Stigmergic Swarming Agents for Fast Subgraph Isomorphism

H. Van Dyke Parunak
ABC Research
Ann Arbor, MI, USA
van.parunak@gmail.com

ABSTRACT

Maximum partial subgraph isomorphism compares two graphs (nodes joined by edges) to find a largest common subgraph. A common use case, for graphs with labeled nodes, seeks to find instances of a *query* graph with q nodes in a (typically larger) *data* graph with d nodes. The problem is NP-complete, and naïve solutions are exponential in $q + d$. The fastest current heuristic has complexity $O(d^2)$. This paper outlines ASSIST (Approximate Swarming Subgraph Isomorphism through Stigmergy), inspired by the ant colony optimization approach to the traveling salesperson. After peering (identifying matching individual nodes in query and data) in time $O(q \cdot log(d))$, the time required for ASSIST's iterative subgraph search, the combinatorially complex part of the problem, is linear in query size and constant in data size. ASSIST can be extended to support matching problems (such as temporally ordered edges, inexact matches, and missing nodes or edges in the data graph) that frustrate other heuristics.

CCS CONCEPTS

• **Mathematics of computing → Graph algorithms**; • **Computing methodologies → Multi-agent systems**; **Randomized search**.

KEYWORDS

Swarming agents; Stigmergy; Subgraph Isomorphism; Ant colony optimization; ACO

ACM Reference Format:
H. Van Dyke Parunak. 2026. Stigmergic Swarming Agents for Fast Subgraph Isomorphism. In *Proc. of the 25th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2026), Paphos, Cyprus, May 25 – 29, 2026*, IFAAMAS, 9 pages. https://doi.org/10.65109/DEZC1973

1 INTRODUCTION

The maximum partial subgraph isomorphism problem is: given two graphs G_1, G_2, find a largest subgraph of G_1 that is isomorphic to a subgraph of G_2. The problem is well-defined whether the graph nodes of the graphs are labeled from some set $\mathbf{L}$ of size $|\mathbf{L}| > 1$ (with some labels possibly repeated), or whether they are unlabeled (equivalently, labeled from a set of size 1). In both cases, the problem is NP-complete. (Our solution requires labeled graphs, but allows some labels to be repeated.)

For example, chemists frequently seek shared structures in large organic molecules, where nodes are the atoms in a molecule (e.g., C, H, O, ...), and edges are chemical bonds between them. Finding a common structure of six atoms between two organic molecules, each of 50 atoms, with a naïve enumeration requires over 10^{17} comparisons, motivating chemists to pursue research in subgraph isomorphism algorithms [34].

In addition to molecular design, many other applications would benefit from the ability to compare even larger graphs. For example:

(1) NOSQL databases maintain networks of relationships among millions of data items. A natural query is a graph of possible relations [21], and analysts seek a largest subgraph in the query that occurs in the data [32].
(2) Financial transaction data is a primary data source for detecting financial crime such as money laundering, if its petabytes of data can be searched efficiently for patterns detailing known transactional behaviors [10].
(3) IP packet data is an important resource for network monitoring. A connection is a five-tuple <source-IP, source-port, dest-IP, dest-port, protocol>, and an edge joins one connection C1 to another C2 just in case the start time of C1 is earlier than that of C2 and at least one IP address is repeated between them. Such graphs will have millions of connections for reasonable periods of time. Common subgraphs between graphs representing different networks, or the same network at different times, highlight shared behaviors that might indicate common actors [16, 25].
(4) A narrative space [33, 36] fuses many possible causal trajectories, and enables highly-understandable social simulations. Such graphs, which can contain hundreds of nodes, enable analysts to visualize and interact in both forensic and forecasting problems, but authoring them is time consuming. Many domains maintain collections of narratives about specific past events [1, 11, 20, 23, 41]. Fusing these narratives at their common subgraphs would yield a narrative space for their domain, greatly accelerating the analytic process.
(5) Fusing patient records in health care can generate a causal model of diagnoses, treatments, and outcomes for resource forecasting and fraud detection [13].
(6) Image recognition makes use of feature graphs that capture adjacency information among different features [24, 27]. More efficient subgraph isomorphism algorithms would allow powerful new image search capabilities.

In these and similar cases, the graphs range from 10^3 to 10^6 nodes or even more. With even the 50-node graphs of organic chemistry posing computational limits, a faster algorithm is clearly important.

This paper describes ASSIST (Approximate Swarming Subgraph Isomorphism through STigmergy), a novel heuristic inspired by ant colony optimization (ACO) [37]. "Stigmergy" refers to coordination of agents by making and sensing changes in a shared environment [17], rather than by direct inter-agent messaging. It is inspired by

Proc. of the 25th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2026), C. Amato, L. Dennis, V. Mascardi, J. Thangarajah (eds.), May 25 – 29, 2026, Paphos, Cyprus. © 2026 International Foundation for Autonomous Agents and Multiagent Systems (www.ifaamas.org). https://doi.org/10.65109/DEZC1973

Table 1: ASSIST compared with other methods for subgraph isomorphism

ASSIST Feature	*Competition*	*Benefits of ASSIST*
Heuristic	Exact [28, 40]	Scalability, Robustness
Direct	Transformed [4, 26, 44, 45]	Robustness, Accessibility
Stochastic	Deterministic [18, 26, 28, 35, 38–40, 44, 45]	Tunable Accuracy
Multiple Solutions	Single Solution [18, 26, 28, 35, 38–40, 44, 45]	Robustness, Accessibility
Incremental	Entire [4, 9, 26, 28, 40, 42, 44, 45]	Scalability

social insects such as ants and termites, who coordinate their work by depositing chemicals ("pheromones") in the environment and making decisions based on the current pheromone strengths in their vicinity. These chemicals evaporate over time, and locations that are reinforced by many ants converge to the selected solution. In nature, these mechanisms enable termites to construct mounds with separate floors and rooms, and ventilation systems to exhaust waste gasses. Ants use them to construct minimal spanning trees joining their nests to food sources. ACO algorithms replace the ants with simple software agents, the pheromones with increments that the agents make to variables on the different locations that they visit, and evaporation with a periodic attenuation of the pheromones by a specified percentage. The results have proven successful in other highly complex problems such as the traveling salesperson. ASSIST applies these techniques to subgraph isomorphism.

In addition to efficient subgraph matching, ASSIST (like other swarming algorithms) demonstrates how stigmergy can integrate partial results obtained by many independent agents without sophisticated coordination mechanisms, inviting its application to other problems of interest to the AAMAS community.

Section 2 compares ASSIST with other approaches to subgraph isomorphism. Section 3 describes the swarming traveling salesperson (TSP) algorithm that inspires ASSIST. Section 4 describes the ASSIST algorithm. Section 5 reports experimental results. Section 6 discusses future work. Section 7 concludes.

2 RELATED WORK

We compare ASSIST first with other graph matching algorithms, then with other stigmergic systems. Graph matching is an active research area. Convenient surveys include [6, 8, 14, 34]. We situate ASSIST in this context along five dimensions, highlighting how its features deliver four key benefits: Tunable Accuracy, Scalability, Robustness, and Accessibility (understandable by non-technical users). Table 1 summarizes the contribution of each feature to the benefits. We discuss the rows in order.

ASSIST is a *heuristic*. *Exact methods* (such as Ullman's pioneering algorithm [40]) are guaranteed to find matching subgraphs, but do not scale to large problems, and they are not robust to noise in the query or data. Successive filtering methods [28] apply a sequence of exact methods to the data, hoping to reduce its size before attempting to identify subgraphs, but are not robust to incompletely labeled data.

ASSIST manipulates the query and data *directly*, making it robust to poorly conditioned graphs and accessible to analysts. Spectral methods [26, 44], Estimation of Distribution Algorithms (EDAs [4]), and Optimal Transport approaches [45] (with best current complexity $O(d^2)$) *transform* the problem (into matrix parameters, probabilistic graphical models, or probability distributions, respectively), which can compromise robustness and accessibility.

ASSIST, like EDAs [4] and genetic algorithms (GAs) [9, 42], is *stochastic*, allowing it to escape from local optima, and also permitting tunable accuracy, dynamically trading off probability of detection (p_d) and false acceptance rate. Most methods are *deterministic*, giving the same answer each time they run, but are vulnerable to local optima, lacking tunable accuracy.

ASSIST considers *multiple* solutions concurrently, ranking them probabilistically (by the strength of the pheromone field on each one) and thus enhancing accessibility. Most methods produce a *single* matching, which is best by some criterion but may not be robust to corrupted patterns or data. EDAs and GAs consider a population of individual competing solutions, allowing alternative matches to emerge, but typically focus down to a single solution, so that analysts never see alternatives.

ASSIST, like some other methods [18, 35, 38, 39], is *incremental*, starting with local matches within each of the two graphs and expanding the match. Incremental match construction enhances scalability by limiting the effort spent on candidate matches that end up failing. Most methods reason about the *entire* query at once.

Stigmergic swarming, the heart of ASSIST, has been applied successfully in many areas, including the traveling salesperson problem [37], telecommunications routing [22], and geospatial reasoning [31]. ASSIST extends these techniques. Unlike geospatial reasoning (but like routing problems) it is not limited to the regular structure of a lattice, but handles arbitrary graphs. Unlike previous graph applications, it has separate pheromone families for nodes and edges, and can be extended to deal with time-sequenced graphs to handle time compactly.

3 AN EXAMPLE OF SWARMING GRAPH COMPUTATION

One of the most successful applications of swarming agents to graph-theoretic problems is to the traveling salesperson problem (TSP): given a set of points in two dimensions and a road network among them, find the shortest Hamiltonian circuit (visiting every node exactly once and returning to the start). This problem is of great importance in problems such as logistics [15] (optimizing fuel usage for truck fleets), telecommunications [5], and wiring plans for printed circuit boards [2], and is successful enough to find widespread commercial use (e.g., www.antoptima.com).

In these applications, successive waves of software agents (digital ants) explore alternative tours in parallel. Each ant chooses at each step among vertices it has not visited, preferring edges with the strongest pheromones left by previous ants, and returning home when all vertices have been visited. After completing a circuit, each ant deposits pheromone on the edges it has traversed,

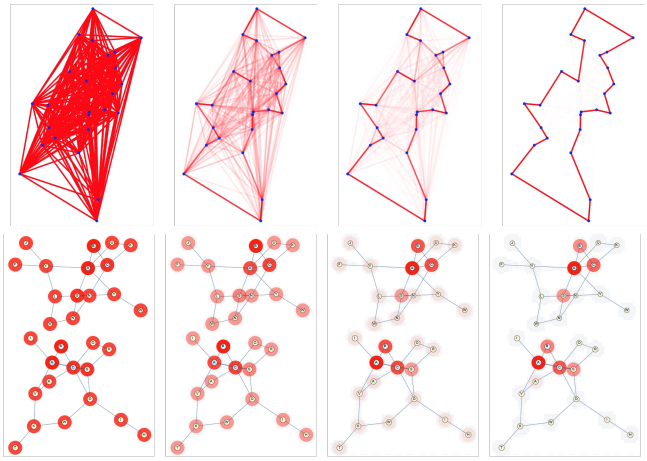

Figure 1: Swarming solutions of the traveling salesperson and subgraph isomorphism

with strength inversely proportional to the length of its overall path. After each wave of agents explores the graph, all pheromone strengths evaporate, multiplied by $1 - \rho$, $\rho < 1$. Edges that are part of shorter circuits accumulate more pheromone, and attract more agents, while edges that are not visited evaporate, and a highly competitive path emerges (Figure 1, top). Depending on details of the application, the time complexity is $O((n \cdot log(n))/\rho)$ [29].

The ASSIST algorithm described in the next section (Figure 1, bottom) uses the same mechanisms of constant pheromone evaporation and selective deposit. Through iteration, a maximal subgraph emerges (in this case, nodes A-B-C-S). (The edge from A to S in the lower graph is present, but obscured by C.)

4 THE ASSIST HEURISTIC

Instead of Hamiltonian circuits, ASSIST seek paths between the query and data graphs that traverse matching fragments of both graphs (typically, a single edge). Figure 2 illustrates the movements of a single agent. Starting at a node labeled A in the query, it seeks an A node in the data that has similar neighbors to its original node (1). Then it looks for a neighbor in the data that matches a neighbor of A in the query (B), returns to a B node in the query, and seeks to complete the circuit. If it is successful, it has found a candidate edge in the desired matching, and augments the pheromones on the nodes and edges in both the query and the data.

Matching edges aggregate into larger subgraphs as swarming agents seek for neighbors of already-matched nodes and reinforce the pheromone levels on those nodes and the edges that join them. Nodes that participate in more than one shared edge are further reinforced. Multiple edges in a shared subgraph reinforce each other, and the larger the subgraph, the more pheromone its nodes and edges accumulate. Pheromones on nodes in either graph that do not have shared edges eventually evaporate to 0. As the pheromones from many agents stabilize, a high quality matching emerges.

The following sections discuss 1) the data model for the graphs we analyze, 2) the classes and pseudocode of the algorithm, 3) how the graphs are initialized and possible matches identified, 4) the matching of a single edge, and 5) how we detect termination.

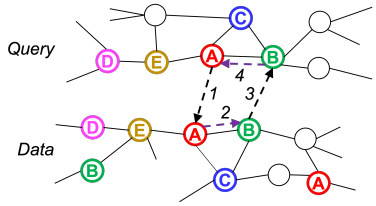

Figure 2: Each agent seeks a matching edge between query and data graphs

4.1 Data Model

For each experiment, we randomly generate three graphs:

(1) A *data* graph represents a graph database that we wish to explore against a query.
(2) A *query* graph represents a set of relations that we are seeking in the data graph. It is typically smaller than the data graph, and may be less specific.
(3) A common *kernel* graph is embedded in both the data graph and the kernel graph. We evaluate ASSIST based on its ability to retrieve the kernel.

We term such a set of three graphs, a *scenario*.

The kernel is not necessary to the operation of the algorithm, but supports evaluation. Having a known common subgraph in query and data lets us compare the effect of independent variables such as query, data, and vocabulary size or degree of ablation.

Our experiments use Barabási-Albert (BA) graphs, in which node degree follows a power law [3], generated with the NetworkX [19] function `barabasi-albert-graph()` with parameter $m = 2$ (each new node is attached to two existing nodes with probability proportional to their existing degree). Graphs with this structure are common in networks of associations, such as social networks or webs of internet sites, and are of particular interest in many graph-structured domains.

In many applications of graph matching, we may be searching for nodes of a given *type* without knowing their detailed *identity*. For example, our data may consist of a detailed graph of financial transactions, in which all participants (businesses, banks, individuals) are fully identified, and our query may be looking for a node of type "person" whose identity is unknown, but who has contacts with other, known individuals and who deals with a specific bank. To support this structure, we furnish each node with an alphabetic *label*, drawn from a fixed vocabulary **L**, representing its type, and a numerical *detail* that is distinct for each different node of a given type. A node in the query with detail = 0 will match any node of the same type (that is, the same label) in the data. Thus A23 and A42 might be specific, known people, while A0 would be a person whose identity is not known, and who may match either A23 or A42. The labels and details of the kernel are preserved in the query and the data, and other nodes in both query and data are generated with distinct label-detail identifiers. In most of our experiments, $|\mathbf{L}| = 100$, and all nodes have non-zero detail. Ablation experiments (Section 5.4) set some proportion of the details to 0, and also explore the effect of vocabulary size.

4.2 Classes and Pseudocode

In this section, "select from **X** by y" refers to roulette selection from the elements of set **X** weighted by attribute y of those elements.

ASSIST has three main classes of objects: the *nodes* $\mathbf{N}_q$, $\mathbf{N}_d$ of the query and data graphs, their undirected edges $\mathbf{E}_q$, $\mathbf{E}_d$ edges , and the swarming agents **A**. $\mathbf{N} \equiv \mathbf{N}_q \cup \mathbf{N}_d$, and $\mathbf{E} \equiv \mathbf{E}_q \cup \mathbf{E}_d$.

Each node has the following attributes:

- *pherLevel* (initially 1.0)
- *nbrPhers*, an array of length |**L**| with the total pheromone for nodes with each label among the node's neighbors
- *peers*, an array of peers in the other graph, each with a *weight* computed as the cosine distance between the *nbrPhers* arrays of the peered nodes
- *liveEdges*, number of edges adjacent to the node with *pherLevel* > 0

Each edge has *pherLevel* (initially 0.0).

Each agent has the following attributes:

- *start*, the node on which the agent started its search
- *location*, the node where the agent is currently located
- *mode* in {1, 2, 3, 4} tracking the agent's progress through the search algorithm outlined below
- *history*, a sequential list of nodes and edges traversed in both graphs; *history*[0] is starting *location*
- *startNbr*, the label of the largest element of the starting node's *nbrPhers* (selected from *history*[0].*nbrPhers* by roulette)

ASSIST is implemented in Repast [30], which measures time in *ticks*. Algorithm 1 shows the sequence of actions in each tick.

Algorithm 1 Sequence of events in each Repast tick

```
1:  procedure doOneTick()
2:      for each Node n in N do
3:          update n.liveEdges
4:      for each Node n in N_q do
5:          if |n.peers| > 0 then
6:              Update weight of each peer
7:              Initialize 1 + 2 · liveEdges agents with start ←
                    history[0] ← location ← n
8:      for each Agent a in A do
9:          Execute procedure step()          ▷ Deposits pheromones
10:     for each Node n in N do
11:         Query neighbors to update n.nbrPhers
12:     for each Edge n in N do               ▷ Evaporate node pheromones
13:         n.pherLevel ← n.pherLevel · 0.9
14:         n.nbrPhers ← n.nbrPhers · 0.9
15:     for each e in E do                    ▷ Evaporate edge pheromones
16:         e.pherLevel ← e.pherLevel · 0.9
17: end procedure
```

The main agent method, invoked in Algorithm 1 line 9, is Algorithm 2.

Algorithm 2 Step method executed by each agent to search for matching edges

```
1:  procedure step()
2:      if mode = 1 then                      ▷ at home in query, seeking peer in data
3:          if |location.peers| > 0 then
4:              select from peers by weight
5:              set location to selected peer
6:              append peer to history
7:              mode ← 2
8:          else
9:              deallocate agent
10:     else if mode = 2 then                 ▷ on peer in data, seeking neighbor in data
11:         if location has nonzero nbrPhers for label startNbr then
12:             select from neighbors with this label by pherLevel
13:             location ← selected neighbor
14:             append neighbor and traversed edge to history
15:             mode ← 3
16:         else
17:             deallocate agent
18:     else if mode = 3 then                 ▷ on neighbor in data, seeking peer in query
19:         if |location.peers| > 0 then
20:             select from peers by weight
21:             set location to selected peer
22:             append peer to history
23:             mode ← 4
24:         else
25:             deallocate agent
26:     else if mode = 4 then                 ▷ back in query, seeking start node
27:         if start is neighbor of location then
28:             append edge from location to start to history
29:             increment pherLevel on each node and edge in
                    history by 0.1
30:         deallocate agent
31: end procedure
```

4.3 Initialization

In initialization, ASSIST ingests the query and data graphs, finds nodes common to them both (a process we call "peering"), and initializes their pheromones.

Peering identifies nodes in one graph that match nodes in the other (matching subgraphs of size 1). This process corresponds to the node label filter used by [28], and is motivated by the observation that all the nodes in any shared subgraph must match between the graphs. Matching requires the peered nodes to have the same label, and (unless one of them has detail 0) also the same detail.

Peering considers each node in the query. We load the data graph as a tree organized according to our data model (Section 4.1), so data access time is logarithmic, for overall time complexity $O(q \cdot log(d))$.

In practice, peering a 100 node query against a 10^6 node data graph requires a median time of 38 ms, which exceeds the median match time of 6 ms for this scenario, but is not overwhelming.

Nodes in the query and data without peers are *pruned* (removed from **N**), and edges incident on them are removed from **E**.

Next, *pherLevel* ← 1.0 on all retained nodes, allowing them to initialize *nbrPhers*. Each time agents update pheromones on visited nodes (Algorithm 2 line 29, invoked in Algorithm 1 line 9), nodes update *nbrPhers* (Algorithm 1, line 11).

A node's *nbrPher* tells a resident agent which labels it can access from there. The weight of a peering between a query node and a data node is the cosine distance between their *nbrPhers*. The existence of a peering does not change over a run, but its weight does change as pheromones evaporate and accumulate.

4.4 Matching a Single Edge

Each agent starts on a peered node in the query and seeks a path corresponding to the dashed loop in Figure 2. A single circuit requires four ticks, advancing through the four values of *mode*. Line numbers reference Algorithm 2.

(1) It first moves from a node in the query to one of its peers in the data, selected from *peers* by *weight* (arrow 1, lines 3-7).
(2) It seeks a neighbor of the peer with label *nbrPher* (arrow 2, lines 11-15).
(3) From this node, it seeks a peer back in the query, and if successful, moves to it (arrow 3, lines 19-23).
(4) Finally, it seeks an edge back to its starting node (arrow 4, lines 27-30).

The moves between the two graphs (arrows 1 and 3) must match both label and detail, while arrow 2 (within the data) needs only match the label, and arrow 4 must arrive back at the starting node.

As it seeks such a circuit, the agent maintains a history of the nodes and edges it has visited. Exploring such a path takes constant time, and can be pursued by many agents in parallel (though our current implementation is serial). An agent that completes all four steps deposits pheromones on the nodes and edges it has visited (Algorithm 2, line 29). At the end of each tick, we evaporate the pheromone on all edges and nodes (Algorithm 1, lines 12-16). Thus nodes and edges that participate in successful circuits accumulate pheromone, attracting more agents in subsequent iterations, while the pheromone on others evaporates.

Matched nodes are a subset of peered nodes. Two nodes (one in the query, the other in the data) are *peered* if they describe the same entity, but they are *matched* only if they are peered *and* are part of a complete circuit (Figure 2). Similarly, two edges are matched if their endpoints are matched, which implies that they are both part of a successful circuit.

A single agent circuit identifies only a single shared edge, but many agents stochastically repeating this behavior can merge them into larger shared subgraphs. Figure 3 shows another circuit, identifying edge *D-E* as shared. Each of these circuits increases the pheromone strength of its endpoints and their shared edge. As node *E*'s pheromone increases with successive circuits, A's *nbrPher* augments the value for label *E*, just as successive circuits validating *A* and *B* increase the strength of label *A* in *E*'s *nbrPher*. As a result, the probability increases that agents visiting *A* will consider

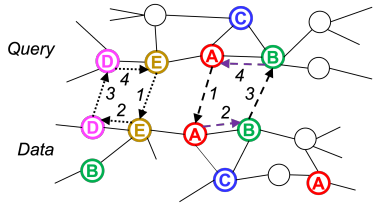

Figure 3: Merging matched edges into larger subgraphs

E as well as *B*, and agents visiting *E* will consider *A* as well as *D*, marking the edge *E-A* in each graph as matched and yielding a larger matched subgraph *D-E-A-B*. A similar process will discover the match between the edges *C-A* and *C-B* in the two graphs.

Repeated visits by many agents reinforce these pheromones, while pheromones on other edges and nodes evaporate, singling out matching nodes and edges. This incremental assembly of smaller subgraphs into larger ones illustrates the power of stigmergy to coordinate the independent efforts of individual agents.

4.5 Detecting Termination and Retrieving Results

ASSIST's matching process is continuous and emergent. For it to be useful in practice, we need to 1) measure its performance on problems of varying complexity, 2) tell when it has converged, and 3) retrieve the matching subgraphs it has found.

We illustrate with a scenario involving a kernel of size 10 embedded in a query of size 30 and a data graph of size 300.

We measure ASSIST's *performance* using the kernel. Because we know the kernel, we can tell when all of its nodes and edges have been identified, by comparing the edges traversed by successful agents in the query with the edges in the kernel. In the example given here, ASSIST discovered the kernel at tick 8. Even on more complex graphs, in general it is discovered in 20 or fewer ticks. For experimental purposes, most of our results report the time required to retrieve the kernel. But this technique does not address the second and third requirements, in cases where we do not know the identity of subgraphs in advance.

Convergence can be detected by the plateauing of various observables in the query, including the total number of matched edges and nodes discovered by successful agents.

Figures 4 and 6 show the same run, terminated when the number of matched edges has been stable for 20 ticks. The kernel is discovered at tick 8 (the vertical dashed line). The run discovers not only the 10-node, 16-edge kernel, but four more nodes and edges that are connected with it in the same way in the query and data.

Figure 4 shows the evolution of the number of matched edges. In general, the matched edges may belong to disjoint subgraphs, but in this case, all four are part of the extensions to the kernel shared by the query and the data. Three of the extra edges are detected after the kernel. Figure 5 shows the recovered graph.

The query has 28 peered nodes. Figure 6 shows, for each of these 28 nodes, a running average of the pheromone level on that node. On 14 nodes, the initial pheromone decays away. It accumulates, however, on 14 other nodes, including the ten nodes that form the kernel, clearly separating from the 14 peered but unmatched nodes (all superimposed on the bottom decaying line). In this run, the

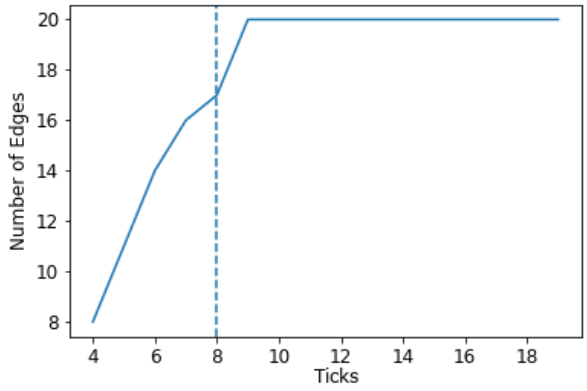

Figure 4: Matched edges by Tick

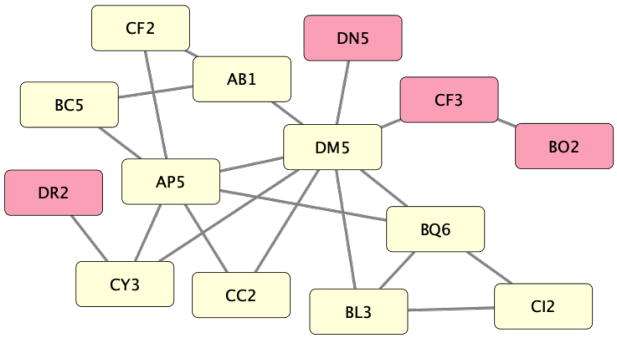

Figure 5: Common subgraph recovered by ASSIST. Yellow: original kernel. Red: other shared nodes and edges.

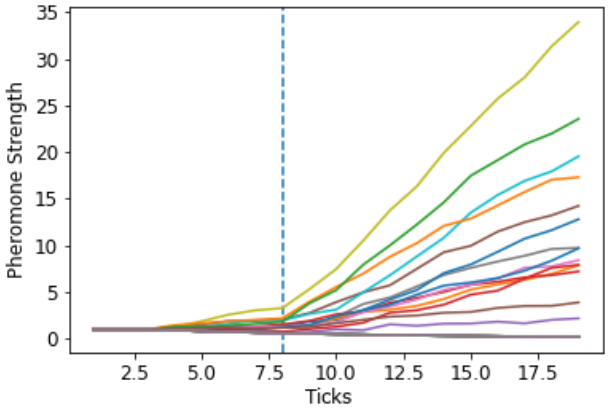

Figure 6: Running average of pheromone levels on query nodes

kernel is discovered soon after matched nodes begin to separate from unmatched ones.

Once the matched edges stabilize, we can *retrieve* the matched subgraph(s), using NetworkX's `from_pandas_edgelist()` function, whose time complexity is linear in the number of discovered edges. For some purposes, it is useful to have a rapid estimate of the largest subgraph discovered at a given point in the process, and we use Cichon's stochastic algorithm [7].

5 EXPERIMENTAL RESULTS

5.1 Dimensions of Interest

To demonstrate ASSIST, we explore the effect of several variables on the time required to match the kernel ("matching time"). Unless otherwise noted, match times are milliseconds (ms) on a MacBook Pro 18,1 with the Apple M1 Pro chip and 32 GB of RAM, running MacOS 26.1 (Tahoe), measured by calls to `System.currentTimeMillis()` from `java.lang.System`. Each run ends when the maximum number of matched edges has not changed for 10 ticks, and we report the time at which the kernel matched (which is often several ticks before matched edges plateaus).

Throughout these experiments, our results are medians over at least five runs with different random seeds, but on the same kernel, query, and data graph for each independent variable. Error bars show the upper and lower quartiles.

The space of interest is huge, and we report only a few results along the following dimensions to illustrate ASSIST's performance.

- Data and Query Size: Given the NP-complete nature of subgraph isomorphism, a primary result of interest is how peering and matching time varies with the size of the query and data graphs.
- Common Graph Size: We expect matching time to increase with the size of the largest common subgraph. As noted in Section 4.5, this may be larger than the kernel.
- Query Ambiguity: In our baseline experiments, within each graph, each node's label-detail combination is unique, and matches require matching both label and detail. We expect matching time to increase as we increase the proportion of nodes in the query with detail = 0.

5.2 Data and Query Size

We expect any algorithm to require more time to process larger query and data graphs. What is the shape of this dependency?

Peering reduces the size of the data graph to the number of peered nodes, which (with unambiguous node identifiers) is bounded by the size of the query, and may be smaller. For example, in one run with $(q, d) = (100, 100)$, the peer set is only 43 nodes. Thus we expect matching time to be independent of data size.

Figure 7 shows matching and peering time for a 100 node query as a function of data size d. The inset (for d from 100 to 10k nodes) shows that for $d > 3000$, matching time is basically flat, and the full graph shows that this holds for 10^5 and 10^6. Under these circumstances, peering dominates matching time for larger data.

Figure 8 shows kernel matching and peering times for $d = 4000$ as a function of query size q. The dependency in both cases is reasonably linear, though peering grows more slowly than matching. We expect this dependency. The larger q, the larger the set of peered nodes and edges that the agents need to explore.

We predicted peering time of $q \cdot log(d)$. Figure 8 shows the linear dependence on q, but the variation in Figure 7 is too high for a useful fit of the $log(d)$ component. Figure 9 shows that peering time for $q = d \in \{100, 1000, 2000, 3000, 4000, 5000, 6000, 7000, 8000, 9000, 10000\}$ is indeed linear in $q \cdot log(d)$.

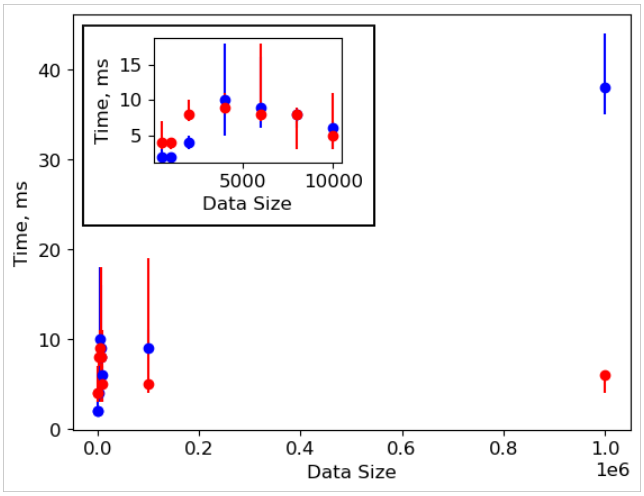

Figure 7: Matching (red) and peering (blue) time for 100 node query by data size

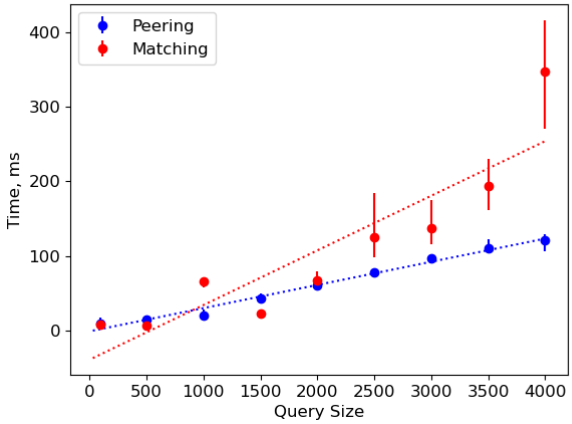

Figure 8: Kernel matching and peering time for 4000 node data by query size

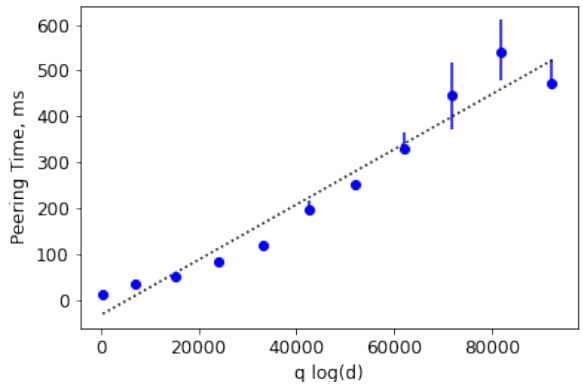

Figure 9: Peering time vs. $q \cdot log(d)$ for $q = d$

5.3 Common Graph Size

Matching time depends not only on query and data size, but also on the size of the discovered subgraph. The subgraph may be larger than the kernel, and we terminate our runs when the number of matched edges plateaus, in an effort to capture a largest subgraph. We estimate its size with the Cichon heuristic [7].

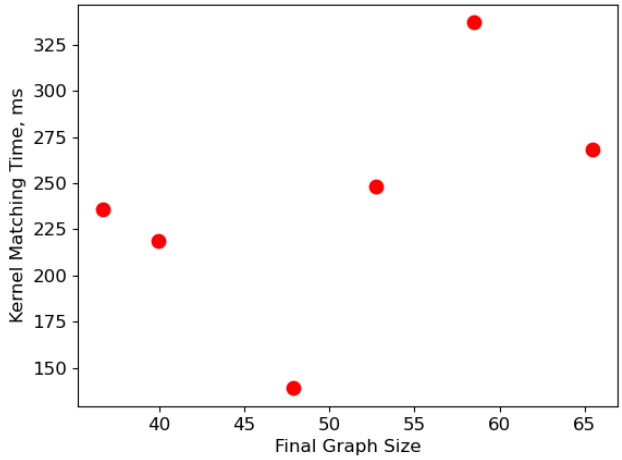

Figure 10: Kernel matching time by largest graph size, 4000 x 4000 scenario

Figure 10 shows matching time as a function of the overall size of the subgraph discovered at the time the kernel is matched. Each point is a single run, so there are no error bars. The general slope is positive, as expected, but with considerable variation.

5.4 Query Ambiguity

In many use cases, the query specifies the type of a node, but not its unique identity. Here we explore the effect of ignoring the details on some proportion of the query nodes, with a kernel of size 40, $q = 100, d = 6000$. When we ignore the detail of a node, we say that we "ablate" it. Ablation requires the algorithm to consider more peers in the data graph for each node in the query than would be needed if we used the detail, and finds subgraphs that satisfy the category (label) of an ablated query node even if the query does not specify the detail.

Figure 11 shows the impact of varying probabilities of ablation on two scenarios, one with label vocabulary 100 (the same size as other experiments reported here), the other with vocabulary of 10. In both cases, matching time is linear, though variation is inverse to vocabulary. Because the labels even without detail carry ten times more information with the larger vocabulary, the peered data graph has fewer nodes and edges than with the smaller vocabulary, and presents an easier search problem.

ASSIST also supports ablation of the data graph, allowing specific individuals identified in the query to match categories in the data. Space precludes presenting examples here.

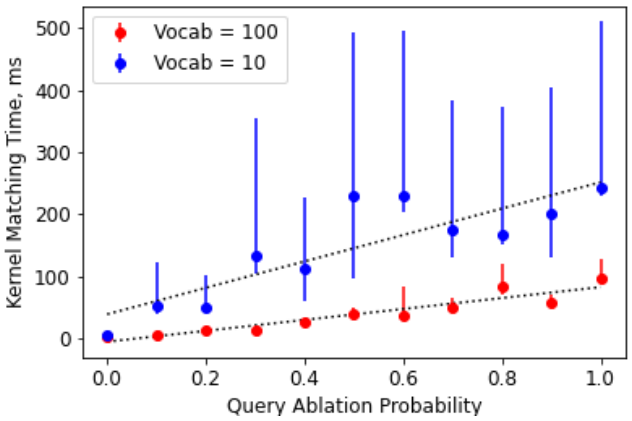

Figure 11: Match time under ablation. Upper curve: vocab = 10. Lower: vocab = 100

6 FUTURE WORK

These experiments show that ASSIST can find subgraphs in time $O(q \cdot log(d))$, but leave room for further study.

6.1 Parallelization

The current code is single-threaded. The highly parallel actions of multiple agents hold great promise for multithreading and reimplementation on a GPGPU.

6.2 Effect of Different Test Graphs

Evaluation of a stochastic algorithm like ASSIST is appropriately done with random sampling. In the experiments reported here, each scenario consists of a single triple of kernel, query, and data graphs. We run each scenario multiple times with distinct random seeds, varying aspects of the algorithm such as the order in which nodes are explored and the selection of neighbors to explore. But the graphs in the scenario are themselves randomly generated, and it would be worthwhile to expand the sampling, so that for a given kernel, query, and data graph size, multiple different random graphs are explored.

6.3 Other Random Graph Models

Our baseline experiments are with Barabási-Albert (BA) graphs, in which node degree follows a power law [3]. Graphs with this structure are common in networks of associations, such as social networks or webs of internet sites, and are of great interest in many potential applications of graph matching. But there are other graph models with different characteristics, including Erdős -Rényi graphs [12] and Watts-Strogatz (small world) graphs [43]. These models differ in characteristics such as distribution of node degree, average path length, and clustering coefficient. We plan to explore the performance of ASSIST on these and other models.

6.4 More Complex Matches

Figure 2 shows the basic matching mechanism. Some problems have additional complexity. For example:

(1) Node labels might not match exactly. A data graph might a node "bank," while the query has "financial institution." Maintaining an ontology to allow such abstractions is not difficult, but the simple matching mechanism described above would miss the match.
(2) The graph might be directed. For example, in a graph of financial transactions, the edges indicate transactions, and the movement of money from A to B is not the same as movement from B to A. This directedness imposes a time ordering on the edges, which may be recorded either as clock time or as a partial order over the edges.
(3) We may want to allow matches in which a node or edge might be missing entirely in either the query (due to oversight by the analyst) or the data (due to the vagaries of data collection).

These complications frustrate many other subgraph algorithms, but straightforward extensions to ASSIST can accommodate them.

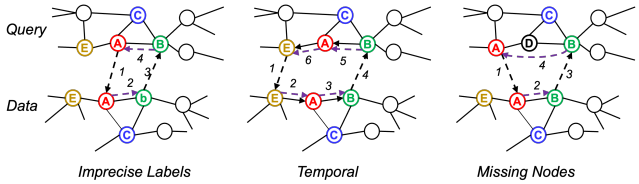

Figure 12: Extensions of ASSIST to more complex matches

The query in Figure 2, repeated in Figure 12 ("Imprecise"), handles case 1 (here, matching 'B' with 'b') by having agents consult an ontology in case of mismatch to see if one of the nodes subsumes the other. The total pheromone deposited for an imprecise match will be less than that for an exact match.

Figure 12 ("Temporal") handles temporal matches by requiring the agent to traverse two edges in the data graph before returning to the query, remembering the sequence of these edges, and then seeking a sequence of edges in the query with the same order to return home.

Figure 12 ("Missing") handles missing data by propagating neighbor pheromones across multiple edges, rather than simply sampling adjacent neighbors as in the present implementation. An agent can then sense the presence of an otherwise desirable node not immediately adjacent to its current node and move to it. In this case, as in the case of imprecise data, the pheromone deposited at the end of the circuit will be less than in the case of a perfect match.

7 CONCLUSION

ASSIST, a swarming stigmergic algorithm, offers an extremely fast heuristic for subgraph isomorphism. After initial peering (which requires time $O(q \cdot log(d))$), matching is linear in query size and constant in data size, much faster than the best previous heuristics, which are quadratic in the number of nodes. In addition, it allows approximate matches, in which a query that specifies only a node's category can retrieve subgraphs that match specific individuals in that category from the data.

In addition to advancing the state of subgraph isomorphism, ASSIST provides a pattern for how stigmergic reasoning can efficiently integrate results produced by multiple agents who are working independently on separate parts of a complex problem.

REFERENCES

[1] US Army. 2006. *Army Lessons Learned Program (ALLP).* Technical Report AR 11-33. US Army.

[2] Heitor T. de Azambuja, Nadia Nedjah, and Luica de Macedo Mourelle. 2024. Automatic Routing of Printed Circuit Board Traces Using Ant Colony Optimization Algorithm. In *2024 IEEE Latin American Conference on Computational Intelligence (LA-CCI).* IEEE, 1–6.

[3] Albert-László Barabási and Réka Albert. 1999. Emergence of Scaling in Random Networks. *Science* 286, 5439 (1999), 509–512. doi:10.1126/science.286.5439.509 arXiv:https://www.science.org/doi/pdf/10.1126/science.286.5439.509

[4] Endika Bengoetxea. 2002. *Inexact Graph Matching Using Estimation of Distribution Algorithms.* Ph. D. Dissertation. Ecole Nationale Supérieure des Télécommunications.

[5] Eric Bonabeau, Florian Henaux, Sylvain Guérin, Dominique Snyers, Pascale Kuntz, and Guy Theraulaz. 1998. Routing in Telecommunications Networks with "Smart" Ant-Like Agents. In *Second International Workshop on Intelligent Agents for Telecommunications Applications (IATA98)*, Vol. Lecture Notes in AI, 1437. Springer, 60–71.

[6] Horst Bunke. 2000. Graph matching: Theoretical foundations, algorithms, and applications. In *Proc. Vision Interface 2000.* 82–88.

[7] Jacek Cichoń, Jakub Lemiesz, and Marcin Zawada. 2011. On Cardinality Estimation Protocols for Wireless Sensor Networks. In *Ad-hoc, Mobile, and Wireless Networks*, Hannes Frey, Xu Li, and Stefan Ruehrup (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 322–331.

[8] Donatello Conte, Pasquale Foggia, Carlo Sansone, and Mario Vento. 2004. Thirty Years of Graph Matching in Pattern Recognition. *International Journal of Pattern Recognition and Artificial Intelligence* 18 (2004), 265–298.

[9] Andrew D. J. Cross, Richard C. Wilson, and Edwin R. Hancock. 1996. Genetic Search for Structural Matching. In *Proceedings of the 4th European Conference on Computer Vision-Volume I - Volume I.* Springer-Verlag, 648894, 514–525.

[10] DARPA. 2025. A3ML: Anticipatory and Adaptive Anti-Money Laundering. https://www.darpa.mil/research/programs/a3ml-anticipatory-adaptive. Accessed on 2025-12-19.

[11] DOE. 1999. The DOE Corporate Lessons Learned Program. http://energy.gov/sites/prod/files/2013/06/f2/std750199.pdf

[12] Paul Erdős and Alfréd Rényi. 1959. On Random Graphs. *Publicationes Mathematicae Debrecen* 6 (1959), 290–297.

[13] Hiba Fareed, Isam Alobaidi, Jennifer Leopold, Layth Almashhadani, and Nathan Eloe. 2024. Graph Mining Healthcare Approach Analysis and Recommendation-Copy. *Polibits* 66 (10 2024), 9–17. doi:10.17562/PB-66(1)-2

[14] Brian Gallagher. 2006. Matching Structure and Semantics: A Survey on Graph-Based Pattern Matching. In *AAAI Fall Symposium.* AAAI.

[15] Luca Maria Gambardella, Andrea-Emilio Rizzoli, Fabrizio Oliverio, Norman Casagrande, Alberto Donati, Roberto Montemanni, and Enzo Lucibello. [n. d.]. Ant Colony Optimization for Vehicle Routing in Advanced Logistics Systems. In *International Workshop on Modelling and Applied Simulation (MAS 2003)*, A.G. Bruzzone and R. Mosca (Eds.). DIP, 3.

[16] Apeksha Godiyal, Michael Garland, and John Hart. 2010. Enhancing Network Traffic Visualization by Graph Pattern Analysis. (01 2010). https://www.researchgate.net/publication/228772551_Enhancing_Network_Traffic_Visualization_by_Graph_Pattern_Analysis

[17] Pierre-Paul Grassé. 1959. La Reconstruction du nid et les Coordinations Inter-Individuelles chez Bellicositermes Natalensis et Cubitermes sp. La théorie de la Stigmergie: Essai d'interprétation du Comportement des Termites Constructeurs. *Insectes Sociaux* 6 (1959), 41–84.

[18] Geoff Gross, Rakesh Nagi, and Kedar Sambhoos. 2014. A fuzzy graph matching approach in intelligence analysis and maintenance of continuous situational awareness. *Information Fusion* 18 (2014), 43–61.

[19] Aric A Hagberg, Daniel A Schult, and Pieter J Swart. 2008. Exploring network structure, dynamics, and function using NetworkX. In *Proceedings of the 7th Python in Science Conference (SciPy2008)*, Gaël Varoquaux, Travis Vaught, and Jarrod Millman (Eds.). Pasadena, CA USA, 11–15.

[20] Jacqueline R. Henningsen. 2010. *Air Force Lessons Learned Program.* Technical Report AFI 90-1601. Dept of the Air Force.

[21] Richards J. Heuer, Jr. and Randolph H. Pherson. 2010. *Structured Analytic Techniques for Intelligence Analysis.* CQ Press, Washington, DC.

[22] Martin Heusse, Sylvain Guérin, Dominique Snyers, and Pascale Kuntz. 1998. Adaptive Agent-Driven Routing and Load Balancing in Communication Networks. *Advances in Complex Systems* 1 (1998), 234–257.

[23] JALLC. 2011. *The NATO Lessons Learned Handbook* (2nd ed.). NATO Joint Analysis and Lessons Learned Centre (JALLC), Lisbon, Portugal.

[24] Hui Jiang and Chong-Wah Ngo. 2004. Graph based image matching. In *Proceedings of the 17th International Conference on Pattern Recognition, 2004. ICPR 2004.*, Vol. 3. 658–661 Vol.3. doi:10.1109/ICPR.2004.1334615

[25] Megan Leierzapf and Julian Rrushi. 2017. Network forensic analysis of electrical substation automation traffic. In *IFIP Advances in Information and Communication Technology (Critical Infrastructure Protection XI, Vol. AICT-512)*, Mason Rice and Sujeet Shenoi (Eds.). Springer International Publishing, Arlington, VA, United States, 63–78. doi:10.1007/978-3-319-70395-4_4

[26] Marius Dan Leordeanu. 2009. *Spectral Graph Matching, Learning, and Inference for Computer Vision.* Ph. D. Dissertation.

[27] Yi Lu, Yaran Chen, Dongbin Zhao, Bao Liu, Zhichao Lai, and Jianxin Chen. 2021. CNN-G: Convolutional Neural Network Combined With Graph for Image Segmentation With Theoretical Analysis. *IEEE Transactions on Cognitive and Developmental Systems* 13, 3 (2021), 631–644. doi:10.1109/TCDS.2020.2998497

[28] Jacob D. Moorman, Thomas K. Tu, Qinyi. Chen, Xie He, and Andrea L. Bertozzi. 2021. Subgraph Matching on Multiplex Networks. *IEEE Transactions on Network Science and Engineering* 8, 2 (2021), 1367–1384.

[29] Frank Neumann, Dirk Sudholt, and Carsten Witt. 2009. *Computational Complexity of Ant Colony Optimization and Its Hybridization with Local Search.* Springer Berlin Heidelberg, Berlin, Heidelberg, 91–120.

[30] Michael J. North, Nicholson T. Collier, Jonathan Ozik, Eric R. Tatara, Charles M. Macal, Mark Bragen, and Pam Sydelko. 2013. Complex adaptive systems modeling with Repast Simphony. *Complex Adaptive Systems Modeling* 1, 1 (2013), 3. doi:10.1186/2194-3206-1-3

[31] H. Van Dyke Parunak. 2007. Real-Time Agent Characterization and Prediction. In *International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS'07), Industrial Track.* ACM, 1421–1428.

[32] H. Van Dyke Parunak. 2013. Dynamic Data Relevance Estimation by Exploring Models (D2REEM). In *8th International Conference on Semantic Technologies for Intelligence, Defense, and Security (STIDS 2013).* George Mason University.

[33] H. Van Dyke Parunak, Sven Brueckner, Liz Downs, and Laura Sappelsa. [n. d.]. Swarming Estimation of Realistic Mental Models. In *Thirteenth Workshop on Multi-Agent Based Simulation (MABS 2012, at AAMAS 2012)*, F. Giardini and F. Amblard (Eds.), Vol. LNAI 7838. Springer, 43–55.

[34] John W. Raymond and Peter Willett. 2002. Maximum common subgraph isomorphism algorithms for the matching of chemical structures. *Journal of Computer-Aided Molecular Design* 16, 7 (2002), 521–533.

[35] Kedar Sambhoos, James Llinas, and Eric Little. 2008. Graphical Methods for Real-Time Fusion and Estimation with Soft Message Data. In *11th International Conference of Information Fusion (FUSION 2008).* 1621–1628.

[36] Laura Sappelsa, H. Van Dyke Parunak, and Sven Brueckner. 2014. The Generic Narrative Space Model as an Intelligence Analysis Tool. *American Intelligence Journal* 31, 2 (2014), 69–78.

[37] Thomas Stuetzle and Marco Dorigo. 1999. *ACO Algorithms for the Traveling Salesman Problem.* Technical Report IRIDIA/99-3. IRIDIA, Université Libre de Bruxelles.

[38] Yuanyuan Tian, Richard C. McEachin, Carlos Santos, David J. States, and Jignesh M. Patel. 2007. SAGA: a subgraph matching tool for biological graphs. *Bioinformatics* 23, 2 (2007), 232–239.

[39] Hanghang Tong, Christos Faloutsos, Brian Gallagher, and Tina Eliassi-Rad. 2007. Fast best-effort pattern matching in large attributed graphs. In *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining.* ACM, 1281271, 737–746.

[40] Julian R. Ullmann. 1976. An Algorithm for Subgraph Isomorphism. *J. ACM* 23, 1 (1976), 31–42.

[41] USMC. 2013. Marine Corps Center for Lessons Learned. http://www.mccll.usmc.mil/

[42] Yuan-Kai Wang, Kuo-Chin Fan, and Jorng-Tzong Horng. 1997. Genetic-based search for error-correcting graph isomorphism. *Trans. Sys. Man Cyber. Part B* 27, 4 (1997), 588–597.

[43] Duncan J. Watts and Steven H. Strogatz. 1998. Collective dynamics of 'small-world' networks. *Nature* 393, 6684 (1998), 440–442.

[44] Richard C. Wilson and Ping Zhu. 2008. A study of graph spectra for comparing graphs and trees. *Pattern Recogn.* 41, 9 (2008), 2833–2841.

[45] Zhichen Zeng, Boxin Du, Si Zhang, Yinglong Xia, Zhining Liu, and Hanghang Tong. 2024. Hierarchical multi-marginal optimal transport for network alignment. In *Proceedings of the Thirty-Eighth AAAI Conference on Artificial Intelligence and Thirty-Sixth Conference on Innovative Applications of Artificial Intelligence and Fourteenth Symposium on Educational Advances in Artificial Intelligence*, Vol. 38. AAAI Press, Article 1857.