t59f11xx	44	100	0	0.303	100	0	0.061	N-t75i11xx	44	100	0	1.463	100	0	0.227
N-t59i11xx	44	100	0	0.140	100	0	0.024	N-t75k11xx	44	100	0	0.330	100	0	0.024
N-t59n11xx	44	100	0	0.026	100	0	0.020	N-t75n11xx	44	100	0	0.138	100	0	0.022
N-t65b11xx	44	0	0.0163	1.003	100	0	0.115	N-t75u11xx	44	100	0	0.196	100	0	0.024
N-t65d11xx	44	100	0	0.467	100	0	0.056	N-be75eec	50	100	0	0.553	100	0	0.081
N-t65f11xx	44	100	0	0.195	100	0	0.023	N-be75np	50	0	0.0062	18.025	15	0.0002	0.172
N-t65i11xx	44	100	0	0.298	100	0	0.070	N-be75oi	50	75	0.0003	45.038	80	0.0002	0.090
N-t65l11xx	44	100	0	0.007	100	0	0.010	N-be75tot	50	90	0.0003	1.031	100	0	0.214
N-t65n11xx	44	100	0	0.191	100	0	0.051	N-tiw56n54	56	95	< 0.0001	1.493	100	0	0.780
N-t65w11xx	44	100	0	0.345	100	0	0.023	N-tiw56n58	56	100	0	0.767	100	0	0.312
N-t69r11xx	44	100	0	0.067	100	0	0.024	N-tiw56n62	56	95	0.0011	1.780	95	0.0011	0.183
N-t70b11xx	44	100	0	0.295	100	0	0.028	N-tiw56n66	56	100	0	1.618	100	0	0.130
N-t70d11xx	44	10	0.0012	0.654	100	0	0.022	N-tiw56n67	56	60	0.0891	8.158	90	0.0218	0.364
N-t70d11xxb	44	100	0	0.675	100	0	0.047	N-tiw56n72	56	65	0.0008	28.039	100	0	0.259
N-t70f11xx	44	100	0	0.145	100	0	0.043	N-tiw56r54	56	60	0.0017	2.643	95	0.0009	0.637
N-t70i11xx	44	100	0	0.227	100	0	0.087	N-tiw56r58	56	100	0	1.509	100	0	0.186
N-t70k11xx	44	60	0.0041	0.907	100	0	0.039	N-tiw56r66	56	100	0	1.892	100	0	0.151
N-t70l11xx	44	100	0	0.033	100	0	0.043	N-tiw56r67	56	55	0.0002	1.610	100	0	0.089
N-t70n11xx	44	100	0	0.150	100	0	0.023	N-tiw56r72	56	95	< 0.0001	1.503	100	0	0.101
N-t70u11xx	44	100	0	0.019	100	0	0.019	N-stabu70	60	0	0.0242	4.791	80	0.0052	1.670
N-t70w11xx	44	100	0	0.322	100	0	0.023	N-stabu74	60	35	0.0160	5.059	100	0	0.913
N-t70x11xx	44	100	0	0.464	100	0	0.023	N-stabu75	60	15	0.0418	4.303	95	0.0007	0.994
N-t74d11xx	44	45	0.0010	1.187	100	0	0.046	N-usa79	79	10	0.0306	25.779	30	0.0051	6.902
								LOLIB Average		80	0.0047	3.432	96	0.0007	0.316

Tables 1 and 2 clearly show that both ACOP and ACOP$^+$ obtained remarkable performances throughout all the instances of the benchmark suites considered. Regarding the success rates, ACOP obtained the optimum in at least one execution (SR>0) on about the 57% of the instances, while ACOP$^+$ reached the instance optimum on all the 105 instances. Moreover, in 63 cases, ACOP$^+$ reached the optimum in all the executions performed (SR=100). Most notably, also when the optimum is not reached, the very small ARPDs clearly show that both ACOP and ACOP$^+$ have been able to obtain very high quality solutions. Indeed, the worst ARPD of ACOP, obtained in the N-sgb75.19 instance (see Table 2), is of only the 0.1057%, while for ACOP$^+$ it is even smaller, i.e., 0.0218% in N-tiw56n57 (see Table 1). Furthermore, though the computational time to reach the best solution increases with the instance size n, the average times reported at the end of the tables show that the two algorithms are able to provide high quality solutions in a reasonable amount of time.

Table 2. Experimental Results on SGB (left) and MB (right) instances

Instance		ACOP			ACOP$^+$			Instance		ACOP			ACOP$^+$		
Name	n	SR	ARPD	Time	SR	ARPD	Time	Name	n	SR	ARPD	Time	SR	ARPD	Time
N-sgb75.01	75	0	0.0670	17.789	50	0.0063	2.983	N-r100a2	100	0	0.0155	29.261	65	0.0005	10.407
N-sgb75.02	75	0	0.0551	33.936	35	0.0002	6.568	N-r100b2	100	0	0.0224	31.812	5	0.0054	10.828
N-sgb75.03	75	0	0.0197	25.826	100	0	1.570	N-r100c2	100	0	0.0375	40.122	25	0.0053	13.498
N-sgb75.04	75	0	0.0225	28.953	100	0	1.509	N-r100d2	100	90	0.0003	24.370	100	0	6.515
N-sgb75.05	75	0	0.0413	41.994	100	0	1.252	N-r100e2	100	10	0.0015	27.703	75	0.0002	7.730
N-sgb75.06	75	0	0.0291	23.843	70	< 0.0001	8.384	N-r150a0	150	70	0.0004	84.635	100	0	29.493
N-sgb75.07	75	0	0.0311	18.408	100	0	0.932	N-r150a1	150	0	0.0157	137.665	15	0.0005	58.211
N-sgb75.08	75	0	0.0341	28.849	100	0	2.774	N-r150b0	150	65	0.0002	74.576	100	0	8.287
N-sgb75.09	75	0	0.0166	69.700	50	0.0016	68.278	N-r150b1	150	0	0.0051	115.006	5	0.0019	36.030
N-sgb75.10	75	0	0.0413	54.677	100	0	2.720	N-r150c0	150	55	0.0007	84.107	100	0	13.054
N-sgb75.11	75	0	0.0841	28.972	50	0.0003	48.659	N-r150c1	150	5	0.0051	129.176	95	0.0001	47.817
N-sgb75.12	75	0	0.0272	14.141	100	0	2.418	N-r150d0	150	0	0.0044	102.742	90	< 0.0001	31.073
N-sgb75.13	75	0	0.0125	28.362	30	0.0001	4.894	N-r150d1	150	0	0.0079	140.940	35	0.0011	72.893
N-sgb75.14	75	0	0.0258	46.338	60	0.0001	3.622	N-r150e0	150	100	0	71.879	100	0	7.742
N-sgb75.15	75	0	0.0850	34.499	90	0.0009	27.441	N-r150e1	150	5	0.0097	139.383	95	0.0001	50.005
N-sgb75.16	75	0	0.0379	16.061	100	0	3.171	N-r200a0	200	0	0.0016	340.860	100	0	70.777
N-sgb75.17	75	0	0.0545	44.567	100	0	1.640	N-r200a1	200	0	0.0042	403.498	95	< 0.0001	140.697
N-sgb75.18	75	0	0.0435	54.402	70	0.0001	2.671	N-r200b0	200	0	0.0024	335.645	100	0	178.688
N-sgb75.19	75	0	0.1057	19.253	90	< 0.0001	4.057	N-r200b1	200	0	0.0099	374.202	10	0.0009	221.687
N-sgb75.20	75	0	0.0415	14.087	70	< 0.0001	2.573	N-r200c0	200	0	0.0029	305.989	45	0.0003	87.9785
N-sgb75.21	75	0	0.0453	18.634	75	< 0.0001	6.069	N-r200c1	200	0	0.0036	349.472	100	0	89.4125
N-sgb75.22	75	0	0.0839	37.686	100	0	1.579	N-r200d0	200	0	0.0015	317.184	100	0	126.250
N-sgb75.23	75	0	0.0320	73.893	90	0.0005	4.519	N-r200d1	200	0	0.0160	459.413	5	0.0025	169.497
N-sgb75.24	75	0	0.0505	65.582	70	0.0001	6.877	N-r200e0	200	25	0.0004	276.738	100	0	55.321
N-sgb75.25	75	0	0.0613	40.212	80	0.0019	2.638	N-r200e1	200	20	0.0016	306.774	55	0.0004	140.123
								N-r250a0	250	15	0.0009	522.939	95	< 0.0001	148.343
								N-r250b0	250	5	0.0006	528.481	75	< 0.0001	369.338
								N-r250c0	250	10	0.0007	528.663	100	0	168.833
								N-r250d0	250	0	0.0034	557.957	95	< 0.0001	318.197
								N-r250e0	250	0	0.0024	556.871	80	< 0.0001	361.753
SGB Average		0	0.0460	35.227	79	0.0005	8.792	**MB Average**		16	0.0060	246.602	72	0.0006	101.683

Finally, a comparison with the ACO algorithm for LOP proposed in [18], namely ACS-IM, has been performed. The results for ACS-IM have been directly taken from its original paper [18], while ACOP and ACOP$^+$ have been run for 20 executions on their same set of 49 instances (old non-normalized LOLIB instances[5]). The termination criterion has been set to 50 000 iteration as in [18]. The ARPD results are provided in Table 3.

The comparison reported in Table 3 clearly shows that ACOPs perform largely better than ACS-IM, thus promoting our proposal as the first prominent ACO approach to the linear ordering problem.

7 Conclusion and Future Work

ACOP, a new precedence-based ACO algorithm for permutation problems, has been proposed.

With respect to other proposals in literature, the main novelty of ACOP is to consider a permutation as a total order which is obtained through an incremental refinement of a, initially empty, partial order. The refinement process works by

[5] Non-normalized LOLIB instances are available at https://www.iwr.uni-heidelberg.de/groups/comopt/software/LOLIB.

Table 3. Experimental Comparison with ACS-IM on non-normalized LOLIB instances

Instance	*n*	ACOP	ACOP$^+$	ACS-IM	Instance	*n*	ACOP	ACOP$^+$	ACS-IM
t59b11xx	44	0.0097	**0**	0.08	t75d11xx	44	0.0018	**0**	0.59
t59d11xx	44	0.0043	**0**	0.03	t75e11xx	44	0.0009	**0**	0.21
t59f11xx	44	0.0061	**0**	0.02	t75i11xx	44	0.0108	**0**	0.05
t59i11xx	44	0.0001	**0**	0.06	t75k11xx	44	0.0163	**0**	0.02
t59n11xx	44	**0**	**0**	0.19	t75n11xx	44	**0**	**0**	0.04
t65b11xx	44	0.0405	**0**	0.09	t75u11xx	44	0.0007	**0**	0.08
t65d11xx	44	0.0066	**0**	0.18	be75eec	50	0.0004	**0**	0.16
t65f11xx	44	0.0016	**0**	0.14	be75np	50	0.0089	**0.0002**	0.0004
t65i11xx	44	0.0102	**0**	0.19	be75oi	50	0.0023	0.0005	**0.004**
t65l11xx	44	**0**	**0**	0.03	be75tot	50	0.0037	**0.0001**	0.12
t65n11xx	44	0.0441	**0**	0.16	tiw56n54	56	0.0039	**0**	0.13
t65w11xx	44	0.0033	**0**	0.14	tiw56n58	56	0.0024	**0**	0.15
t69r11xx	44	**0**	**0**	0.41	tiw56n62	56	0.0068	**0**	0.08
t70b11xx	44	0.0043	**0**	0.03	tiw56n66	56	0.0057	**0**	0.13
t70d11xn	44	0.0095	**0**	0.05	tiw56n67	56	0.1623	**0**	0.45
t70d11xx	44	0.0005	**0**	0.13	tiw56n72	56	0.0106	**0.0002**	0.17
t70f11xx	44	**0**	**0**	0.16	tiw56r54	56	0.0057	**0.0003**	0.19
t70i11xx	44	**0**	**0**	0.15	tiw56r58	56	0.0085	**0**	0.16
t70k11xx	44	0.0084	**0**	0.05	tiw56r66	56	0.0086	**0**	0.09
t70l11xx	44	**0**	**0**	0.24	tiw56r67	56	0.0101	**0**	0.39
t70n11xx	44	**0**	**0**	0.1	tiw56r72	56	0.0630	**0**	0.11
t70u11xx	44	0.0015	**0**	0.07	stabu1	60	0.0665	**0.0015**	0.26
t70w11xx	44	0.0021	**0**	0.04	stabu2	60	0.0714	**0**	0.27
t70x11xx	44	0.0026	**0**	0.02	stabu3	60	0.0600	**0**	0.27
t74d11xx	44	0.0051	**0**	0.24					
					Average		0.0141	$<$ **0.0001**	0.1454

iteratively adding up a selected precedence relations together with the induced precedences. Also the pheromone model and the heuristic values are defined on the precedence relations.

This approach is particularly suited for those permutation problems where the precedence relations play an important role, for instance in the linear ordering problem (LOP). An ACOP implementation for LOP is then proposed and experimentally validated on a wide set of popular LOP benchmark instances. ACOP is competitive with the state-of-the-art results and clearly outperform the previous ACO proposals for LOP.

Future research directions are: a thorough investigation of the pheromone update strategies in ACOP, the application to other permutation problems and the proposal of a ACO scheme for problems where the solutions are partial orders.

References

1. Baioletti, M., Milani, A., Santucci, V.: Algebraic particle swarm optimization for the permutations search space. In: IEEE Congress on Evolutionary Computation CEC 2017 (in press)
2. Baioletti, M., Milani, A., Santucci, V.: Linear ordering optimization with a combinatorial differential evolution. In: Proc. of 2015 IEEE International Con-

ference on Systems, Man, and Cybernetics, SMC 2015. pp. 2135–2140 (2015), http://dx.doi.org/10.1109/SMC.2015.373
3. Blum, C., Sampels, M.: Ant colony optimization for fop shop scheduling: a case study on different pheromone representations. In: Proc. of the 2002 Congress on Evolutionary Computation, CEC 2002. vol. 2, pp. 1558–1563 (2002)
4. Charon, I., Hudry, O.: An updated survey on the linear ordering problem forweighted or unweighted tournaments. Annals of Op. Res. 175(1), 107–158 (2010)
5. Chira, C., Pintea, C.M., Crisan, G.C., Dumitrescu, D.: Solving the linear ordering problem using ant models. In: Proc. of the 11th Conference on Genetic and Evolutionary Computation, GECCO 2009. pp. 1803–1804. ACM, New York (2009)
6. Dorigo, M., Birattari, M., Stützle, T.: Ant colony optimization. IEEE Computational Intelligence Magazine 1(4), 28–39 (2006)
7. Dorigo, M., Gambardella, L.M.: Ant colony system: a cooperative learning approach to the traveling salesman problem. IEEE Trans. on Evol. Comput. 1(1), 53–66 (1997)
8. Dorigo, M., Maniezzo, V., Colorni, A.: Ant system: optimization by a colony of cooperating agents. IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics) 26(1), 29–41 (1996)
9. Gambardella, L.M., Taillard, E.D., Dorigo, M.: Ant colonies for the quadratic assignment problem. Journal of the Oper. Res. Society 50(2), 167–176 (1999)
10. Glover, F., Klastorin, T., Kongman, D.: Optimal weighted ancestry relationships. Management Science 20(8), 1190–1193 (1974)
11. Gonçalves, J.F., Resende, M.G.C.: Biased random-key genetic algorithms forcombinatorial optimization. Journal of Heuristics 17(5), 487–525 (2011)
12. Kemeny, J.G.: Mathematics without numbers. Daedalus 88(4), 577–591 (1959)
13. Leontief, W.W., Leontief, W.: Input-output economics. Oxford Univ. Press (1986)
14. Li, K., Tang, X., Veeravalli, B., Li, K.: Scheduling precedence constrained stochastic tasks on heterogeneous cluster systems. IEEE Transactions on Computers 64(1), 191–204 (2015)
15. López-Ibánez, M., Stützle, T., Dorigo, M.: Ant colony optimization: A componentwise overview. Techreport, IRIDIA, Universite Libre de Bruxelles (2015)
16. Martí, R., Reinelt, G.: The linear ordering problem: exact and heuristic methods in combinatorial optimization. Springer Science & Business Media (2011)
17. Montgomery, J., Randall, M., Hendtlass, T.: Solution bias in ant colony optimisation: Lessons for selecting pheromone models. Computers & Operations Research 35(9), 2728–2749 (2008)
18. Pintea, C.M., Crisan, G.C., Chira, C., Dumitrescu, D.: A hybrid ant-based approach to the economic triangulation problem for input-output tables. In: Proc. of Hybrid Art. Int. Systems, HAIS 2009. pp. 376–383. Springer, Berlin (2009)
19. Rajendran, C., Ziegler, H.: Ant-colony algorithms for permutation flowshop scheduling to minimize makespan/total flowtime of jobs. European Journal of Operational Research 155(2), 426–438 (2004)
20. Santucci, V., Baioletti, M., Milani, A.: Algebraic differential evolution algorithm for the permutation flowshop scheduling problem with total flowtime criterion. IEEE Transactions on Evolutionary Computation 20(5), 682–694 (2016), http://dx.doi.org/10.1109/TEVC.2015.2507785
21. Stützle, T., Hoos, H.H.: Maxmin ant system. Future Generation Computer Systems 16(8), 889–914 (2000)