AGILE Program Workflows

Purpose

The purpose of this document is to describe the four AGILE Program workflows, plus their derived kernels, and three industry standard benchmarks that will drive the research and development of the AGILE system designs. This document includes metric tables along with links that provide additional information related to specific metrics. These links are for descriptive purposes only and while each workflow will include the general computation, it may not employ the specific methods or approaches discussed in the link. Performers are always free to implement the general computation as best suited for their architecture. However, these links should help when the proposer determines the estimated performance for each entry in the metric tables.

Introduction

Today's explosive data growth has ushered in a new generation of HPC applications that transform massive, unstructured, heterogeneous data streams into actionable knowledge. Data is increasing exponentially in volume, velocity, variety, and complexity. It is often sparse and stored in structures with poor data locality. In many scenarios, data arrives continuously and must be processed and stored concurrently with analysis. Processes that ingest, transform, and store data (ITS) can no longer be ignored as in many scenarios they account for a significant fraction of the overall execution time and consume a significant portion of machine resources.

Given the nature of data analytic applications, benchmarks such as LINPACK, Graph500, and MLBench – which measure individual computations in insolation without consideration of ITS processes, the reorganization and movement of data between computational components, and the reporting of results – cannot measure the true performance and scalability of real-world analytic applications. Consequently, AGILE has adopted a three-tier, holistic evaluation process to measure the performance and scalability of proposed hardware and software system designs. The three tiers are: end-to-end workflows, stand-alone kernels of critical workflow components, and industry standard benchmarks.

The top tier comprises four end-to-end workflows in subject matters important to the IC:

1) Knowledge Graphs [1]
2) System and Event Pattern Detection [2]
3) Classification of Streaming Data [3]
4) Network and Cyber-physical Systems [4]

Each workflow implements a specific scenario from input to output and may have multiple modes of operations. For example, the System and Event Pattern Detection workflow has three modes returning, respectively, exact pattern matches, approximate pattern matches, and partial matches of sub patterns as new data is added to the world graph. For each workflow we will release a problem statement, task graph, reference implementation in a high-level programming language, and input/output files at different scales. For each workflow, a test program will be provided.

The workflows comprise different types of computations and ingest both static and streaming data sets. For example, the Knowledge Graph workflow includes both machine learning and graph methods. The Classification of Streaming Data consumes multiple data streams and requires that the data and processing rates are commensurate, i.e., no data packets are dropped. The Network and Cyber-physical Systems workflow implements a game playing scenario in which red and blue team moves and reconfiguration of the system must occur within strict time limits. The workflows will require performers to demonstrate the performance of different data structures, parallel computing constructs, and synchronization operations.

The second tier are subsets of the workflow task graphs, referred to as *kernels,* that measure the key rates listed in the Workflow Metric Tables for each subject matter area. Measuring a rate may require execution of more than one task; for example, Kernel 1 of Workflow 1 comprises four tasks (see Figure 2). And a particular task may be used in measuring more than one rate; for example, the Graph Neural Network Model task is used in Kernels 2, 3, and 4 of Workflow 1 (see Figures 2, 3, and 4). We will release complete codes and data sets for each kernel. We will also provide a test program for each code.

Since it may be challenging for performers to execute an entire workflow or kernel in the early phases of the project, we will release reference implementations for some individual workflow tasks, referred to as *derived kernels,* their inputs and outputs, and the organization of the data at the conclusion of the prior task in the workflow. Any reported execution time for a derived kernel must include the time the code takes to reorganize or move data between the conclusion of the prior task to the start of the derived kernel under consideration. No free lunch!!!

Finally, the AGILE program will use three industry standard benchmarks:

- Breadth First Search (BFS) - https://graph500.org/
- Triangle Counting - http://graphchallenge.mit.edu/
- Jaccard Coefficients - https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=7529958

These three kernels (benchmarks) provide performers a quick start and a way to compare the performance and scalability of their novel designs relative to current systems. Table 1 lists the goal for AGILE system designs for each kernel at different scales. This table appears in Section A.2.3 of the BAA as Table A.2.3-5.

Benchmark Metric	**Target Metrics**		
	Scale 36	**Scale 42**	**Report**
Breadth First Search (BFS)	1 GTEPS/core	1 GTEPS/core	Traversed edges per second (TEPS)
Triangle Counting	1 M triangles/sec/core	1 M triangles/sec/core	Number of triangles & number of 2-paths

Benchmark Metric	Target Metrics		
	Scale 36	Scale 42	Report
Jaccard Coefficients	1 M coefficients/sec/core	1 M coefficients/sec/core	Number of Jaccard Coefficients & number of Jaccard Coefficients per second
GTEPS = 10^9 traversed edges per second M = 10^9			

Table 1 – Industry standard benchmark metrics

We will release links to reference implementations and standard data sets. Please see the Industry Standard Benchmark document for more information.

WORKFLOW 1 – Knowledge – Groups, Relationships, & Interests
Three important questions asked of Knowledge Graphs are: vertex classification, link prediction, and multi-hop reasoning. A formal problem definition is:

Let **G** = *(V, E, C_V, E_V)* be a property graph where *V* is a set of vertices, *E* is a set of edges, C_V is the set of vertex property labels, and C_E is the set of edge property labels. Compute embeddings for *V* and *E,* and then train a graph neural network model **Γ** for **G,** and models **Γ'** and **Γ"** such that

- **Γ':** maps a vertex in *V* to a label in C_V.
- **Γ":** maps two vertices in *V* to a label in C_E.

Then solve separately the following three problems:

1) *Vertex Classification:* Given an unlabeled vertex *s* in *V,* return **Γ'** *(s).* [5]
2) *Link Prediction:* Given vertices *s* and *t* in *V,* return **Γ"** *(s, t).* [6]
3) *Multi-hop Reasoning:* Given vertices *s* and *t* in *V* and an edge label *p* in C_E, return the *K* best paths in **G** from s to t with length less than L_{max} as scored by **Γ.** [7]

Table 2 lists the current rates and AGILE targets for constructing the knowledge graph, computing a model of the graph, and the time to classify vertices, predict relationships, and return the best paths between two vertices. Current rates are for knowledge graphs of ten billion records whereas the AGILE target is for 1 trillion records (100x current size). This table appears in Section A.2.3 of the BAA as Table A.2.3-1.

The first row (kernel) of the table measures the time to read data records, transform the raw data, resolve vertex and edge ambiguities, and build-out the common internal data structures used by downstream tasks. Row 2 measures the time to compute vertex and edge embeddings and to learn graph neural network (GNN) models for vertex classification, edge prediction, and to score paths [8].

The third and fourth rows of the table measure the time to classify vertices and edges and the fifth row measures the time to return the best paths given two vertices in the graph and an edge type. The time reported includes the time to learn the specific GNN model for the problem.

Metric	Today	AGILE Target
Data ingestion rate	0.1 G data-elements per second from a Single source, single data type	10 G data-elements per second from 3 or more sources and data types (100x faster for 3 sources)
Time to learn embedding (Graph Size > 1 PB)	1,440 minutes	30 minutes (48x faster)
Time to classify vertices and edges	> 1,440 minutes	30 minutes (> 48x faster)
Time to predict and infer new relationship	> 1,440 minutes	30 minutes (> 48x faster)
Time to reason about higher-order relationships using multi-hop reasoning	1 – 2 hops (exact matches) in 30 minutes	3 – 5 hops (approximate/fuzzy matches) in 1 minute (30x faster)

Table 2 – Metric table for Workflow 1

Figure 1 depicts a task graph for Workflow 1 and Figures 2 – 5 map the components of the workflow to the rows of metric tables (the kernels). Data records are read from a formatted file, processed, and stored in internal data structures. If an edge record references an unknown vertex, the new vertex is processed and stored. A graph is then constructed from the data structures, which is passed to downstream tasks that compute embeddings and models, classify vertices, predict edge labels, and perform multi-hop reasoning.

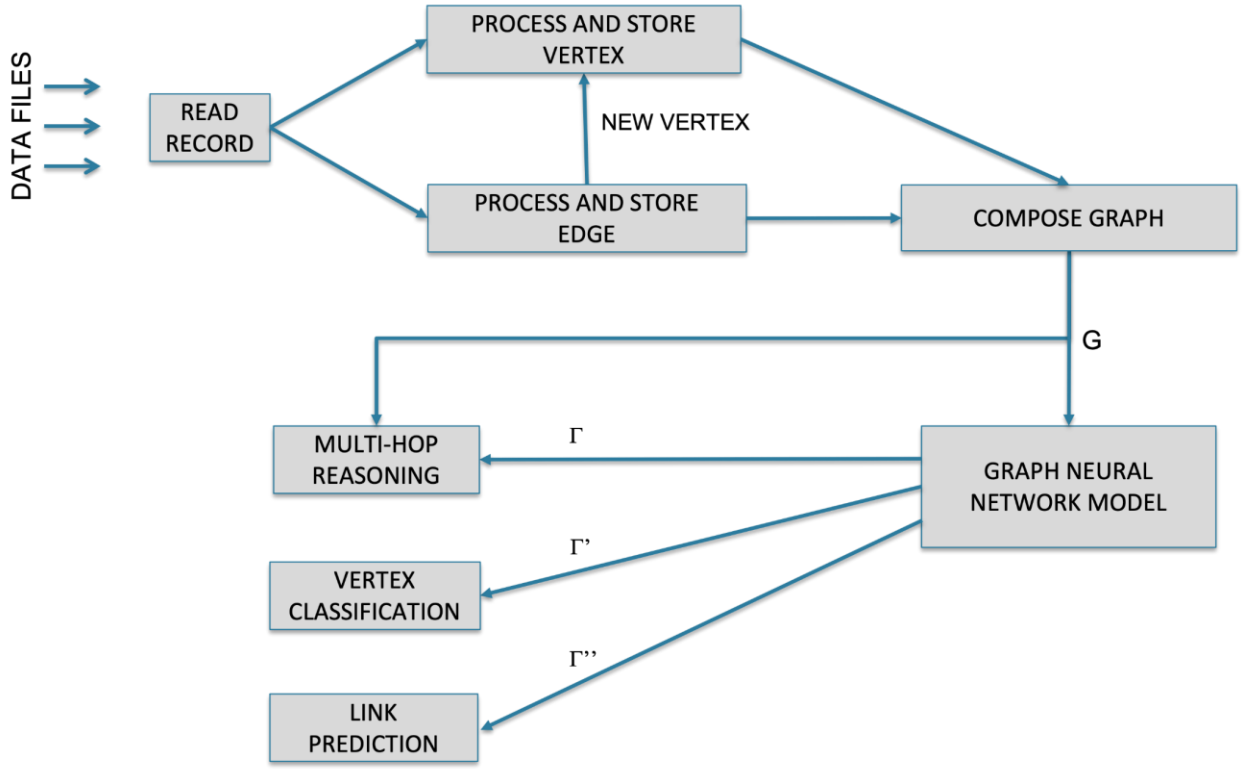


Figure 1 – A task graph for Workflow 1

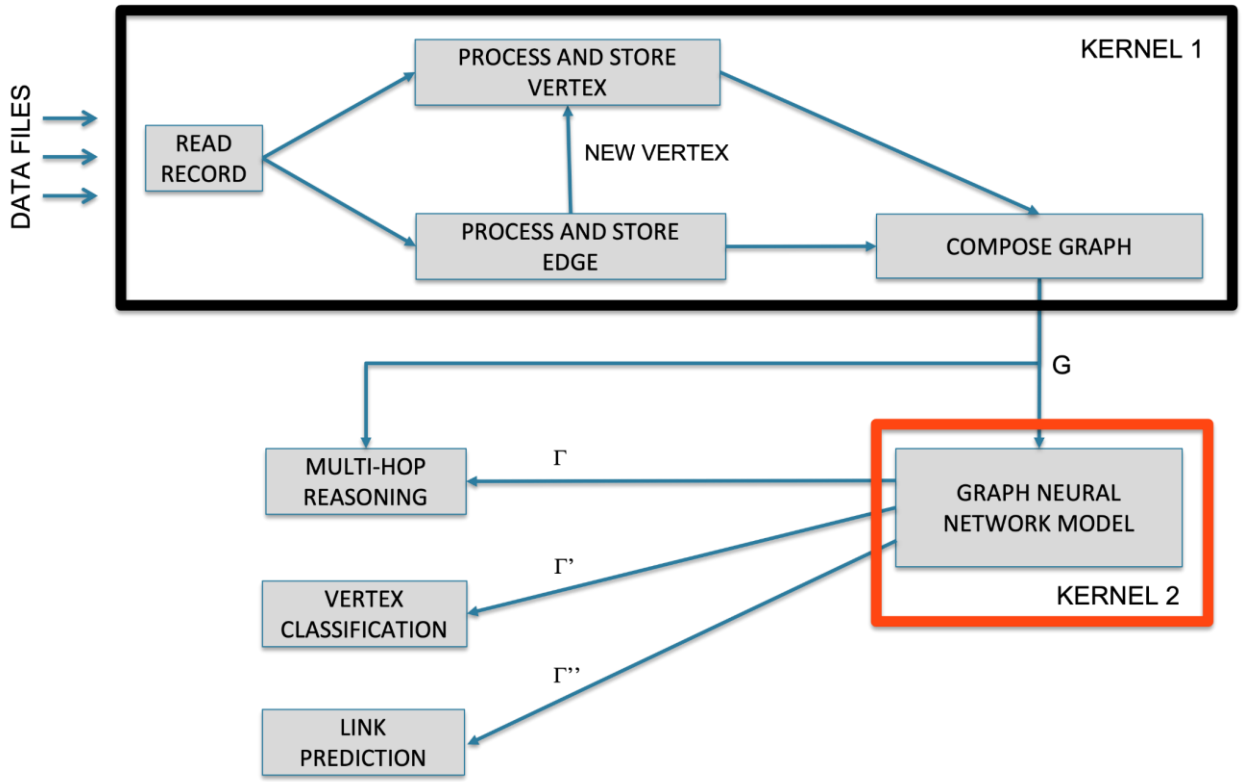


Figure 2 – Workflow 1, Kernels 1 and 2

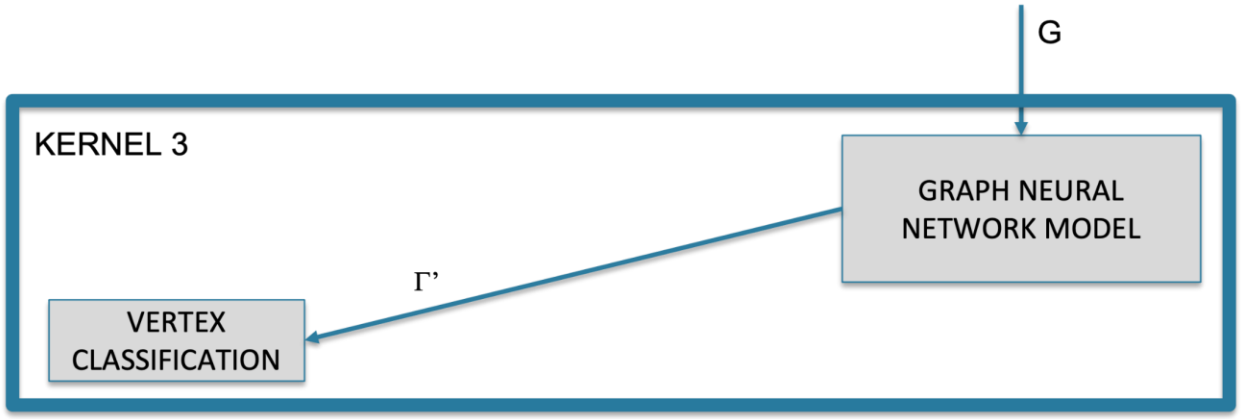


Figure 3 – Workflow 1, Kernel 2

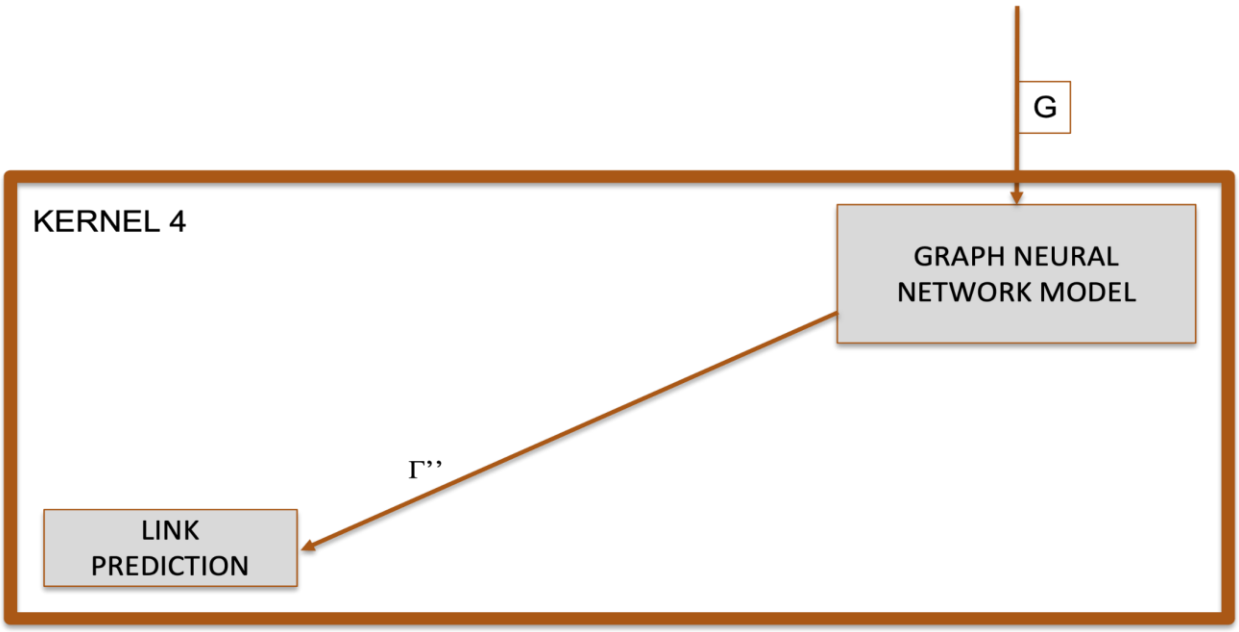


Figure 4 – Workflow 1, Kernel 4

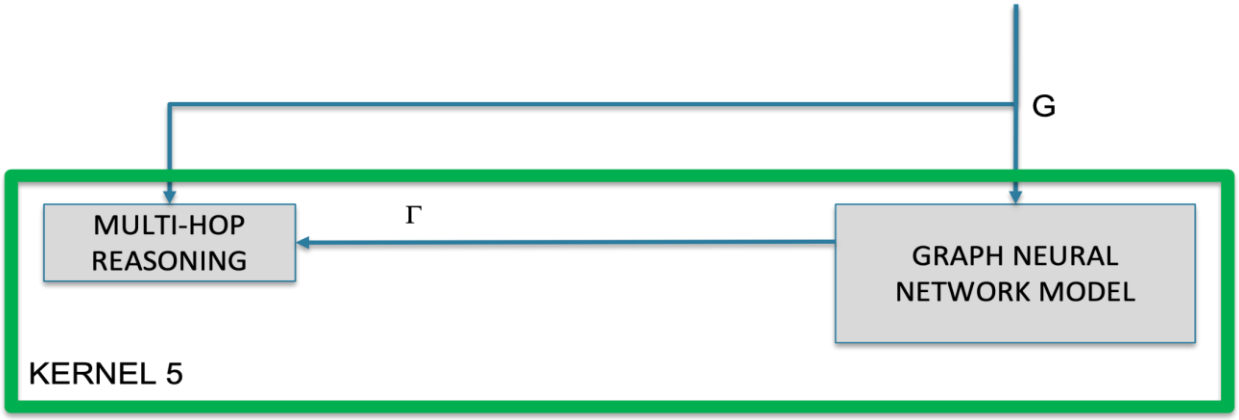


Figure 5 – Workflow 1, Kernel 5

A possible task graph implementing multi-hop reasoning is shown in Figure 6. Given two vertices, *S* and *T*, and a relationship *P*, return the best *K* paths from *S* to *T* as measured by the GNN model **Γ.** Partial candidate paths from *S* to *T* are stored in the vector PATHS. Initially, the vector has a single candidate path {S}. For every candidate path of length less than L_{max}, the neighbors of the last vertex on the path are examined. If the vertex is already on the path, the vertex is ignored (no loops) and the next neighbor is examined. If the neighbor is *T*, the path has reached the terminator and it is pushed to the array variable RESULTS. If the neighbor is not *T*, the path is scored using **Γ,** and the path and its score are pushed to SCORES. After all neighbors are examined, SCORES is sorted in descending order by the scores and the top B highest scoring paths are pushed to PATHS to be processed on the next iteration. An implementation of the computation written in C++ using STL data structures is shown in Figure 7.

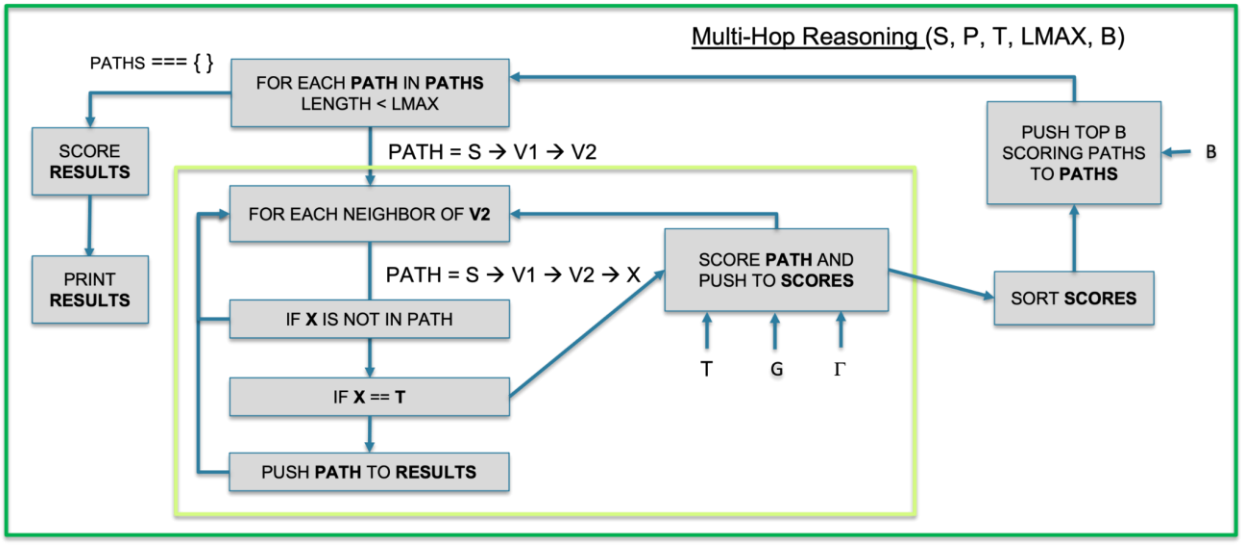


Figure 6 – Task graph for multi-hop reasoning

```
using vertexVec = std::vector <vertex>;
using paths_t   = std::vector <vertexVec>;
using scorePair = std::pair <double, vertexVec>;


bool greaterScore(scorePair & a, scorePair & b) { return a.first > b.first; }


Results_t GreedyReasoning(Params_t params, Embeddings_t embeddings, Graph_t graph) {
  vertex S = params.S;
  vertex T = params.T;
  paths_t results = [];
  paths_t new_paths = [ ];
  paths_t old_paths = [ [S] ];


  while (old_paths.size() > 0) {


    for (vertexVec & path : old_paths) {
      if (path.size() == params.max_path_length) continue; // reached maximum length


      vertex last = path[-1];
      std::vector < std::pair<double, vertexVec> > scores;


      for (vertex x : Neighbors(last)) {
        if (path.find(x) != path.end()) continue;           // x is already in path


        vertexVex cand_path = path;
        cand_path.push_back(x);


        if (x == T) {results.push_back(path); continue;}    // t reached, push to results


        double score = computeScore(path, T, embeddings, graph);
        scores.push_back( std::make_pair(score, path) );
      }


      std::sort(scores.begin(), scores.end(), greaterScore);
```

```
        if (scores.size() > params.B) scores.resize(b);          // keep best B paths
        for (auto score : scores) new_paths.push_back(score.second);
      }

      std::swap(new_paths, old_paths);
    }

    // score results from best to worse
    // if (results.size > params.N) results.resize(params.N);     // keep best K results
    // print results
}
```

Figure 7 – A C++ implementation for ranking multi-hop reasoning paths

WORKFLOW 2 – Detection - System and Event Patterns

Workflow 2 performs exact, approximate, and partial matching of a pattern graph against a world graph. A formal problem statement is:

Let **G** = (V, E, C_V, E_V) be a property graph where V is a set of vertices, E is a set of edges, C_V is the set of vertex property labels, and C_E is the set of edge property labels. Let **P** be a pattern graph and let $\{T_1, T_2, \ldots T_K\}$ be K subgraphs of **P** such that their union is **P.** Then solve separately, the following three problems:

1) *Exact Matching:* Find all instances of **P** in **G.** [9]
2) *Approximate Matching:* Return the N closest matches of **P** in **G** as measured by some graph edit function. [9]
3) *Partial Matching:* As new data is added to **G,** alert when a subgraph T_i appears in **G.** [10]

Table 3 lists the current rates and AGILE targets for the kernels of Workflow 2. This table appears in Section A.2.3 of the BAA as Table A.2.3-2. Current rates are for graphs of ten billion records whereas the AGILE target is for 1 trillion records (100x current size). The first kernel, data ingestion rate, measures the time to read data records, transform the raw data, resolve vertex and edge ambiguities and build-out the common internal data structures used by downstream tasks. The second kernel measures the rate at which streaming data records can be read, their data transformed, vertex and edge ambiguities resolved and added to the common internal data structures used by downstream tasks. This kernel must run concurrently with downstream tasks requiring those tasks to work correctly when data is being inserted, deleted, or modified.

Rows 3 and 4 measure the time to complete partial, exact, and approximate pattern matching. Given a data graph **G,** pattern graph **P,** and a set of subgraphs of **P,** Kernel 3 measures the time to identify the emergence of sub patterns and combinations of sub patterns of **P** in the data graph as new data are added. The exact and approximate kernels process a static graph and return all exact matchings of **P** and the top K approximate matches of **P** as scored by some graph edit function.

The fifth row measures the time for partial, exact, and approximate matching of more complex patterns comprising multiple time and location features.

Metric	Today	AGILE Target
Size of graph	0.01 PB	10 PB (1000x larger)
Data ingestion rate	0.1 G data-elements per second from a single source, single data type	10 G data-elements per second from a three or more sources and data types (100x faster for 3 sources)
Insert/Delete/Modify rate for vertices and edges	0.01 G *edits* / second (batched)	10 G *edits* / second (continuous) (1000x faster)
Pattern Detection per minute	Single event, linear paths, exact match	Multiple events, branches, prioritized approximate/fuzzy matching
Incremental analysis	NOT DONE	Commensurate with data rate
Time to complete multiple day / multiple location queries	NOT DONE	Completed in minutes

Table 3 – Metric Table for Workflow 2

Figure 8 depicts a task graph for Workflow 2 and Figures 9 and 10 map the rows of Table 3 to kernels. Data records are read from a formatted file, processed, and stored in internal data structures. If an edge record references an unknown vertex, the new vertex is processed and stored. A world graph is then constructed from the data structures against which the *Exact Matching* and *Approximate Matching* task (the third row of boxes) compute matches of the pattern graph **P** against **G.** For approximate matching, we employ an iterative belief propagation method that optimally associates of vertices and edges in **G** with those in **P.**

The bottom set of boxes depicts a streaming data scenario in which data records are continuously read, processed, and added to the world graph. The *Partial Matching* task runs concurrently admitting alerts whenever a subgraph T_i of **P** emerges in **G'.**

Figure 11 depicts a possible task graph for *Exact Matching*. Let the pattern graph **P** be expressed as a linear sequence of terms and filters as shown in Figure 12, then *Exact Match* processes the terms in order by recursively calling *Process Term*. Upon finding an instance of the final term that satisfies the term's filter, RESULT holds a match of **P.**

Figure 12 shows a pattern graph of three netflow edges and an equivalent expression of three terms and filters. The initial call to *Process Term* will consider each netflow edge in **G** and recursively call *Process Term* for every edge with source port equal to **5.** Say edge

{src = **v1,** dst = **v2,** src_port = **5,** time = **22.5**}

is one such edge. The recursive call will consider each netflow edge with source node **v2** and recursively call *Process Term* for each edge with source port equal to **5** and time greater than **22.5.** Say edge

{src = **v2,** dst = **v3,** src_port = **5,** time = **23.8**}

is one such edge. The third recursive call will consider each netflow edge with source node **v1** and destination node **v3.** Any edge with source port equal to **8** and time greater than **22.5** and less than **23.8,** completes a match of **P.**

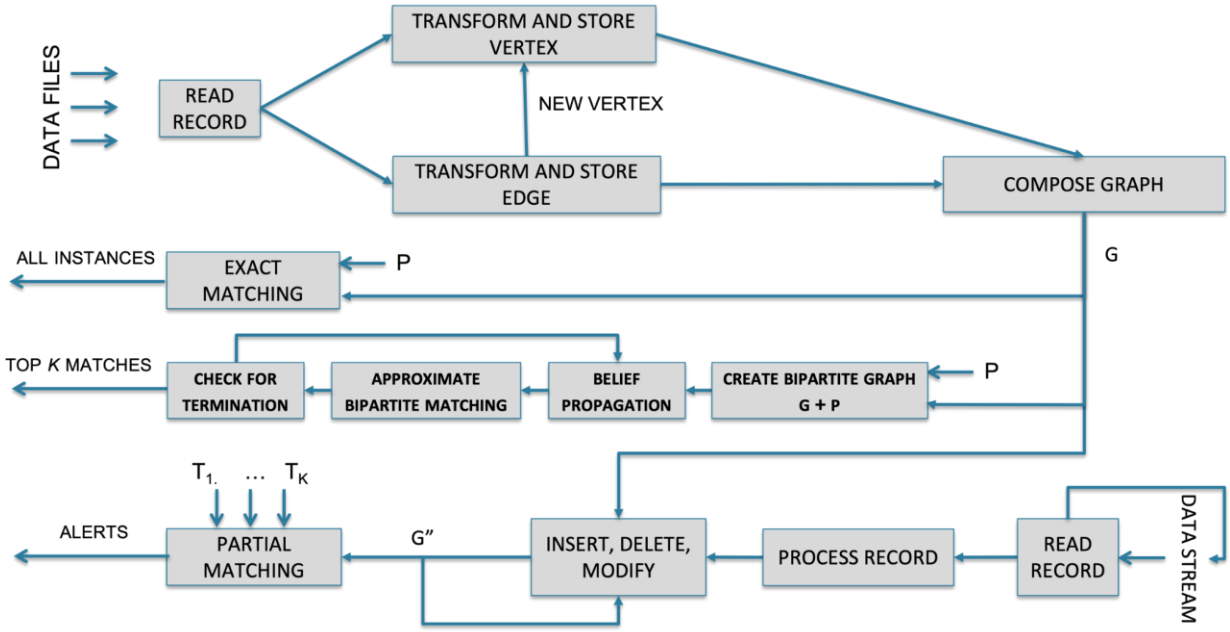


Figure 8 – A task graph for Workflow 2

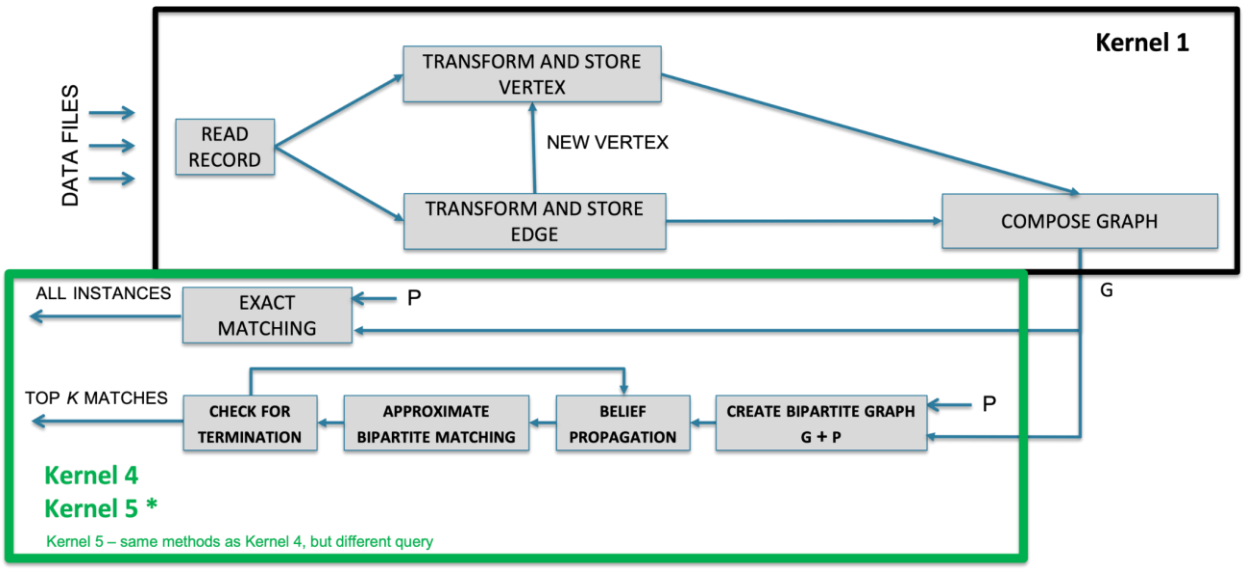


Figure 9 – Workflow 2, Kernels 1, 4, and 5

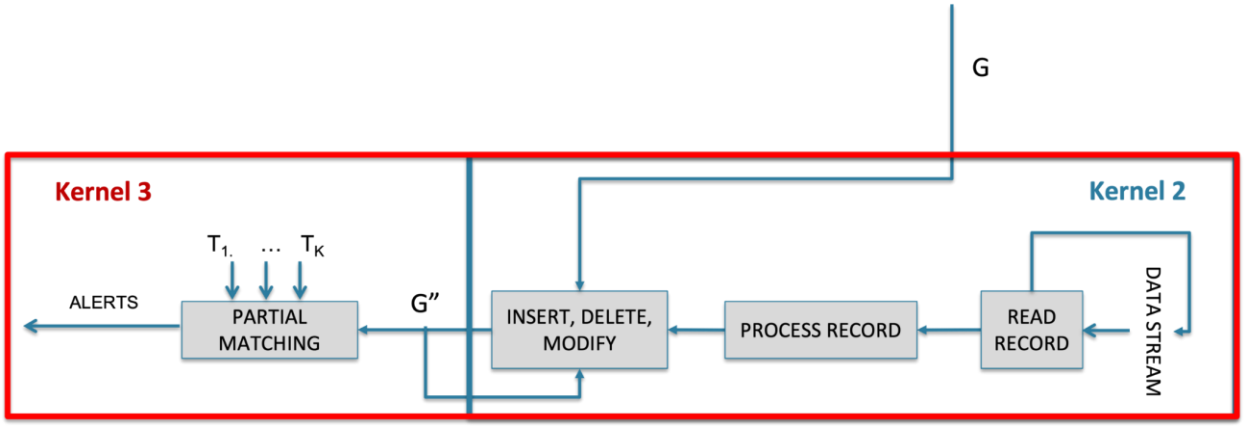


Figure 10 – Workflow 2, Kernels 2 and 3

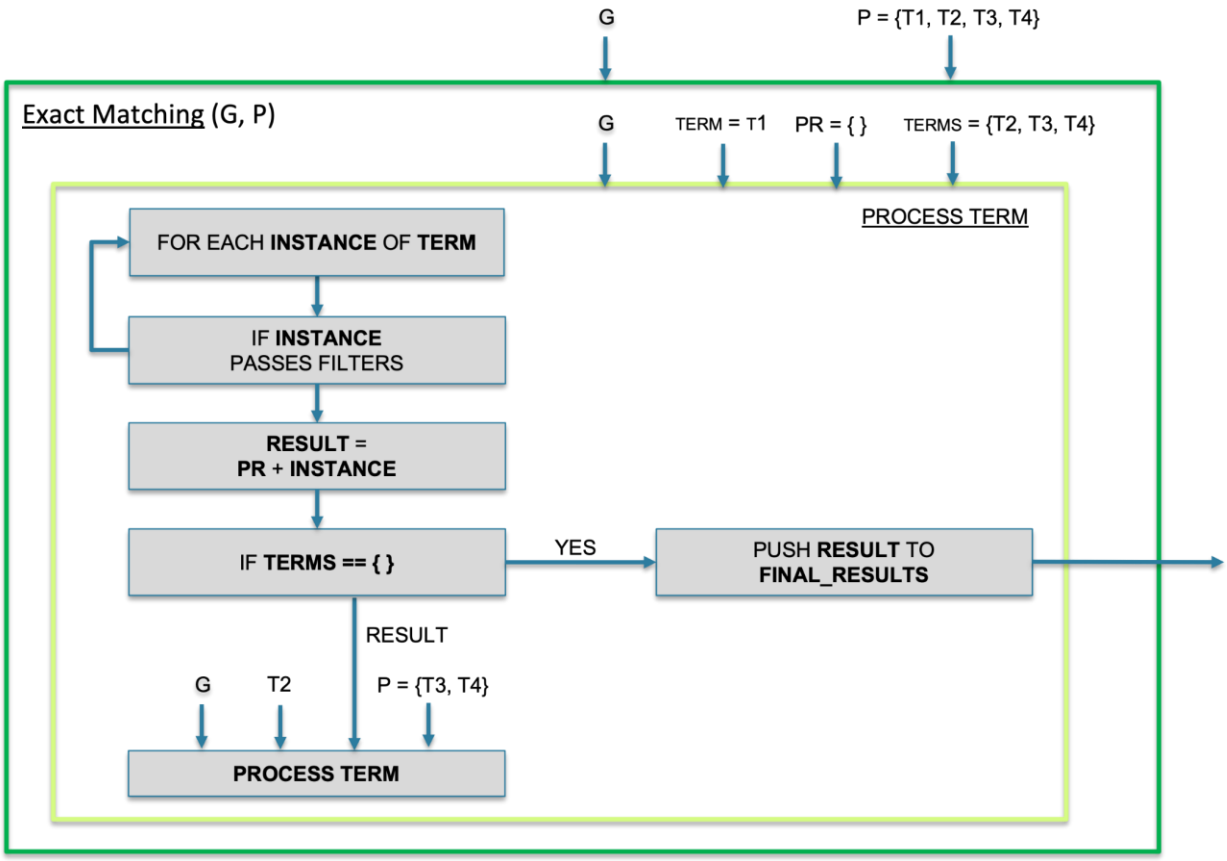
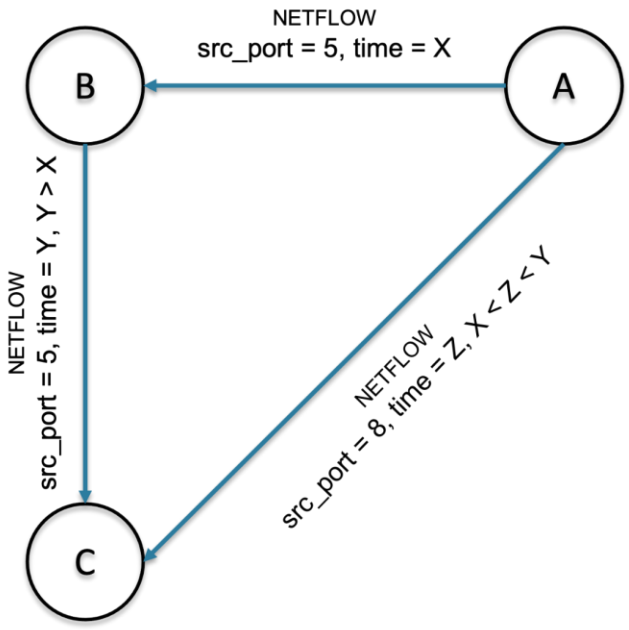


Figure 11 – Task graph for exact matching




```
P = {T1 = (E1:netflow, src_port = 5),
     T2 = (E2:netflow, src = E1.dst, time > E1.time),
     T3 = (E3:netflow, src = E1.src, dst = B.dst, E1.time < time < E2.time)
    }
```

Figure 12 – A netflow pattern of three connections

WORKFLOW 3 – Sequence Data – Identification and Clustering

Classification of sequence data occurs in many forms depending on the type of data: netflow records, sensor data, voice recordings, emails, blogs, etc. Important performance metrics are listed in Table 4. This table appears in Section A.2.3 of the BAA as Table A.2.3-3. If a system is unable to process records commensurate with the arrival rate, then records are dropped degrading analysis.

The first row (kernel) of table measures the time to read data records, transform the raw data, resolve vertex and edge ambiguities, and build-out the common internal data structures used by downstream tasks. Row 2 measures the rate at which records can be processed. Ideally, the rate is equivalent to the ITS rate measured in Row 1; if so, then no records are dropped.

Row 3 measures the time to compute a similarity graph of records using methods such as Jaccard Index [11] and row 4 measures the time to cluster the graph into communities of similar features or functionalities [12]. The fifth row measures the time to classify new records and recognize the emergence of new communities [13], and the final kernel measures the time to compute uncertainties and process alerts when communities merge, divide, or grow beyond a certain threshold [14]. Rates listed in rows 3, 4, and 5 for Today assume a graph of 10 billion vertices and100 billion edges running on departmental size cluster. For the same rows, the AGILE Target rates assume a graph of 1 trillion vertices and 100 trillion edges.

A problem description, task graph, and kernel specifications will be available after the start of the AGILE program.

Metric	Today	AGILE Target
Data ingestion rate	1 M records per second from a single source	10 M records per second from 3 or more sources (10x faster for 3 sources)
Records processing rate	0.1 M per second	10 M per second (100x faster)
Time to construct similarity graphs using metrics such as Jaccard index	400 hours	6 hours (67x faster)
Time to cluster similarity networks	500 hours	10 hours (50x faster)
Time to predict labels (functions) of new sequences	200 hours	4 hours (50x faster)
Time to estimate uncertainties of labels and functions, and prioritize alerts	NOT DONE	Completed

Table 4 – Metric Table for Workflow 3

WORKFLOW 4 – Network - Cyber-physical Systems

Table 5 lists the important rates for modeling network and cyber-physical systems. This table appears in Section A.2.3 of the BAA as Table A.2.3-4. The first rate measures the time to construct a model of a cyber-physical system using game-theoretic means. The second and third kernels measure the time to identify the *K* must influential nodes in the model using a linear cost function [15] and a time-dependent, non-linear cost function [16], respectively.

The fourth [17] and fifth kernel implement a game playing scenarios where red and blue teams make alternating moves attacking and defending the system. The kernels measure the time to analyze the system, chose among alternate moves, propagate model changes, and issue alerts of imminent breaches or collapse of subnetworks.

A problem description, task graph, and kernel specifications will be available at the end of March 2022.

Metric	Today	AGILE Target
Construct 1 PB graph through game theoretic modeling	120 minutes	2 minutes (60x faster)
Identification of top *k* influential nodes (simple model)	60 minutes	1 minute (60x faster)
Identification of top *k* influential nodes (enhanced model)	600 minutes	30 minutes (20x faster)
Propagate labels/confidence score	120 minutes	2 minutes (60x faster)
Incremental analysis	NOT DONE	Never recomputed from scratch

Table 5 – Metric Table for Workflow 4

References

1. https://en.wikipedia.org/wiki/Knowledge_graph
2. https://en.wikipedia.org/wiki/Graph_matching
3. Brzezinski and Stefanowski, *Stream Classification*, https://www.researchgate.net/publication/301349169_Stream_Classification
4. Lux and Budke, *Playing with Complex Systems? The Potential to Gain Geographical System Competence through Digital Gaming, Educ. Sci.* **2020**, *10*(5), 130; **https://doi.org/10.3390/educsci10050130**
5. Kipf and Welling, *Semi-Supervised Classification with Graph Convolutional Networks*, Proceedings of ICLR 2017, Toulon, France, 2017.
6. https://en.wikipedia.org/wiki/Link_prediction
7. Lin, Socher, and Xiong, *Multi-Hop Knowledge Graph Reasoning with Reward Shaping*, Proceedings 2018 Conference on Empirical Methods in Natural Language Processing, Brussels, Belgium, 2018.
8. Lengeling, Reif, Pearce, and Wittschko, *A Gentle Introduction to Graph Neural Networks*, Google Research, https://distill.pub/2021/gnn-intro/.
9. Gallagher, *Matching Structure and Semantics: A Survey on Graph-Based Pattern Matchine*, AAAI Fall Symposium: Capturing and Using Patterns for Evidence Detection, 2006.
10. Fan, et.al., *Distributed Incremental Pattern Matching on Streaming Graph*, SIGMOD '11, Athens Greece, 2011.

11. Besta, Kanakagiri, et.al., *Communication-Efficient Jaccard similarity for High-Performance Distributed Genome Comparisons,* Proceedings 2020 IEEE IPDPS, New Orleans, LA, 2020.
12. Fortunato, *Community detection in graphs*, Physics Reports, 486:3-5, February 2010.
13. Hollocou, Bonald, Maudet, and Legarge, *A linear streaming algorithm for community detection in very large network,* Proceedings Knowledge Discovery and Data Mining, Halifax, Canada, 2017.
14. Bertozzi and Merkurjev, *Chapter 12 – Graph-based optimization approaches for machine learning, uncertainty quantification and networks,* Handbook of Numerical Analysis, 20, 2019.
15. Li, Fan, Wang, and Tan, *Influence Maximization on Social Graphs: A Survey*, In IEEE Transactions on Knowledge and Data Engineering, 30:10, October 2018.
16. Yang and Pei, *Influence Analysis in Evolving Networks: A Survey, ,* In IEEE Transactions on Knowledge and Data Engineering, 33:3, March 2021.
17. Silver and Veness, *Monte-Carlo Planning in Large POMDPs*, Proceedings Advances in Neural Information Processing Systems 23, NIPS 2010, Vancouver Canada, 2010.