Negative Learning Ant Colony Optimization for Network Alignment

Guillem Rodríguez Corominas ∗1,2, Christian Blum †1, and Maria J. Blesa ‡2

1Artificial Intelligence Research Institute (IIIA-CSIC)
2Universitat Politècnica de Catalunya (UPC - BarcelonaTech)

Presented at GECCO 2022
The Genetic and Evolutionary Computation Conference
Boston, USA
July 9-13, 2022

Abstract

The Network Alignment problem is a hard Combinatorial Optimization problem with a wide range of applications, especially in computational biology. Given two (or more) networks, the goal is to find a mapping between their respective nodes that preserves the topological and functional structure of the networks. In this work we extend a novel ant colony optimization approach for network alignment by adding a recently proposed Negative Learning mechanism. In particular, information for Negative Learning is obtained by solving sub-instances of the tackled problem instances at each iteration by means of an Integer Linear Programming solver. The results show that the proposed algorithm not only outperforms the standard ant colony optimization approach but also current state-of-the-art methods from the literature.

∗ORCID: 0000-0002-3863-2017; EMAIL: guillem.rodriguez.corominas@upc.edu
G. Rodríguez Corominas acknowledges support by MCIN/AEI/10.13039/501100011033 under grant PID2019-104156GB-I00 (CI-SUSTAIN).

†ORCID: 0000-0002-1736-3559; EMAIL: christian.blum@csic.es
C. Blum also acknowledges support by MCIN/AEI/10.13039/501100011033 under grant PID2019-104156GB-I00 (CI-SUSTAIN).

‡ORCID: 0000-0001-8246-9926; EMAIL: maria.j.blesa@upc.edu
M. Blesa acknowledges support by AGAUR (GENCAT) under grant 2017-SGR-786 (ALBCOM), and by MCIN/AEI/10.13039/501100011033 under grant PID2020-112581GB-C21 (MOTION).

The final publication is available at ACM via http://dx.doi.org/10.1145/3512290.3528795
1 Introduction

The Network Alignment (NA) problem asks to find a mapping between the nodes of two (or more) networks optimizing a particular, application-dependent, quality measure. Applications which can be modeled in terms of finding alignments between networks are abundant in real-life settings. Generally, the objective is to transfer knowledge from one network to the other. One of the main application areas is Biology, where, for instance, network alignment can be used to transfer knowledge from well-studied organisms to lesser-known ones. Other applications include identifying orthologous relationships between different proteins or phylogenetic relationships between species [14]. Another wide area of study are social networks, where network alignment plays a crucial role in various fields and applications related to this domain [25]. Given the fact that users tend to behave similarly in the different social networks they are registered to, one can transmit user behavior from one social network to another. Applications include link prediction [2, 6], which is based on the fact that if two users are friends in a social network, it is conceivable that they are also friends in the other; and cross-network recommendation [11], where holistic user profiles are generated based on their behavior in a source network in order to suggest interests to target network users.

As the NA problem is applied in different research areas, there exist a large number of different algorithms for solving it. This makes it difficult to identify a single algorithm as the current state of the art. The most common methods from the literature are heuristics that greedily align the nodes by using some form of node similarity extracted from the input networks. Well-known examples include Netal [18] and HubAlign [8]. Metaheuristics are also widely used. They usually aim to maximize a given quality measure, but sometimes they ignore the underlying topological information that might be useful when constructing solutions. Some methods in this category are Magna [23], Acogna2 [24] and Sana [17]. Other types of methods include representation learning methods such as Regal [10], mine-and-merge methods such as PathBLAST [12], and exact techniques such as Natalie2 [7]. A last family of algorithms worth mentioning is the GRAAL family, commonly used in the context of Protein-Protein Interaction (PPI) networks. They usually make use of the graphlet signature similarity in order to build an alignment. The latest example is L-Graal [16].

With the aim of obtaining an algorithm showing a high performance in different fields of application, a new Ant Colony Optimization (ACO) approach, AntNetAlign, was introduced in [22], which makes use of similarity information to bias the construction process. AntNetAlign, as any other standard ACO algorithm, is exclusively based on learning from positive examples. This is achieved by means of a pheromone model that learns to favor solution components that appear in good solutions. Nonetheless, in certain contexts, nature has shown that the additional use of negative learning can be beneficial. A recent extension of ACO algorithms with a negative learning mechanism was introduced in [19]. This extension makes use of additional optimization methods for identifying components of solutions that should receive a negative feedback, in contrast to a positive one.

In this work, we aim at improving the performance of AntNetAlign, which already performs at state-of-the-art level in comparison to competing approaches, with a negative learning extension along the lines of [19]. In the remainder of this section, we introduce some basic concepts that will help to define the tackled problem. In Section 2 we first provide a rough description of AntNetAlign, before we describe in detail the extension of the algorithm with negative learning. In particular, we give a detailed explanation of how negative learning is incorporated into the algorithm. In Section 3 the experimental results are presented. Finally, in Section 4 we conclude our work and present some ideas for future work.

1.1 Terms and concepts

An undirected graph (or network) $G = (V, E)$ is a mathematical structure amounting to a non-empty set $V$ of objects (called vertices or nodes) connected by undirected edges (those in $E$), which represent pairwise symmetric relations between these objects. An edge $e \in E$ is denoted by its two
endpoints \( u, v \in V \), i.e., \( e = (u, v) \). Moreover, an edge \( e = (u, v) \) is said to be incident to \( u \) and \( v \). Given a graph \( G \), we denote the set of vertices as \( V(G) \), and the set of edges as \( E(G) \). The order of a graph \( G \) is equal to the number of vertices (i.e., \(|V|\)), while the size of a graph refers to the total number of edges (i.e., \(|E|\)). Moreover, given \( V' \subseteq V \), let \( G[V'] \) be the induced subgraph of \( G \) with node set \( V' \), i.e., the graph \( G[V'] = (V', E') \) where \( E' = \{(u, v) \mid u, v \in V', (u, v) \in E\} \). Assuming that the networks do not have loops (i.e., there are no edges of the form \((u, u)\)), the (open) neighborhood of a vertex \( v \) is the set of its neighboring nodes, i.e., \( N(v) := \{u \in V \mid (v, u) \in E\} \). Similarly, the closed neighborhood of \( v \) is denoted as \( N[v] := N(v) \cup \{v\} \). As usual, the degree of a vertex is the number of edges incident to it (or number of neighbors), i.e., \( deg(v) = |N(v)| \). Lastly, the \( D \)-restricted neighborhood of a vertex \( v \) is described as \( N_D(v) := N(v) \cap D \), i.e., the set of neighbors of \( v \) that are in \( D \).

An alignment between two graphs \( G_1 = (V_1, E_1) \) and \( G_2 = (V_2, E_2) \), where \(|V_1| \leq |V_2|\), is an injective function \( f : V_1 \mapsto V_2 \), i.e., a mapping between their respective nodes. \( G_1 \) and \( G_2 \) will be henceforth referred to as the source and target networks, respectively. Note that there is no loss of generality by assuming that one network has smaller order than the other. Given an alignment \( f \) between two networks \( G_1 \) and \( G_2 \), nodes \( v \in V_1 \) and \( \pi \in V_2 \) are said to be aligned if and only if \( f(v) = \pi \). Moreover, an edge \((v, w) \in E_2 \) is said to be induced by \( f \) if and only if \( \exists u_1, u_2 \in V_1 \) such that \( f(u_1) = v \) and \( f(u_2) = w \), i.e., there exist two (not necessarily connected) nodes from the source network that are mapped to the endpoints of the induced edge. By an abuse of notation, we can express the set of induced edges from \( G_2 \) given a (partial) mapping over the subset of nodes \( V \subseteq V_1 \) as \( E(G_2[f(V)]) \). Although induced edges are (usually) only considered in the target network, given that we will work with partially-defined mappings, we extend the previous definition to edges from the source network. Note that this is not necessary when only complete solutions are considered as, given a complete mapping, all the edges on the source network are induced. Thus, we say that an edge is induced if and only if both of its endpoints are aligned with a node from the other network. Finally, an edge \((u, v) \in E_1\) is said to be conserved if and only if \((f(u), f(v)) \in E_2\).

### 1.2 Problem formulation

The Network Alignment (NA) problem asks to find an alignment between two networks that preserves the topological and functional structure. In order to evaluate the quality of an alignment, given the wide range of applications, many measures are available. A few of them are limited to functional aspects, while others focus purely on the topological or structural view. Functional scoring methods usually take into account the context of application. For instance, the significance of the biological function can be considered when aligning Protein-Protein Interaction (PPI) networks. Nonetheless, in other research areas, functional information only considers the role of the aligned nodes inside the respective networks. On the other hand, structural quality measures are mainly focused on the inherent topological relation between the aligned networks. In this work, we only consider structural quality measures. Given two networks \( G_1 = (V_1, E_1) \) and \( G_2 = (V_2, E_2) \) and an alignment \( f \) between them, some of the most common structural quality measures are the following:

- **Edge Correctness (EC)** \([14]\): Portion of edges from the source network that are mapped to edges from the target network, i.e., the percentage of conserved edges within the source network:

\[
EC = \frac{|f(E_1) \cap E_2|}{|E_1|} \times 100
\]

where \( f(E_1) = \{(f(u), f(v)) : (u, v) \in E_1\} \). As the EC score only considers the source network, it is not able to penalize alignments that map sparse regions of the source network to denser regions of the target network.

- **Induced Conserved Structure (ICS)** \([20]\): Percentage of edges conserved by the alignment with respect to the number of edges in the induced subnetwork of \( G_2 \) formed by the aligned
nodes from $G_1$:

$$\text{ICS} = \frac{|f(E_1) \cap E_2|}{|E(G_2[f(V_1)])|} \times 100$$

Contrarily to EC, since the ICS score is only defined with respect to the target network, this measure fails to penalize alignments mapping denser regions to sparser ones.

- **Symmetric Substructure Score ($S^3$) [23]**: In order to overcome the drawbacks of the previous scores, in this case, the source network $G_1$ and the induced subnetwork of the target graph under the alignment ($G_2[f(V_1)]$) are overlapped into a composite graph. Then, the $S^3$ score measures the ratio of conserved edges with respect to the total number of unique edges in this composite graph:

$$S^3 = \frac{|f(E_1) \cap E_2|}{|E_1| + |E(G_2[f(V_1)])| - |f(E_1) \cap E_2|} \times 100$$

Therefore, the $S^3$ score is able to penalize both alignments that map regions of dissimilar density (i.e., sparse to dense and vice versa).

The objective of the NA problem, then, is to find an optimal alignment with respect to a given quality measure. It is easy to see that the NA problem is $NP$-hard by reducing the well-known $NP$-complete subgraph isomorphism problem to the decision version of the NA problem [13, 9, 3].

### 2 The algorithmic approach

As mentioned before, in this paper we extend the AntNetAlign approach from [22] with negative learning, with the aim of improving the algorithms’ already excellent performance.

The algorithmic framework of AntNetAlign is the one of a standard MAX-MIN Ant System (MMAS) in the hypercube framework [4]. At each iteration of the algorithm, $n_{ants}$ solutions are probabilistically constructed based on greedy and pheromone information. Afterwards, the pheromones values are updated based on (maximally) three solutions: the iteration-best solution, the restart-best solution and the best-so-far solution. Upon reaching convergence, the algorithm is restarted by re-initializing the pheromone values. The algorithm terminates when it reaches a fixed number of constructed solutions. Due to space restrictions we cannot explain the problem-independent parts of the algorithm in more detail. They can, however, be consulted in [21] [22].

In the following we will focus on the description of the problem-dependent part of the algorithm, which is the construction of solutions. Afterwards, we will show how we extended the algorithm with negative learning. However, first of all, we will describe the solution representation and the pheromone model.

Given the two input networks $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$, a solution to the problem is represented by a vector $\pi$ of size $|V_1|$, where $\pi_i = j$ if and only if node $v_i \in V_1$ is aligned to node $v_j \in V_2$. AntNetAlign makes use of a pheromone model $T$ which contains a pheromone value $\tau_{ij}$ for each of the $|V_1| \times |V_2|$ possible alignments of a source node $v_i \in V_1$ to a target node $v_j \in V_2$, where $0 < \tau_{\text{min}} \leq \tau_{ij} \leq \tau_{\text{max}} < 1$. Note that, as in most applications of MMAS algorithms in the hypercube framework, $\tau_{\text{min}}$ is set to 0.001 and $\tau_{\text{max}}$ to 0.999. The evaluation function $g : \pi \mapsto \mathbb{R}^+$ assigns a score to each solution using any of the structural measures introduced in Section 1. In the case of partial solutions, only the subset of nodes aligned so far are considered for this purpose.

#### 2.1 Solution construction in AntNetAlign

In the following, we describe the construction of solutions in AntNetAlign, without the use of negative learning. The pseudo code of this solution construction mechanism is presented in Algorithm 1. A solution is constructed by mapping each node from the source network to a node from the target network. To this end, the function receives as input the two networks
Algorithm 1 Solution construction procedure

1: input: networks $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$, pheromone model $T$, topological similarity matrix $S$
2: $U := V_1$ #Set of unaligned (source) nodes
3: $C := V_2$ #Set of candidate (target) nodes
4: $\pi := \text{null}$
5: while $U \neq \emptyset$ do #While not all vertices have been aligned
6: $v_i := \text{selectNextNode}(G_1, U)$
7: $v_l := \text{selectCandidate}(C, i, T, S)$
8: $\pi_i := l$
9: $U := U \setminus \{v_i\}$
10: $C := C \setminus \{v_l\}$
11: end while
12: output: solution $\pi$

$G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ and the pheromone model $T$. At each construction step, $U \subseteq V_1$ denotes the set of so-far unaligned nodes from the source network. Furthermore, $C \subseteq V_2$ denotes the set of possible target nodes from the target network. At the start of a solution construction, these sets are initialized as follows: $U = V_1$ and $C = V_2$. Moreover, the partial solution $\pi$ is empty, that is, $\pi = \text{null}$.

At each construction step, first, function $\text{selectNextNode}()$ selects an unaligned node $v_i \in U$ from the source network. This selection is based on the assumption that it might be easier to find acceptable candidate nodes for those source nodes with a high number of already aligned neighbors. A parameter $d_{\text{select}}$ (called deterministic rate) is used in order to decide whether this choice is done in a deterministic way (i.e., node $v_i \in U$ with the highest number of aligned neighbors is chosen) or selected using roulette-wheel selection (i.e., selected with probability proportional to the number of aligned neighbors). Second, a target node $\pi_l \in C$ is selected among all the possible candidates in function $\text{selectCandidate}()$. To this end, for each option $\pi_j \in C$, a probability $p_{ij}$ is calculated as follows

$$p_{ij} = \frac{\eta_{ij} \cdot \tau_{ij}}{\sum_{\pi_k \in C} \eta_{ik} \cdot \tau_{ik}}, \quad (1)$$

where $\eta_{ij} = s_{ij} \cdot \phi_{ij}$ is the greedy information, which is obtained by combining two factors:

1. $s_{ij}$, the pairwise similarity score between nodes $v_i$ and $\pi_j$;
2. $\phi_{ij}$, an improvement score estimating the score increase of the current partial solution if $v_i$ is aligned to $\pi_j$.

In this way, AntNetAlign incorportes two different sources of information into the solution construction process. The similarity score ($s_{ij}$) provides global information which is independent of the current state of the solution, while the improvement score ($\phi_{ij}$) allows to discriminate between the alignments based on the already aligned nodes. A description of how these two values are computed is presented in the following.

After the generation of the probabilities of Equation 1, a node $\pi_l \in C$ is selected by making use again of a determinism rate $d_{\text{align}}$. More specifically, the node with the highest probability is chosen with probability $d_{\text{align}}$. Otherwise, a node is selected by roulette-wheel selection proportional to the candidate nodes’ probabilities. Finally, $v_i$ is mapped the chosen target node $\pi_l$. This procedure is repeated until $U = \emptyset$, i.e., no nodes from the source network remain unaligned.

2.1.1 Calculation of the pairwise similarity scores

For computing the pairwise similarity between the nodes of both networks, AntNetAlign uses the topological similarity matrix introduced in the context of NetAL [18]. More specifically, a
topological score matrix $S$ is constructed, where each position $s_{ij}$ of $S$ specifies the topological similarity between nodes $v_i \in V_1$ and $v_j \in V_2$. For calculating these similarity values, NETAL assumes that topologically similar nodes must have similar neighbors. Details can be found in [18].

2.1.2 Calculation of the improvement scores

For computing the improvement score $\phi_{ij}$, we denote $\text{ind}_{G_1}(i,j)$ and $\text{ind}_{G_2}(i,j)$ as the number of edges that would be induced in the source and target networks, respectively, by aligning node $v_i \in V_1$ with node $v_j \in V_2$, i.e.,

$$\text{ind}_{G_1}(i,j) = |\{v_k \in N(v_i) \mid v_k \in A\}|,$$

$$\text{ind}_{G_2}(i,j) = |\{v_k \in N(v_j) \mid v_k \in f(A)\}|,$$

being $A = V_1 \setminus U$ the set of already aligned nodes from the source network. Similarly, let $\text{con}(i,j) = |\{v_k \in N(v_j) \mid v_k \in f(N(v_i) \cap A)\}|$, i.e., the number of edges that would be conserved. Note that $\text{con}(i,j) \leq \text{ind}_{G_2}(i,j)$. Therefore, we define

$$\phi_{ij} = \frac{\text{con}(i,j) + \epsilon}{\text{ind}_{G_1}(i,j) + \text{ind}_{G_2}(i,j) - \text{con}(i,j) + \epsilon},$$

where $\epsilon$ is a very small constant to prevent getting a value of zero in the numerator or denominator.

With this improvement score, the authors’ intention is to map nodes within areas of similar density while the solution is being constructed. Despite the fact that some optimizing measures do not take into account the induced edges in the target network (conversely, in the source network), they consider that keeping the same improvement formula can be beneficial no matter the objective function used. For instance, if the improvement score did not consider the number of induced edges in the target network when maximizing $\text{EC}$, the algorithm might start greedily aligning nodes within sparse areas of the source network to denser ones at the early stages of the construction process, given that it is easier to conserve edges this way. Nevertheless, this would be detrimental in the long run, as it could cause the densities of the remaining unaligned regions to be notably different, thus being harder to conserve edges at later stages of the construction process.

2.2 Adding Negative Learning

The Negative Learning mechanism introduced in [19] requires a second pheromone model for storing and using the learned negative information. Therefore, we introduce a second pheromone model $T_{\text{neg}}$, composed of a pheromone value $\tau_{\text{neg}}_{ij}$ for each pair of nodes $v_i \in V_1$ and $v_j \in V_2$, i.e., for each possible alignment. In contrast to the standard pheromone values which are initialized to 0.5 at the start of the algorithm, the negative pheromone values are set to $\tau_{\text{min}}$ each time the algorithm is (re-)started.

At each iteration of AntNetAlign, after generating $n_{\text{ants}}$ solutions, a sub-instance of the tackled problem instance is created in which only a subset of all possible alignments is considered. More specifically, the sub-instance consists only of those alignments that form part of at least one of the solutions generated at that iteration. Then, an independent algorithm is applied to find the best possible solution to this sub-instance (within a pre-defined time limit). After that, the negative pheromones are updated as outlined below. Finally, an algorithm iteration finishes with the update of the standard pheromone values, which remains unchanged.

In the following we first describe how the sub-instance is generated and solved. Then, we outline the update of the negative pheromone values. Finally, we explain how the negative pheromone values are used to bias the construction of solutions (in addition to the standard pheromone values).
2.2.1 Generating and Solving Sub-instances

As mentioned above, at each iteration of AntNetAlign a sub-instance of the tackled problem instance is generated by merging the $n_{ants}$ solutions generated at that iteration. In the following we explain our way to model a sub-instance as an Integer Linear Programming (ILP) model which can then be solved by an ILP solver. For this purpose, we first present an ILP model for the NA problem.

\[
\text{max } \sum_{e_{ik} \in E_1, e_{jl} \in E_2} c_{ijkl} \tag{2}
\]

\[
\text{s.t. } \sum_{v \in V_2} x_{ij} = 1 \quad \forall \ v_i \in V_1 \tag{3}
\]

\[
\sum_{v \in V_1} x_{ij} \leq 1 \quad \forall \ \bar{\tau}_j \in V_2 \tag{4}
\]

\[
x_{ij} + x_{kl} \geq 2c_{ijkl} \quad \text{for each pair } e_{ik} \in E_1, e_{jl} \in E_2 \tag{5}
\]

\[
x_{ij} \in \{0, 1\} \quad \text{for all pairs } v_i \in V_1, \bar{\tau}_j \in V_2 \tag{6}
\]

\[
c_{ijkl} \in \{0, 1\} \quad \text{for all pairs } e_{ik} \in E_1, \bar{\tau}_j \in E_2 \tag{7}
\]

First, this model works with a binary variable $x_{ij}$ for each possible alignment of a node $v_i \in V_1$ and a node $\bar{\tau}_j \in V_2$, indicating whether or not vertex $v_i \in V_1$ is aligned with vertex $\bar{\tau}_j \in V_2$. Second, the model has another binary variable $c_{ijkl}$ for each pair of edges $e_{ik} \in E_1$ and $e_{jl} \in E_2$. These variables will indicate conserved edges. Constraints (3) and (4) ensure that each node of $G_1$ is aligned to exactly one node from $G_2$ and that each node from $G_2$ is aligned to at most one node from $G_1$, respectively. Furthermore, constraints (5) guarantee that an edge $e_{ik} \in E_1$ is only indicated as conserved, if (1) there is another edge $e_{jl} \in E_2$, (2) node $v_i$ is aligned with node $\bar{\tau}_j$, and (3) node $v_k$ is aligned with node $\bar{\tau}_l$. Note that it can not happen that more than one variable $c_{ijkl}$ is assigned value one, as the previous constraints ensure that each node of the source network is aligned with exactly one node of the target network.

Note also that the objective function of our ILP model maximizes the number of conserved edges in the source network, i.e., the EC score. This is because the EC score allows for a linear ILP model. Moreover, our hypothesis is that, if the AntNetAlign algorithm uses a different score such as, for example, the $S^3$ score, then vertices in the obtained sub-instance will be restricted to be aligned with other vertices within regions of similar density.

In order to use the ILP from above for solving a sub-instance of the tackled problem instance, the following is done. First, a variable $x_{ij}$ is removed from the model if vertex $v_i$ was not aligned to node $\bar{\tau}_j$ in at least one of the $n_{ants}$ solutions generated at the current iteration. Moreover, these variables are removed from constraints (5) and (7). Second, we remove a variable $c_{ijkl}$ (and the corresponding constraint [7]) if either $x_{ij}$ or $x_{kl}$ were removed from the model. The resulting reduced ILP model corresponds to the sub-instance obtained by merging the solutions constructed at the current iteration of AntNetAlign. In the context of this paper, we applied the ILP solver CPLEX 20.1 for solving sub-instance.

As it is not guaranteed that CPLEX always terminates within a reasonable amount of time, we limit the computation time of CPLEX for each call depending on an upper bound of $t^{sub}$ CPU seconds and on the current value of the convergence factor $0 \leq cf \leq 1$ of AntNetAlign. More specifically, the solver is given a time limit of $(1-cf)t_{\text{sub}} + 0.1cf$ CPU seconds, i.e., the computation time limit decreases as the convergence factor increases. For all experiments we used a maximum computation time of $t^{sub} = 60$ seconds, which is based on our preliminary observations of the behavior of the algorithm. The motivation behind making the time limit of CPLEX dependent on the convergence factor is that, when the convergence factor is low, the high variance between the constructed solution results in larger sub-instances, i.e., more possible alignments are considered. Thus, a larger amount of time is needed to solve it.

Finally, note that CPLEX is warm-started with the iteration-best solution (denoted as $\pi^{ib}$) of AntNetAlign in order to avoid that the solver output is worse than the iteration-best solution.
(possibly due to a restrictive time limit in some cases). After the application of the solver, \( \pi^{ib} \) is updated with the solver output.

### 2.2.2 Negative Information Update

The negative pheromone values are updated based on the iteration-best solution \( \pi^{ib} \) (which might have been improved by solving the sub-instance of the current iteration). For this purpose we split the set of alignment of the current sub-instance into two sets. First, the set of contributing alignments are those alignments of \( \pi^{ib} \) that contribute to edges of the source network being conserved. Note that this is not necessarily the case for all alignments of \( \pi^{ib} \). The remaining set of alignments in the sub-instance are labeled non-contributing alignments. In order to penalize a non-contributing alignment of a node \( v_i \) with a node \( v_j \), the corresponding update formula for the negative pheromones is as follows:

\[
\tau_{ij}^{neg} := \tau_{ij}^{neg} + \rho^{neg} \cdot (\xi_{ij}^{neg} - \tau_{ij}^{neg}),
\]

where \( \rho^{neg} \) is the negative learning rate and \( \xi_{ij}^{neg} = 0 \) in case the corresponding alignment is a contributing alignment (or \( \xi_{ij}^{neg} = 1 \), otherwise). This update only concerns those alignments that form part of the sub-instance, i.e., that form part of at least one of the solutions generated at that iteration.

### 2.2.3 Negative Information Use

The negative pheromone values are used, in addition to the standard pheromone values, when constructing solutions in \textsc{AntNetAlign}. More specifically, Equation \( \mathbf{1} \) is replaced by:

\[
p_{ij} = \frac{\eta_{ij} \cdot \tau_{ij} \cdot (1 - \eta_{ij}^{neg})}{\sum_{\alpha \in C} \eta_{\alpha \cdot \tau_{\alpha} \cdot (1 - \eta_{\alpha}^{neg})}}.
\]

Hence, those alignments that have a high negative pheromone values (either because they have not appeared in previous solutions derived by CPLEX or they appeared but did not conserve any edge) have lower probabilities of being selected for the construction of solutions in subsequent iterations.

### 3 Experimental Evaluation

In order to analyze the performance of our approach, henceforth referred to as \textsc{AntNetAlign\_neg}, we compare it with the standard version of \textsc{AntNetAlign} and with two current state-of-the-art algorithms: \textsc{Netal} \cite{18} and \textsc{L-Graal} \cite{16}. Both methods are popular in the respective areas and share certain algorithmic similarities with our algorithms. \textsc{Netal} is a heuristic from which we adopted the topological score matrix. All the parameters are left to their default values. Moreover, only the topological similarity is considered. \textsc{L-Graal} is the most recent member of the well-known \textsc{Graal} family. It uses a novel alignment search heuristic based on Integer Programming and Lagrangian relaxation. All the parameters are set to default, with a maximum number of 1000 iterations and a time limit of 1 hour. The graphlet signatures are calculated with the provided software, with a graphlet size of 4. For the experimental evaluation, we used the implementations provided by the respective authors. We perform an experimental evaluation using real-world networks. Our own algorithms were implemented in ANSI C++ and compiled using GCC 7.5.0. Furthermore, CPLEX 20.1 was used for solving the ILPs at each iteration of \textsc{AntNetAlign\_neg}. The experiments were performed on a cluster of computers with “Intel\textsuperscript{®} Xeon\textsuperscript{®} CPU 5670” CPUs of 12 nuclei of 2933 MHz and (in total) 32 Gigabytes of RAM \cite{11}.
Table 1: Benchmark instances (high-quality PPI networks)

<table>
<thead>
<tr>
<th>Scientific Name</th>
<th>Label</th>
<th>Organism</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>B. taurus</td>
<td>bt</td>
<td>cattle</td>
<td>119</td>
<td>114</td>
<td>1.91</td>
</tr>
<tr>
<td>O. sativa</td>
<td>os</td>
<td>rice</td>
<td>181</td>
<td>192</td>
<td>2.12</td>
</tr>
<tr>
<td>R. norvegicus</td>
<td>rn</td>
<td>rat</td>
<td>289</td>
<td>236</td>
<td>1.63</td>
</tr>
<tr>
<td>B. subtilis</td>
<td>bs</td>
<td>bacterium</td>
<td>334</td>
<td>775</td>
<td>4.64</td>
</tr>
<tr>
<td>E. coli</td>
<td>ec</td>
<td>bacterium</td>
<td>709</td>
<td>692</td>
<td>1.95</td>
</tr>
<tr>
<td>S. pombe</td>
<td>sp</td>
<td>yeast</td>
<td>1,335</td>
<td>2,176</td>
<td>3.26</td>
</tr>
<tr>
<td>M. musculus</td>
<td>mm</td>
<td>mouse</td>
<td>1,966</td>
<td>2,704</td>
<td>2.75</td>
</tr>
<tr>
<td>C. elegans</td>
<td>ce</td>
<td>worm</td>
<td>4,762</td>
<td>11,889</td>
<td>4.99</td>
</tr>
<tr>
<td>S. cerevisiae</td>
<td>sc</td>
<td>yeast</td>
<td>5,260</td>
<td>22,064</td>
<td>8.39</td>
</tr>
<tr>
<td>A. thaliana</td>
<td>at</td>
<td>plant</td>
<td>5,686</td>
<td>25,306</td>
<td>8.90</td>
</tr>
<tr>
<td>D. melanogaster</td>
<td>dm</td>
<td>fly</td>
<td>8,382</td>
<td>31,754</td>
<td>7.58</td>
</tr>
<tr>
<td>H. sapiens</td>
<td>hs</td>
<td>human</td>
<td>12,466</td>
<td>60,248</td>
<td>9.67</td>
</tr>
</tbody>
</table>

3.1 Benchmark instances

Our experiments use a set of Protein-Protein Interaction (PPI) networks, representing pairwise biophysical protein interactions of organisms. More specifically, we use the PPI networks from HINT \[5\], a database of high-quality protein-protein interactomes. Note that the objective of these experiments is not to provide an analysis from a biological point of view, but rather to make use of real-world instances that can help to validate the correctness and applicability of our approach. A summary of the PPI networks used in this work is presented in Table 1. For each network, the scientific name of the corresponding organism is given, along with a label and its generic name. The labels will be henceforth used to identify each network. Additionally, we provide the order (number of proteins) and size (number of interactions) of each network, alongside the mean degree \( \langle k \rangle = \frac{|E(G)|}{|V(G)|} \). As it can be observed, the tested networks significantly differ in size and density. Given that many methods for the NA problem, specially in this area, we do not allow neither multi-edges nor isolated vertices (self-loops), and thus they are previously removed from the networks.

3.2 Tuning

Finding suitable values for the parameters of AntNetAlign and AntNetAlign\_neg is required for a correct experimental evaluation. Some of them, however, are set by means of preliminary experiments. For instance, the maximum number of solution constructions per run is set to 1000, since the algorithms can converge (even multiple times) before this limit is reached. In the case of AntNetAlign\_neg, we limit the run time additionally by setting a maximum run time of 1 hour in order to be consistent with L-Graal, which also makes internal use of an ILP solver.

For the remaining parameters, we make use of the irace tool \[15\] in order to choose their best settings. In particular, we selected two pairs of medium-sized networks as tuning instances. More specifically, E.coli is aligned with S.pombe (ec-sp), and S.pombe with M.musculus (sp-mm).

Table 2 shows a summary of the parameters chosen for tuning, along with the considered domains. Both ACO versions require values for (1) \( n_{ants} \) (solution constructions per iteration), (2) \( \rho \) (learning rate), (3) \( d_{select} \) (determinism rate, source network), and (4) \( d_{align} \) (determinism rate, target network). Additionally, AntNetAlign\_neg requires values for \( \rho_{neg} \) (negative learning rate) and emphasis (MIP emphasis parameter of CPLEX). Two separate tuning runs for both AntNetAlign and AntNetAlign\_neg were conducted, one for maximizing the EC score and the other one for the \( S^3 \) score. Hereby, irace was given a budget of 750 in the case of AntNetAlign and 1000 in the case of AntNetAlign\_neg, given the higher number of parameters.

Table 2 also shows the best parameter values obtained from the tuning process. The most notable difference is that the number of solutions constructions per iteration \( \langle n_{ants} \rangle \) is higher for
AntNetAlign than for AntNetAlign\textsubscript{neg}. One of the reasons could be that a lower number of solutions constructions leads to smaller sub-instance sizes, which makes it easier for the ILP solver to find better solutions within the given time limit. Concerning the other parameters, the degree of determinism when selecting the next node to align is high in all the cases, but still low enough to ensure a certain degree of randomness. Moreover, a rather deterministic selection of the candidate nodes is also required. Concerning the learning rate ($\rho$), AntNetAlign\textsubscript{neg} seems to require a slightly higher value than AntNetAlign. The results for the negative learning rate are rather curious. While a rather low one seems to be required when optimizing EC, a value from the middle of the range is indicated when optimizing $S^3$. Finally, we are not sure if the results for the MIP emphasis value of CPLEX are really significant (a value of '0' means that CPLEX keeps a balance between optimality and feasibility, while a value of '2' indicates a focus on optimality).

### Table 2: Tuning results.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Tested Domain</th>
<th>Chosen Values</th>
<th>AntNetAlign</th>
<th>AntNetAlign\textsubscript{neg}</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n_{ants}$</td>
<td>(3, 5, 10, 20)</td>
<td>10 10 5 5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\rho$</td>
<td>${0.1, \ldots , 0.5}$</td>
<td>0.1 0.3 0.3 0.4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$d^\text{select}$</td>
<td>${0.0, \ldots , 1.0}$</td>
<td>0.7 0.8 0.6 0.7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$d^\text{align}$</td>
<td>${0.0, \ldots , 0.9}$</td>
<td>0.9 0.9 0.9 0.9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\rho^\text{neg}$</td>
<td>${0.1, \ldots , 0.5}$</td>
<td>– – 0.1 0.4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\text{emphasis}$</td>
<td>${0, \ldots , 5}$</td>
<td>– – 0 2</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

3.3 Results

In order to test the performance of our approach we performed a set of experiments consisting of the alignment of all different pair combinations of the 12 high-quality PPI networks from Table 1. This makes a total of $(12 \times 11)/2 = 66$ problem instances. Keep in mind that the source network must have a smaller order than the target network. Thus, the total number of viable combinations is divided by two. By taking into account all the possible combinations, we obtain a broad data set on which we can test the behavior of our algorithms in different scenarios. As our two approaches and Netal are of stochastic nature, we apply each of these three algorithms 10 times to each problem instance.

Figures 1 and 2 show the obtained results regarding the $S^3$ and EC scores, respectively. Labels on the x-axis specify the respective problem instances, ordered in increasing order of the source and target networks, respectively. For each algorithm and tested instance, we display the obtained mean (dot) and standard deviation (vertical line) of the corresponding results. A gray background is used to separate the instances with respect to their source networks, along with horizontal lines joining the different results. However, it is important to note that there exists no real relation between these neighboring instances (except having a common source network), and that this is only used for visual clarity.

After examining the obtained results regarding the $S^3$ score, the following analysis can be made. Both ACO versions are clearly the best-performing methods. This difference is accentuated with an increasing difference in the order of the source and target networks. The same happens when comparing the two ACO versions, that is, the advantage of AntNetAlign\textsubscript{neg} over AntNetAlign increases with an increasing difference in the order of source and target networks. In any case, AntNetAlign\textsubscript{neg} outperforms AntNetAlign in nearly all of the tested instances.

\footnote{Note: Contrarily to other methods, L-Graal considers the source network to have a smaller size (number of edges) than the target network, rather than a smaller order (number of nodes). Therefore, the bs and ec networks are swapped when aligning them using this algorithm. Thus, as the presented results in this instance may not be consistent with respect to the other methods, we indicate this case with an asterisk (*).}
Figure 1: Results concerning the $S^3$ score for all pairs of high-quality PPI Networks.
Figure 2: Results concerning the EC score for all pairs of high-quality PPI Networks.
Note that, even when optimizing a certain measure—as for example $S^3$ in the case of Figure 1—the obtained solutions can also be evaluated in terms of the other measures. When examining the solutions of AntNetAlign obtained by optimizing the $S^3$ score (graphical results are not provided due to space restrictions) it can be observed that the EC score of these solutions is usually much better than the ICS score. Only for one specific instance (instance rn-bs) this is the other way around. Interestingly, in this specific case, AntNetAlign slightly outperforms AntNetAlign\textsubscript{neg}. Our reading of this result is as follows. As EC score of the AntNetAlign solutions in this specific case is already good, the use of negative learning which is based on solving sub-instances with an ILP solver (optimizing the EC score) does not pay off. This would indicate that maximizing EC in the context of generating the negative learning information can help to improve the performance of the algorithm when maximizing $S^3$, specially in those cases in which the constructed solutions in standard AntNetAlign have a lower EC score than ICS score.

Regarding the performance of the other algorithms, L-Graal obtains rather consistent results, consistently worse than the ACO approaches. Its performance drops dramatically as the size of the instances increases, sometimes obtaining worse results than Netal. This decrease in performance for some specific instances—e.g. bt-bs or rn-ce—can be explained by the dissimilarity of the aligned networks, because L-Graal conducts a filtering by means of the similarity threshold in order to generate good sub-instances to solve. It is also worth mentioning that L-Graal is, sometimes, not able to fully converge within the given time limit.

On the other hand, Netal is the worst-performing method. Nonetheless, it is still able to achieve good-quality solutions when aligning networks of similar order. Moreover, this method seems to suffer sometimes from its random decisions, as, in some cases, the standard deviations are rather large, specially for smaller instances. Some examples for which this phenomenon is more accentuated are the bt-ec or ec-mm instances. However, the method seems more robust as the size of the instances increases, which makes sense as one miss-alignment in the context of smaller instances can be much more penalizing.

Concerning the results when optimizing the EC score, the following observations can be made. Again, AntNetAlign\textsubscript{neg} outperforms AntNetAlign nearly for all problem instances. Moreover, for the instances with os as source network (i.e., os-\* instances) AntNetAlign\textsubscript{neg} achieves lower standard deviations than AntNetAlign, meaning that negative learning sometimes helps to produce more robust results. For small and medium instances, AntNetAlign\textsubscript{neg} outperforms both standard AntNetAlign and L-Graal. This difference is more visible in instances such as those obtained when aligning os and ec networks with others. Netal usually starts with low-quality alignments when aligning a network with others of similar size, but starts increasing in performance when the difference between the order of the networks becomes more notable. The same holds for the standard deviation of its results, with the variance between solutions being greater when both networks resemble. In those cases in which Netal performs well, AntNetAlign\textsubscript{neg} is nevertheless capable to compete with it. This holds, for example, for instances with bt, rn and ec as source networks (i.e., bt-\*, rn-\* and ec-\* instances).

In the context of larger instances, Netal outperforms the other methods, except when aligning at with dm and hs, where all the methods show a fairly similar performance. Regarding the other methods, L-Graal generally outperforms the ACO approaches. However, this difference is not significant in many cases, with AntNetAlign\textsubscript{neg} outperforming L-Graal, for example, when aligning the sp network.

4 Conclusions and Future Work

In this work we proposed an extension of a recent ACO approach, AntNetAlign, for network alignment. The extension is characterized by adding a negative learning mechanism to AntNetAlign, in line with the work in [19]. At each iteration, the proposed method creates a sub-instance of the tackled problem instance by merging the solutions constructed at the same iteration. This sub-instance is then modeled as an integer linear program and solved using the
A second pheromone model is used for accumulating the negative learning experience for those individual node-to-node alignments that appeared in ACO solutions, without being included into the best solution of the respective sub-instances. Although the ILP formulation maximizes the edge conservation score, the negative information generated is still helpful even when the main ACO approach maximizes the $S^3$ score, specially in those cases in which the EC score of the constructed solutions is higher than the ICS score. This makes our proposed approach the best-performing one concerning this score. Regarding the EC score, our negative learning approach is able to outperform existing approaches for small and medium size instances, while it has some deficiencies in the context of larger problem instances.

In future work, we aim to test other alternatives for generating and updating the negative information. Furthermore, we will try to apply this framework to other algorithms and problems of different nature.

References


