# A New Approach for Making Use of Negative Learning in Ant Colony Optimization

Teddy Nurcahyadi(✉) and Christian Blum

Artificial Intelligence Research Institute (IIIA-CSIC),
Campus of the UAB, Bellaterra, Spain
{teddy.nurcahyadi,christian.blum}@iiia.csic.es

**Abstract.** The overwhelming majority of ant colony optimization approaches from the literature is exclusively based on learning from positive examples. Natural examples from biology, however, indicate the potential usefulness of negative learning. Several research works have explored this topic over the last two decades in the context of ant colony optimization, with limited success. In this work we present an alternative proposal for the incorporation of negative learning in ant colony optimization. The results obtained for the capacitated minimum dominating set problem indicate that this approach can be quite useful. More specifically, our extended ant colony algorithm clearly outperforms the standard approach. Moreover, we were able to improve the current state-of-the-art results in 10 out of 36 cases.

## 1 Introduction

Combinatorial optimization (CO) problems are of utmost importance in many real-life scenarios. Large-scale instances of hard CO problems are often solved by heuristic methods. The family of metaheuristics [2] includes techniques based on local search (such as tabu search) and it includes a whole range of bio-inspired techniques such as ant colony optimization and evolutionary algorithms. In this paper we deal with the metaheuristic ant colony optimization (ACO) [5,6], whose development was inspired by the shortest path finding behavior of natural ant colonies. ACO, which is a metaheuristic based on learning, works as follows. At each iteration, a number of artificial ants generate solutions to the tackled optimization problem in a probabilistic way. This is done based on two types of information: greedy information and pheromone information. Then, the best ones of these solutions are used to update the pheromone values, with the aim of moving the probability distribution used for generating solutions to areas of the search space in which high-quality solutions can be found.

As in most metaheuristics based on learning, the type of learning generally used in ACO is *positive learning*, that is, the algorithm tries to learn which components are necessary for assembling high-quality solutions. Nevertheless, learning from negative examples (negative learning) seems to play an important


M. Dorigo et al. (Eds.): ANTS 2020, LNCS 12421, pp. 16–28, 2020.
https://doi.org/10.1007/978-3-030-60376-2_2

role in biological self-organizing systems. Pharaoh ants (*Monomorium pharaonis*), for example, make use of negative trail pheromone in order to deploy 'no entry' signals to mark unrewarding foraging paths [15]. Another example is the use of anti-pheromone hydrocarbons produced by male tsetse flies. These anti-pheromones play an important role in tsetse communications [17]. As already noted in [18], it might therefore be possible to boost the performance of ACO with an additional mechanism that learns (or marks) *undesirable components* by means of a negative feedback mechanisms.

The research community has made several attempts to take benefit from negative learning. Maniezzo [11] and Cordón et al. [4] were the first ones to introduce an active decrease of pheromone values based on low-quality solutions. Montgomery and Randall [12] proposed three different anti-pheromone strategies, partially inspired by previous works [4,8]. In their first approach some amount of pheromone is removed from the solution components of the worst solution in each iteration. Their second approach makes explicit use of negative pheromone in addition to the standard pheromone. Finally, their third approach allocates a small number of ants at each iteration to explore the use of solution components with lower pheromone values, without adding dedicated anti-pheromones. Unfortunately, the experimental evaluation did not show a clear advantage of any of the three strategies over standard ACO. Simons and Smith [19] explored different extensions of [12]. They state, however, that nearly all their approaches were proved counter-intuitive by the results. The only approach that showed some usefulness was to make use of a high amount of anti-pheromone in the very early stages of the search process. In [16], Rojas-Morales et al. propose an extension of an ACO algorithm for the multidimensional knapsack problem based on opposite learning. In a first phase, the algorithm builds anti-pheromone values whose intention it is to repel the ACO algorithm during the second phase from solution components that seem locally attractive (due to a rather high heuristic value) but that lead to low-quality solutions. Unfortunately, the results do not show a consistent improvement over standard ACO. Finally, note that earlier strategies based on opposition-based learning were tested on four small TSP instances in [10].

In this paper we introduce a conceptually new way of making use of negative learning in ACO, in the context of the so-called capacitated minimum dominating set (CapMDS) problem [14]. Our results show that the performance of the standard ACO algorithm (without negative learning) is significantly improved for most of the considered problem instance types. Moreover, the current state-of-the-art algorithm is improved in the context of 10 out of 36 cases.

## 2 The CapMDS Problem

Before introducing the CapMDS problem and the developed algorithms, let us briefly recall some necessary definitions and notions from graph theory. Henceforth, $G = (V, E)$ denotes an undirected graph with a set $V = \{v_1, v_2, \cdots, v_n\}$ of $n$ vertices, and a set $E$ of edges. We assume that the given graph neither

contains self-loops nor multi-edges. Two vertices $u, v \in V$ are called neighbors—that is, they are adjacent—if and only if $(u, v) = (v, u) \in E$. Furthermore, $N(v) := \{u \in V \mid (v, u) \in E\}$ is called the *(open) neighborhood* of $v$ and denotes the set of neighbors of $v \in V$. In contrast, the *closed neighborhood* $N[v]$ of a vertex $v \in V$ is $N[v] := N(v) \cup \{v\}$. The *degree* $\deg(v)$ of $v$ is defined as the cardinality of the set of neighbors of $v$, that is, $\deg(v) = |N(v)|$. Any subset $S \subseteq V$ is called a *dominating set* of $G$ if each vertex $v \in V \setminus S$ is adjacent to at least one vertex from $S$. A vertex from $S$ is called a *dominator*. Given an undirected graph $G = (V, E)$, the classical minimum dominating set (MDS) problem asks to find a smallest-size dominating set $S \subseteq V$.

A problem instance of the CAPMDS problem is given by a tuple $(G, Cap)$ that consists of an undirected (simple) graph $G = (V, E)$ and a capacity function $Cap : V \to \mathbb{N}$. This capacity function assigns a positive integer $Cap(v) > 0$ to each vertex $v \in V$, indicating the maximum number of adjacent vertices this vertex is allowed to dominate in a valid solution.

A solution $S$ to an instance $(G, Cap)$ is a tuple $(D^S, \{C^S(v) \mid v \in D^S\})$, where $D^S \subseteq V$ is the set of selected dominators, and $\{C^S(v) \mid v \in D^S\}$ is a set that contains for each dominator $v \in D^S$ the (sub-)set $C^S(v) \subseteq N(v)$ of those of its neighbors that are (chosen to be) dominated by $v$. The following conditions have to be fulfilled in order for $S$ to be a valid solution:

1. $D^S \cup \left(\bigcup_{v \in D^S} C^S(v)\right) = V$, that is, all vertices from $V$ are either chosen to be a dominator, or are dominated by at least one dominator.
2. $|C^S(v)| \leq Cap(v)$ for all $v \in D^S$, that is, all chosen dominators dominate at most $Cap(v)$ of their neighbors.

Finally, the objective function value (to be minimized) is defined as $f(S) := |D^S|$.

*ILP Model for the CAPMDS Problem.* The following integer linear program (ILP) is reproduced from [13]. The model is presented because it plays an important role for the negative learning mechanism that is presented later. It works on the following sets of binary variables. First, a binary variable $x_v$ is associated to each vertex $v \in V$ indicating whether or not $v$ is selected as a dominator. Second, the model contains for each edge $(v, v') \in E$ two binary variables $y_{v,v'}$ and $y_{v',v}$. Variable $y_{v,v'}$ takes value one if vertex $v$ is chosen to dominate vertex $v'$; similarly for $y_{v',v}$. The CapMDS problem can then be stated as follows:

$$\textbf{minimize} \sum_{v \in V} x_v \tag{1}$$

$$\textbf{s.t.} \sum_{v' \in N(v)} y_{v',v} \geq 1 - x_v \quad \forall v \in V \tag{2}$$

$$\sum_{v' \in N(v)} y_{v,v'} \leq Cap(v) \quad \forall v \in V \tag{3}$$

$$y_{v,v'} \leq x_v \quad \forall v \in V, v' \in N(v) \tag{4}$$

$$x_v, y_{v,v'} \in \{0, 1\} \tag{5}$$

Hereby, constraint (2) ensures that all non-chosen vertices must be dominated by at least one dominator, whereas constraint (3) limits the total number of vertices dominated by a particular vertex $v$ to $Cap(v)$. Consequently, a dominator $v$ can dominate at most $Cap(v)$ vertices from its (open) neighborhood.

## 3 Proposed Approach

First of all, we present a standard ACO approach (without negative learning) for the CapMDS problem. We chose a $\mathcal{MAX}$-$\mathcal{MIN}$ Ant System (MMAS) implemented in the Hypercube Framework [1] for this purpose. In the context of this algorithm, the construction of a solution $S$ is done in a step-by-step manner. At each construction step, first, exactly one new dominator $v \in V \setminus D^S$ is chosen. In the second part of the construction step, it is decided which ones of the so-far non-dominated neighbors of $v$ will be dominated by $v$. Therefore, the pheromone model $\mathcal{T}$ used by our algorithm consists of the following values:

1. A value $\tau_v$ for each $v \in V$. These values are used to choose dominators.
2. Values $\tau_{v,v'}$ and $\tau_{v',v}$ for each edge $(v, v') \in E$. These values are used in the second part of each construction step for deciding which ones of its neighbors a newly chosen dominator will dominate.

In general terms, a MMAS algorithm (when implemented in the Hypercube Framework) works as follows (see also Algorithm 1). At each iteration, first, $n_a$ solutions are probabilistically generated both based on pheromone and on greedy information. Second, the pheromone values are modified using (at most) three solutions: (1) the iteration-best solution $S^{ib}$, (2) the restart-best solution $S^{rb}$, and (3) the best-so-far solution $S^{bs}$. The pheromone update is done with the aim to focus the search process of the MMAS algorithm on areas of the search space with high-quality solutions. Note that the algorithm also performs restarts when necessary—that is, a re-initializations of the pheromone values is performed once convergence is detected. Restarts are controlled by a convergence measure called the *convergence factor* (*cf*) and by a Boolean control variable called bs_update. The implementation of all these components for the CapMDS is detailed in the following.

InitializePheromoneValues(): In this function all pheromone values $\tau_v$ for $v \in V$ are initialized to 0.5. Moreover, all pheromone values $\tau_{v,v'}$ and $\tau_{v',v}$ for all $(v, v') \in E$ are equally initialized to 0.5.

Construct_Solution(): The construction of a solution starts with an empty solution $S = (D^S = \emptyset, \emptyset)$. Moreover, the set of non-dominated neighbors of each vertex $v \in V$, denoted by $\mathrm{ND}_v$, is initialized to $N(v)$. At each construction step, first, one vertex $v^*$ is chosen from a set $O$ (options) that includes all those vertices $v$ that still have non-dominated neighbors and that do not already form part of $D^S$:

$$O := \{v \in V \mid \mathrm{ND}_v \neq \emptyset, v \notin D^S\} \tag{6}$$

**Algorithm 1.** MMAS for the CapMDS problem

**input:** a problem instance $(G, Cap)$
$S^{bs} :=$ NULL, $S^{rb} :=$ NULL, $cf := 0$, bs_update := FALSE
InitializePheromoneValues()
**while** termination conditions not met **do**
  $\mathcal{S}^{\text{iter}} := \emptyset$
  **for** $k = 1, \ldots, n_a$ **do**
    $S^k :=$ Construct_Solution()
    $\mathcal{S}^{\text{iter}} := \mathcal{S}^{\text{iter}} \cup \{S^k\}$
  **end for**
  $S^{ib} := \text{argmin}\{f(S) \mid S \in \mathcal{S}^{\text{iter}}\}$
  **if** $f(S^{ib}) < f(S^{rb})$ **then** $S^{rb} := S^{ib}$
  **if** $f(S^{ib}) < f(S^{bs})$ **then** $S^{bs} := S^{ib}$
  ApplyPheromoneUpdate($cf$, bs_update, $S^{ib}$,$S^{rb}$,$S^{bs}$)
  $cf :=$ ComputeConvergenceFactor()
  **if** $cf > 0.9999$ **then**
    **if** bs_update = TRUE **then**
      $S^{rb} :=$ NULL, and bs_update := FALSE
      InitializePheromoneValues()
    **else**
      bs_update := TRUE
    **end if**
  **end if**
**end while**
**output:** $S^{bs}$, the best solution found by the algorithm

Note that the solution construction process stops once $O = \emptyset$. The greedy function value $\eta(v)$ of a vertex $v \in O$ is defined as $\eta(v) := \min\{Cap(v), |\text{ND}_v|\} + 1$. Based on this greedy function, the probability for a vertex $v \in O$ to be selected is determined as follows:

$$\mathbf{p}^{\text{step1}}(v) := \frac{\eta(v) \cdot \tau_v}{\sum_{v' \in O} \eta(v') \cdot \tau_{v'}} \tag{7}$$

Given the probabilities from Eq. (7), a vertex $v^* \in O$ is chosen in the following way. First a value $0 \leq r \leq 1$ is drawn uniformly at random. In case $r \leq d_{\text{rate}}$, the vertex with the highest probability is chosen deterministically. Otherwise, a vertex is chosen randomly according to the probabilities (roulette-wheel-selection). Hereby, the *determinism rate* $d_{\text{rate}} \leq 1$ is a parameter of the algorithm. Note that after choosing $v^*$, the sets of non-dominated neighbors of the neighbors of $v^*$ are updated by removing $v^*$.

In the second part of each construction step, a set of $\min\{Cap(v^*), |\text{ND}_{v^*}|\}$ non-dominated neighbors of $v^*$ is chosen and placed into $C^S(v^*)$ as follows. In case $|\text{ND}_{v^*}| \leq Cap(v^*)$, we set $C^S(v^*) := \text{ND}_{v^*}$. Otherwise, vertices are sequentially selected from $\text{ND}_{v^*}$ in the following way. First, the probability for

**Table 1.** Setting of $\kappa_{ib}$, $\kappa_{rb}$, and $\kappa_{bs}$ depending on the convergence factor $cf$ and the Boolean control variable bs_update

| | bs_update = FALSE | | | | bs_update = TRUE |
|---|---|---|---|---|---|
| | $cf < 0.4$ | $cf \in [0.4, 0.6)$ | $cf \in [0.6, 0.8)$ | $cf \geq 0.8$ | |
| $\kappa_{ib}$ | 1 | 2/3 | 1/3 | 0 | 0 |
| $\kappa_{rb}$ | 0 | 1/3 | 2/3 | 1 | 0 |
| $\kappa_{bs}$ | 0 | 0 | 0 | 0 | 1 |

each vertex $v \in \mathrm{ND}_{v^*}$ to be selected is determined as follows:

$$\mathbf{p}^{\mathrm{step2}}(v) := \frac{(|\mathrm{ND}_v| + 1) \cdot \tau_{v^*,v}}{\sum_{v' \in \mathrm{ND}_{v^*}} (|\mathrm{ND}_{v'}| + 1) \cdot \tau_{v^*,v'}} \tag{8}$$

Then, given the probabilities from Eq. (8), a vertex $\hat{v} \in \mathrm{ND}_{v^*}$ is chosen in the same way as outlined above in the context of the first part of the construction step. Vertex $\hat{v}$ is then added to an initially empty set $C^S(v^*)$, the respective ND-sets are updated, the probabilities from Eq. (8) are recalculated, and the next vertex from $\mathrm{ND}_{v^*}$ is chosen. This process stops once $\min\{Cap(v^*), |\mathrm{ND}_{v^*}|\}$ are selected. Finally, $C^S(v^*)$ is added to solution $S$, and the solution construction process proceeds with the next construction step.

ApplyPheromoneUpdate($cf$, bs_update, $S^{ib}$, $S^{rb}$, $S^{bs}$)**:** This is a standard procedure in any MMAS algorithm implemented in the Hypercube Framework. In particular, solutions $S^{ib}$, $S^{rb}$, and $S^{bs}$ are used for the pheromone update. The influence of each of these solutions on the pheromone update is determined on the basis of the convergence factor ($cf$) and the value of bs_update (see Table 1). Each pheromone value $\tau_v$ is updated as follows: $\tau_v := \tau_v + \rho \cdot (\xi_v - \tau_v)$, where $\xi_v := \kappa_{ib} \cdot \Delta(S^{ib}, v) + \kappa_{rb} \cdot \Delta(S^{rb}, v) + \kappa_{bs} \cdot \Delta(S^{bs}, v)$. Hereby, $\kappa_{ib}$ is the weight of solution $S^{ib}$, $\kappa_{rb}$ the one of solution $S^{rb}$, and $\kappa_{bs}$ the one of solution $S^{bs}$. Moreover, $\Delta(S, v)$ evaluates to 1 if and only if $v \in D^S$ (that is, $v$ is chosen as a dominator). Otherwise, the function evaluates to 0. Note also that the three weights must be chosen such that $\kappa_{ib} + \kappa_{rb} + \kappa_{bs} = 1$. Finally, note that in the case of pheromone values $\tau_{v,v'}$, the pheromone update is the same, just that functions $\Delta(S, v)$ are replaced by functions $\Delta(S, v, v')$. Hereby, function $\Delta(S, v, v')$ evaluates to 1 if and only if $v \in D^S$ and $v' \in C^S(v)$ (that is, dominator $v$ is chosen to dominate its neighbor $v'$ in solution $S$). After the pheromone update, pheromone values that exceed $\tau_{\max} = 0.99$ are set back to $\tau_{\max}$, and pheromone values that have fallen below $\tau_{\min} = 0.01$ are set back to $\tau_{\min}$. This prevents the algorithm from reaching the state of complete convergence.

ComputeConvergenceFactor($\mathcal{T}$)**:** The value of the convergence factor $cf$ is computed, in a standard way, on the basis of the pheromone values:

$$cf := 2 \left( \left( \frac{\sum_{\tau \in \mathcal{T}} \max\{\tau_{\max} - \tau, \tau - \tau_{\min}\}}{|\mathcal{T}| \cdot (\tau_{\max} - \tau_{\min})} \right) - 0.5 \right) \tag{9}$$

Hereby, $\mathcal{T}$ stands for the set of all $\tau_v$-values and all $\tau_{v,v'}$-values. With this formula, the value of $cf$ results in zero, when all pheromone values are set to 0.5. In contrast, when all pheromone values have either value $\tau_{\min}$ or $\tau_{\max}$, the value $cf$ evaluates to one. In all other cases, $cf$ has a value between 0 and 1. This completes the description of all components of the proposed algorithm.

### 3.1 Adding Negative Learning

First of all, for all pheromone values $\tau_v$ ($v \in V$) we introduce a negative pheromone value $\tau_v^{\text{neg}}$. Moreover, for all pheromone values $\tau_{v,v'}$ we also introduce the negative version $\tau_{v,v'}^{\text{neg}}$. In contrast to the standard pheromone values, these negative pheromone values are initialized to $\tau_{\min}$ at the start of the algorithm, and whenever the algorithm is restarted (which still depends exclusively on the standard pheromone values).

The negative pheromone values are used in the following way to change the probabilities in both phases of each step for the construction of a solution $S$. Remember that the first phase concerns the choice of the next dominator $v^*$, and the second phase concerns the choice of a set $C^S(v^*)$ of so-far uncovered neighbors of $v^*$ that $v^*$ will dominate. The updated formula for calculating the probabilities in the first phase is as follows (compare to Eq. 7):

$$\mathbf{p}^{\text{step1}}(v) := \frac{\eta(v) \cdot \tau_v \cdot (1 - \tau_v^{\text{neg}})}{\sum_{v' \in O} \eta(v') \cdot \tau_{v'} \cdot (1 - \tau_{v'}^{\text{neg}})} \tag{10}$$

In the second phase of each construction step $C^S(v^*)$ is sequentially filled with vertices taken from $\text{ND}_{v^*}$ (the set of currently uncovered neighbors of $v^*$) in the following way. First, the probability for each vertex $v \in \text{ND}_{v^*}$ to be selected is determined as follows (compare to Eq. 8):

$$\mathbf{p}^{\text{step2}}(v) := \frac{(|\text{ND}_v| + 1) \cdot \tau_{v^*,v} \cdot (1 - \tau_{v^*,v}^{\text{neg}})}{\sum_{v' \in \text{ND}_{v^*}} (|\text{ND}_{v'}| + 1) \cdot \tau_{v^*,v'} \cdot (1 - \tau_{v^*,v'}^{\text{neg}})} \tag{11}$$

Another change in comparison to the standard way of generating solutions is that, during this second phase, only vertices whose probability $\mathbf{p}^{\text{step2}}(v)$ is greater or equal to 0.001 can be selected. This makes it possible to generate solutions in which a vertex selected as a dominator might not be chosen to dominate as many of its uncovered neighbors as possible in that moment.

As mentioned before, we strongly believe that the information that is used to determine the negative pheromone values should originate from an algorithmic component different to the ACO algorithm itself. In the context of the CapMDS

problem we therefore propose the following. At each iteration of our MMAS algorithm, the set of solutions generated at the incumbent iteration ($\mathcal{S}^{\text{iter}}$) is used for generating a subinstance of the tackled problem instance. Such a subinstance $I^{\text{sub}}$ is a tuple $(D^{\text{sub}}, \{C^{\text{sub}}(v) \mid v \in D^{\text{sub}}\})$ where

- $D^{\text{sub}} := \bigcup_{S \in \mathcal{S}^{\text{iter}}} D^S$
- $C^{\text{sub}}(v) := \bigcup_{\substack{S \in \mathcal{S}^{\text{iter}} \\ \text{s.t. } v \in D^S}} C^S(v)$

After generating the $n_a$ solutions per iteration, the ILP solver CPLEX is used (with a time limit of $t_{\text{ILP}}$ CPU seconds) to solve the corresponding subinstance (if possible) to optimality. Otherwise, the best solution found within the allotted computation time is returned. In any case, the returned solution is denoted by $S^{\text{ILP}}$. In order to solve the subinstance, the ILP model from Sect. 2 is used with the following additional restrictions. All variables $x_v$ such that $v \notin D^{\text{sub}}$ are set to zero. Moreover, all variables $x_{v,v'}$ such that either $v \notin D$ or $v' \notin C^{\text{sub}}(v)$ are set to zero too.

After obtaining solution $S^{\text{ILP}}$ both the standard pheromone value update and the update of the negative pheromone values is performed. The update of the negative pheromone values is done with the same formula as in the case of the standard pheromone update (see the description of function ApplyPheromoneUpdate(*cf*, bs_update, $S^{ib}$, $S^{rb}$, $S^{bs}$)). Only the learning rate $\rho$ is replaced by a *negative learning rate* $\rho^{\text{neg}}$, and the definition of the $\xi_v$ (respectively $\xi_{v,v'}$) values changes. In particular, $\xi_v$ is set to 1 for all $v \in D^{\text{sub}}$ with $v \notin D^{S^{\text{ILP}}}$. In all other cases $\xi_v$ is set to 0. Moreover, $\xi_{v,v'}$ is set to 1, for all $v' \in C^{\text{sub}}(v)$ with $v' \notin C^{S^{\text{ILP}}}(v)$. In all other cases $\xi_{v,v'}$ is set to 0. In other words, those *solution components* that form part of the subinstance (and, therefore, form part of at least one of the solutions generated by MMAS) but that do not form part of the (possibly optimal) solution $S^{\text{ILP}}$ to the subinstance, are penalized.

Note that—in contrast to the standard MMAS algorithm, which is simply denoted by ACO in the following section—the algorithm making use of negative learning is henceforth denoted by $\text{ACO}_{\text{neg}}$. Finally, not taking profit from solution $S^{\text{ILP}}$ in an additional, more direct, way may result in wasting valuable information. Therefore, we also test an extended version of $\text{ACO}_{\text{neg}}$, henceforth denoted by $\text{ACO}^{+}_{\text{neg}}$, that updates solutions $S^{rb}$ and $S^{bs}$ at each iteration with solution $S^{\text{ILP}}$ if appropriate.

## 4 Experimental Evaluation

All experiments concerning ACO, $\text{ACO}_{\text{neg}}$ and $\text{ACO}^{+}_{\text{neg}}$ were performed on a cluster of machines with Intel® Xeon® CPU 5670 CPUs with 12 cores of 2.933 GHz and a minimum of 32 GB RAM. Moreover, for solving the subinstances in $\text{ACO}_{\text{neg}}$ and $\text{ACO}^{+}_{\text{neg}}$ we used CPLEX 12.8 in one-threaded mode.

The proposed algorithms were evaluated on the largest ones of the general graphs benchmark set for the CapMDS problem from [14]. These graphs are

**Table 2.** Outcome of parameter tuning.

| Algorithm | $n_a$ | $d_{\text{rate}}$ | $\rho$ | $\rho^{\text{neg}}$ | $t_{\text{ILP}}$ |
|---|---|---|---|---|---|
| ACO | 5 | 0.9 | 0.1 | n.a | n.a. |
| $\text{ACO}_{\text{neg}}$ | 20 | 0.7 | 0.1 | 0.3 | 10.0 |
| $\text{ACO}^{+}_{\text{neg}}$ | 20 | 0.6 | 0.1 | 0.2 | 5.0 |

characterized by a number of vertices ($n$), a number of edges ($m$), a vertex capacity type (uniform vs. variable), and a capacity. In the case of uniform capacities, graphs with three different capacities (2, 5 and $\alpha$) exist. Hereby, $\alpha$ refers to the average degree of the corresponding graph. In the case of variable capacities, the vertex capacities are—for each vertex—randomly chosen from the following three intervals: $(2, 5)$, $(\alpha/5, \alpha/2)$ and $[1, \alpha]$. For each combination of these graph characteristics, the benchmark set consists of 10 randomly generated graphs.

*Algorithm Tuning.* All three algorithm variants require parameter values to be set to well-working options. In particular, all three algorithm versions need parameter values for $n_a$ (the number of solutions per iteration), $d_{\text{rate}}$ (the determinism rate for solution construction), and $\rho$ (the learning rate). Additionally, $\text{ACO}_{\text{neg}}$ and $\text{ACO}^{+}_{\text{neg}}$ require values for parameters $\rho^{\text{neg}}$ (the negative learning rate) and $t_{\text{ILP}}$ (the time limit, in CPU seconds, for CPLEX at each iteration). For the purpose of parameter tuning we made use of irace [9], which is a scientific tool for parameter tuning. This tool was used for generating one single parameter setting for each algorithm. As tuning instances we chose the first (out of 10) instances for each combination of the four input graph characteristics. Moreover, a budget of 2000 applications was given to irace. The parameter value domains were fixed as follows: $n_a \in \{3, 5, 10, 20\}$, $d_{\text{rate}} \in \{0.1, 0.2, \ldots, 0.8, 0.9\}$, $\rho, \rho^{\text{neg}} \in \{0.1, 0.2, 0.3\}$, and $t_{\text{ILP}} \in \{2.0, 3.0, 5.0, 10.0\}$ (in seconds). The parameter value settings determined by irace are shown in Table 2.

*Numerical Results.* Each algorithm was applied exactly once (with a time limit of 1000 CPU seconds) to each problem instance. The results, averaged over 10 instances per table row, are shown in Table 3 (uniform capacity graphs) and in Table 4 (variable capacity graphs). While the two tables separate the instances with respect to the vertex capacity type (uniform vs. variable), the first three columns of each table provide information about the remaining three input graph characteristics ($n$, $m$, and vertex capacity). The fourth table column provides information about the best result known from the literature, while the fifth and sixth table columns present the results of CMSA, which is the current state-of-the-art algorithm from [13]. Both the results of CMSA and of the three ACO versions are shown by means of the average solution quality and the average computation time needed for producing these results.

**Table 3.** Results for general graphs with uniform capacity.

| $n$ | $m$ | Cap. | Best known | CMSA | | ACO | | $\text{ACO}_{\text{neg}}$ | | $\text{ACO}^{+}_{\text{neg}}$ | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | Avg. | Time | Avg. | Time | Avg. | Time | Avg. | Time |
| 800 | 1000 | 2 | 267.0 | **267.0** | 3.6 | 285.3 | 136.2 | **267.0** | 8.4 | **267.0** | 2.5 |
| 800 | 2000 | 2 | 267.0 | **267.0** | 3.9 | 269.4 | 80.3 | 269.3 | 129.3 | **267.0** | 67.8 |
| 800 | 5000 | 2 | 267.0 | **267.0** | 3.2 | **267.0** | 59.0 | 271.1 | 192.1 | **267.0** | 119.4 |
| 1000 | 1000 | 2 | 334.0 | **334.0** | 7.9 | 364.0 | 157.1 | **334.0** | 7.2 | **334.0** | 0.6 |
| 1000 | 5000 | 2 | 334.0 | **334.0** | 6.5 | 334.2 | 88.0 | 384.6 | 50.3 | **334.0** | 126.9 |
| 1000 | 10000 | 2 | 334.0 | **334.0** | 5.8 | **334.0** | 32.4 | 379.5 | 176.7 | 337.2 | 136.7 |
| 800 | 1000 | 5 | 242.5 | 243.1 | 205.6 | 262.8 | 113.7 | 245.5 | 89.2 | 244.4 | 76.1 |
| 800 | 2000 | 5 | 162.8 | 162.8 | 574.7 | 177.0 | 116.1 | 163.2 | 61.0 | **161.9**$^{*}$ | 79.1 |
| 800 | 5000 | 5 | 134.0 | **134.0** | 4.7 | 135.3 | 72.4 | 158.7 | 6.3 | **134.0** | 160.2 |
| 1000 | 1000 | 5 | 333.7 | **333.7** | 10.5 | 362.8 | 141.2 | **333.7** | 8.8 | **333.7** | 0.6 |
| 1000 | 5000 | 5 | 167.0 | **167.0** | 40.8 | 172.2 | 101.1 | 206.3 | 61.4 | **167.0** | 173.6 |
| 1000 | 10000 | 5 | 167.0 | **167.0** | 3.7 | 167.8 | 67.3 | 188.4 | 8.6 | **167.0** | 102.7 |
| 800 | 1000 | $\alpha$ | 267.0 | **267.0** | 4.6 | 284.0 | 153.8 | **267.0** | 10.1 | **267.0** | 2.8 |
| 800 | 2000 | $\alpha$ | 162.8 | 162.8 | 537.3 | 178.8 | 93.0 | 163.4 | 73.8 | **162.0**$^{*}$ | 69.7 |
| 800 | 5000 | $\alpha$ | 91.1 | 93.0 | 717.9 | 92.9 | 62.8 | 90.9 | 74.0 | **89.2**$^{*}$ | 104.3 |
| 1000 | 1000 | $\alpha$ | 334.0 | **334.0** | 13.7 | 365.1 | 175.2 | **334.0** | 6.9 | **334.0** | 0.6 |
| 1000 | 5000 | $\alpha$ | 132.5 | 135.0 | 782.9 | 137.3 | 82.0 | 131.6 | 65.4 | **127.3**$^{*}$ | 116.3 |
| 1000 | 10000 | $\alpha$ | 81.3 | 86.8 | 518.7 | 82.6 | 67.9 | 87.9 | 98.4 | **80.7**$^{*}$ | 133.1 |

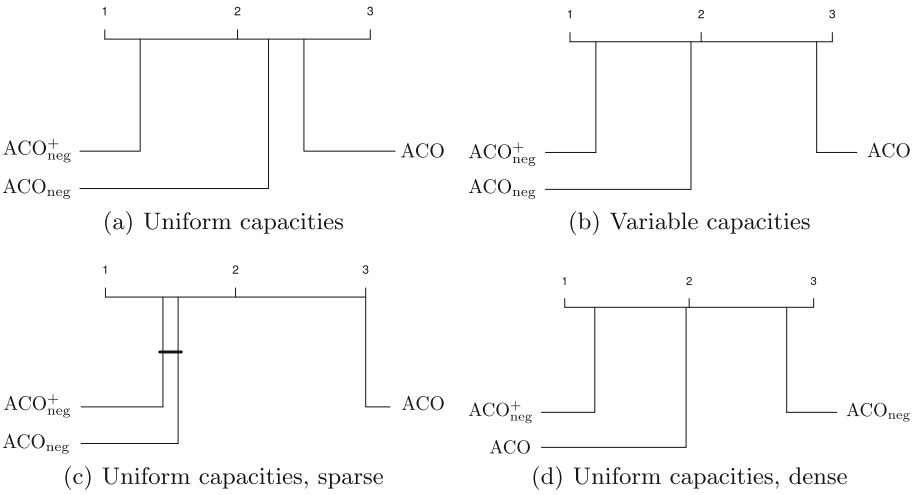
In order to facilitate an interpretation of these results we provide the corresponding *critical difference* (CD) plots [3]. First, the Friedman test was used to compare the three approaches simultaneously. As a consequence of the rejection of the hypothesis that the techniques perform equally, the corresponding pairwise comparisons were performed using the Nemenyi post-hoc test [7]. The obtained results are graphically shown by means of the above-mentioned CD plots in Fig. 1. In these plots, each considered algorithm variant is placed on the horizontal axis according to its average ranking for the considered subset of problem instances. The performances of those algorithm variants that are below the critical difference threshold (computed with a significance level of 0.05) are considered as statistically equivalent; see the horizontal bars joining the markers of the respective algorithm variants.

The graphic in Fig. 1(a) shows the CD plot for the uniform capacity instances, and the one in Fig. 1(b) for the variable capacity instances. In both graphics it can be seen that both algorithm variants with negative learning ($\text{ACO}_{\text{neg}}$ and $\text{ACO}^{+}_{\text{neg}}$) significantly improve over the standard ACO approach. Moreover, $\text{ACO}^{+}_{\text{neg}}$ improves over $\text{ACO}_{\text{neg}}$ with statistical significance. This is also the general picture given by the numerical results in Tables 3 and 4.

Interestingly, when separating the instances according to different graph densities, it can be noticed that negative learning is especially useful in the context of sparse graphs. In contrast, when moving towards dense graphs the efficacy of negative learning is reduced. In the context of graphs with uniform capacities,

**Table 4.** Results for general graphs with variable capacity.

| $n$ | $m$ | Cap. | Best known | CMSA | | ACO | | $\text{ACO}_{\text{neg}}$ | | $\text{ACO}^{+}_{\text{neg}}$ | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | Avg. | Time | Avg. | Time | Avg. | Time | Avg. | Time |
| 800 | 1000 | $(2, 5)$ | 248.1 | 248.2 | 79.2 | 269.2 | 131.7 | 251.8 | 47.4 | 249.9 | 68.6 |
| 800 | 2000 | $(2, 5)$ | 181.2 | 181.5 | 341.7 | 195.0 | 98.7 | 180.8 | 73.1 | **179.8**$^*$ | 79.7 |
| 800 | 5000 | $(2, 5)$ | 134.1 | **134.1** | 28.1 | 139.1 | 99.6 | 138.4 | 127.3 | **134.1** | 94.3 |
| 1000 | 1000 | $(2, 5)$ | 333.8 | **333.8** | 3.9 | 365.6 | 146.9 | **333.8** | 8.8 | **333.8** | 1.9 |
| 1000 | 5000 | $(2, 5)$ | 169.0 | **169.0** | 85.1 | 182.8 | 86.3 | 171.2 | 109.7 | 169.6 | 105.0 |
| 1000 | 10000 | $(2, 5)$ | 167.0 | **167.0** | 27.5 | 168.4 | 92.3 | 198.3 | 7.7 | **167.0** | 170.1 |
| 800 | 1000 | $(\alpha/5, \alpha/2)$ | 400.0 | **400.0** | 2.6 | 409.3 | 112.5 | 400.2 | 66.7 | **400.0** | 0.8 |
| 800 | 2000 | $(\alpha/5, \alpha/2)$ | 273.4 | **273.4** | 6.5 | 283.2 | 87.5 | 274.6 | 101.6 | **273.4** | 7.6 |
| 800 | 5000 | $(\alpha/5, \alpha/2)$ | 115.0 | 115.1 | 178.6 | 123.0 | 83.7 | 116.5 | 77.1 | **115.0** | 85.7 |
| 1000 | 1000 | $(\alpha/5, \alpha/2)$ | 500.0 | **500.0** | 8.6 | 517.7 | 122.2 | **500.0** | 11.2 | **500.0** | 1.0 |
| 1000 | 5000 | $(\alpha/5, \alpha/2)$ | 168.1 | **168.1** | 77.4 | 181.3 | 128.7 | 170.9 | 105.4 | 168.8 | 92.2 |
| 1000 | 10000 | $(\alpha/5, \alpha/2)$ | 104.7 | 107.1 | 247.9 | 104.8 | 97.1 | 108.6 | 131.0 | **95.6**$^*$ | 121.3 |
| 800 | 1000 | $[1, \alpha]$ | 300.2 | **300.2** | 4.0 | 316.0 | 144.9 | **300.2** | 6.8 | **300.2** | 0.5 |
| 800 | 2000 | $[1, \alpha]$ | 186.2 | 186.2 | 442.6 | 204.9 | 105.3 | 187.3 | 61.5 | **185.8**$^*$ | 63.9 |
| 800 | 5000 | $[1, \alpha]$ | 98.1 | 98.1 | 683.7 | 101.7 | 63.3 | 96.8 | 84.4 | **95.6**$^*$ | 80.2 |
| 1000 | 1000 | $[1, \alpha]$ | 400.8 | **400.8** | 6.4 | 409.6 | 141.5 | **400.8** | 7.3 | **400.8** | 0.5 |
| 1000 | 5000 | $[1, \alpha]$ | 143.8 | 143.8 | 866.9 | 151.9 | 95.4 | 141.4 | 101.1 | **140.6**$^*$ | 98.9 |
| 1000 | 10000 | $[1, \alpha]$ | 90.1 | 90.1 | 541.8 | 88.2 | 66.8 | 87.8 | 132.1 | **85.6**$^*$ | 108.0 |




(a) Uniform capacities (b) Variable capacities

(c) Uniform capacities, sparse (d) Uniform capacities, dense

**Fig. 1.** Critical difference plots

it is even the case that standard ACO outperforms $ACO_{neg}$ for dense graphs. This is shown in the context of uniform capacity graphs in Figs. 1(c) and (d).

## 5 Conclusions and Outlook

In this paper we introduced a new approach for making use of negative learning in ant colony optimization. This approach builds, at each iteration, a subinstance of the original problem instance by merging the solution components found in the solutions generated by the ant colony optimization algorithm in that iteration. Then it uses a different optimization technique—CPLEX was used here—for finding the best solution in this subinstance. The solution components from the subinstance that do not form part of this solution are penalized by means of increasing their negative pheromone values. The proposed approach is shown to be very beneficial for the capacitated minimum dominating set problem.

Future work will center along two lines. First, we plan to study why this new approach is more useful in sparse graphs. And second, we plan to apply this approach to a whole range of different combinatorial optimization problems.

**Acknowledgements.** This work was supported by project CI-SUSTAIN funded by the Spanish Ministry of Science and Innovation (PID2019-104156GB-I00).

## References

1. Blum, C., Dorigo, M.: The hyper-cube framework for ant colony optimization. IEEE Trans. Syst. Man Cybern. Part B **34**(2), 1161–1172 (2004)
2. Blum, C., Roli, A.: Metaheuristics in combinatorial optimization: overview and conceptual comparison. ACM Comput. Surv. **35**(3), 268–308 (2003)
3. Calvo, B., Santafé, G.: scmamp: statistical comparison of multiple algorithms in multiple problems. R J. **8**(1) (2016)
4. Cordón, O., Fernández de Viana, I., Herrera, F., Moreno, L.: A new ACO model integrating evolutionary computation concepts: the best-worst ant system. In: Proceedings of ANTS 2000 - Second International Workshop on Ant Algorithms, pp. 22–29 (2000)
5. Dorigo, M., Stützle, T.: Ant Colony Optimization. MIT Press, Cambridge (2004)
6. Dorigo, M., Stützle, T.: Ant colony optimization: overview and recent advances. In: Gendreau, M., Potvin, J.Y. (eds.) Handbook of Metaheuristics. International Series in Operations Research & Management ScienceInternational Series in Operations Research & Management Science, vol. 272, pp. 311–351. Springer, Cham (2019). https://doi.org/10.1007/978-3-319-91086-4_10
7. García, S., Herrera, F.: An extension on "statistical comparisons of classifiers over multiple data sets" for all pairwise comparisons. J. Mach. Learn. Res. **9**, 2677–2694 (2008)
8. Iredi, S., Merkle, D., Middendorf, M.: Bi-criterion optimization with multi colony ant algorithms. In: Zitzler, E., Thiele, L., Deb, K., Coello Coello, C.A., Corne, D. (eds.) EMO 2001. LNCS, vol. 1993, pp. 359–372. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-44719-9_25

9. López-Ibánez, M., Dubois-Lacoste, J., Cáceres, L.P., Birattari, M., Stützle, T.: The irace package: iterated racing for automatic algorithm configuration. Oper. Res. Perspect. **3**, 43–58 (2016)
10. Malisia, A.R., Tizhoosh, H.R.: Applying opposition-based ideas to the ant colony system. In: 2007 IEEE Swarm Intelligence Symposium, pp. 182–189. IEEE press (2007)
11. Maniezzo, V.: Exact and approximate nondeterministic tree-search procedures for the quadratic assignment problem. INFORMS J. Comput. **11**(4), 358–369 (1999)
12. Montgomery, J., Randall, M.: Anti-pheromone as a tool for better exploration of search space. In: Dorigo, M., Di Caro, G., Sampels, M. (eds.) ANTS 2002. LNCS, vol. 2463, pp. 100–110. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-45724-0_9
13. Pinacho-Davidson, P., Bouamama, S., Blum, C.: Application of CMSA to the minimum capacitated dominating set problem. In: Proceedings of GECCO 2019 - The Genetic and Evolutionary Computation Conference, pp. 321–328. ACM, New York (2019)
14. Potluri, A., Singh, A.: Metaheuristic algorithms for computing capacitated dominating set with uniform and variable capacities. Swarm Evol. Comput. **13**, 22–33 (2013)
15. Robinson, E.J.H., Jackson, D.E., Holcombe, M., Ratnieks, F.L.W.: 'No entry' signal in ant foraging. Nature **438**(7067), 442 (2005)
16. Rojas-Morales, N., Riff, M.-C., Coello Coello, C.A., Montero, E.: A cooperative opposite-inspired learning strategy for ant-based algorithms. In: Dorigo, M., Birattari, M., Blum, C., Christensen, A.L., Reina, A., Trianni, V. (eds.) ANTS 2018. LNCS, vol. 11172, pp. 317–324. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-00533-7_25
17. Schlein, Y., Galun, R., Ben-Eliahu, M.N.: Abstinons - male-produced deterrents of mating in flies. J. Chem. Ecol. **7**(2), 285–290 (1981)
18. Schoonderwoerd, R., Holland, O., Bruten, J., Rothkrantz, L.: Ant-based load balancing in telecommunications networks. Adapt. Behav. **5**(2), 169–207 (1997)
19. Simons, C., Smith, J.: Exploiting antipheromone in ant colony optimisation for interactive search-based software design and refactoring. In: Proceedings of GECCO 2016 - Genetic and Evolutionary Computation Conference Companion, pp. 143–144. ACM (2016)