A Survey of State-of-the-Art Predictive Models for Graph Processing

Skip Thijssen
Vrije Universiteit Amsterdam, The Netherlands
Universiteit van Amsterdam, The Netherlands
skip.thijssen@student.uva.nl

ABSTRACT

This literature review examines predictive models for graph processing, identifying 15 models through analysis of ten papers. Our investigation shows a machine learning model preference and frequent diameter and maximum degree parameter use in hybrid CPU-GPU setups. Furthermore, more restrictive data structures consistently include network bandwidth-related parameters as a key predictive factor. These findings offer insight into the common methods and operational environments, allowing researchers and developers to make better-informed decisions regarding the design of their performance models.

KEYWORDS

Systematic Literature Review, Graph Processing, Performance Modelling and Prediction.

ACM Reference Format:
Skip Thijssen. 2024. A Survey of State-of-the-Art Predictive Models for Graph Processing. In *Literature Study*. Amsterdam, The Netherlands, 11 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Graphs are widely used to represent data structures and their execution across various applications. To optimize the method of processing these structures, we use predictive models. These models offer insight into graph algorithm execution, recommending optimal algorithms based on graph characteristics, and determining efficient resource allocations. Unfortunately, there is no standardized method of modeling graph processing performance. Furthermore, to the best of our knowledge, no overview of predictive models targeted at graph processing currently exists. For this reason, this literature study aims to provide an overview of the predictive models currently in use and create a broad classification of these models to help researchers and developers make better-informed decisions regarding the predictive model that fits their use case. To this end, we perform a systematic search through various databases and select prior research that is most relevant to our keywords. Next, we summarize the contents of these texts to provide insight into their relevancy. Lastly, we cluster the predictive models encountered in the texts into different classes based on our observations and document our insights into any notable patterns.

2 BACKGROUND

Graphs or *networks* are structures used to represent relationships between objects and consist of *vertices* (nodes) and *edges* (links). An example of a graph algorithm is Breadth-First Search (*BFS*), which explores all connected vertices of a graph in layers based on their distance to a source vertex.

A *predictive model*, in the context of graph processing, is an abstraction or simulation that uses historical data or knowledge of the underlying system to make predictions about future events or behaviors. These models can be utilized for application execution time predictions or guide the optimization effort by offering insights into the expected behavior of hardware configurations or algorithm selections.

Currently, several generic graph processing frameworks exist [12, 25, 27] which speedup the design of graph processing applications. However, there exists a trade-off between ease of implementation and performance. Early frameworks required algorithms to be written by hand, offering high performance for specific tasks. However, newer frameworks abstract many complexities of the implementation, easing development at the potential cost of performance degradation.

At a high level, graphs possess features such as *edges*, *vertices*, *clustering coefficients*, and metrics like diameter or the size of strongly and weakly connected components (*SCC* and *WCC*), which provide insight into different properties of the graph's structure.

Common graph processing tasks include Shortest Path (SP)[1] or Single-Source Shortest Path (SSSP), PageRank (PR)[2], and Triangle Counting (TC), each posing unique challenges. Algorithms like BFS[3] or Depth-First Search (DFS)[4] , visualized in Figure 1, address some of these problems. Their performance is often evaluated by metrics such as Traversed Edges Per Second (TEPS), which shows the processing speed in relation to the size of the graph.

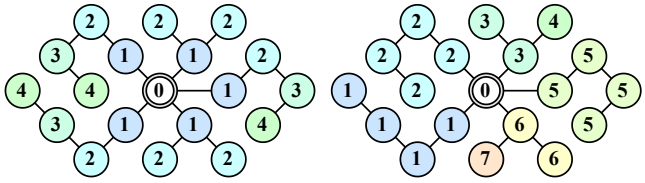

Figure 1: Example of BFS (left) compared to DFS (right) algorithm exploration of an arbitrary graph starting at vertex *0*.

3 RELATED WORK

Various graph processing frameworks and suitable hardware configurations, particularly in parallel and distributed systems, have been explored in surveys focusing on different aspects of the domain. However, to our knowledge, none specifically address the types of

[1] https://en.wikipedia.org/wiki/Shortest_path_problem
[2] https://en.wikipedia.org/wiki/PageRank
[3] https://en.wikipedia.org/wiki/Breadth-first_search
[4] https://en.wikipedia.org/wiki/Depth-first_search

predictive models suitable for graph processing performance. We identify the following four surveys whose work closely relates to our investigation of graph performance models.

[16] examines vertex-centric frameworks for large-scale distributed graph processing. Although it provides insights into the design and use of vertex-centric approaches for graph analytics, it does not explore models for predicting the performance of these frameworks.

Similarly, [18] offers an overview of parallel graph processing frameworks, outlining their architectural differences and performance characteristics. However, the survey does not focus on predictive models that could estimate performance or execution time across these frameworks.

[11] surveys graph processing accelerators, discussing the challenges and opportunities encountered when investigating hardware and software solutions designed to speed up graph processing tasks.

[1] investigates the performance of CPU, GPU, and hybrid graph processing frameworks, comparing their effectiveness across different graph processing tasks. While it surveys a range of frameworks, the study does not consider the use of predictive models for investigating runtime performance.

In contrast to these surveys, the current literature study focuses specifically on predictive models for the estimation of graph processing performance at various levels. Previous surveys have discussed graph processing frameworks, their challenges, and hardware accelerators but did not specifically address the predictive methods available for estimating performance. In our work, we aim to address this topic by providing an overview of predictive models, which can be helpful for informed decision-making in graph processing projects.

4 STUDY DESIGN

4.1 Research Goal

This study's research was performed with the intention of providing an overview of the current state-of-the-art of predictive models for graph processing. Specifically, using the Goal-Question-Metric approach [6], we formulate our objective as follows:

Purpose	Investigate and analyze the predictive models for graph processing currently in use.
Issue	The challenge of selecting an optimal graph processing predictive model from the available methods.
Object	Across various computing environments, including distributed and parallel systems.
Viewpoint	From the perspective of researchers and developers aiming to understand and optimize graph processing performance by modeling program behavior.

By categorizing predictive models and their inputs, this study highlights the common differences and similarities between modern graph processing environments and the predictive graph features they use as input parameters to model performance.

4.2 Research Questions

In this literature study, we aim to explore the various predictive models useful for performance prediction of graph processing. We formulate the research question underlying this goal as follows:

"What is the current state-of-the-art for performance modeling in graph processing?"

To address this question, we identify the different approaches to performance modelling, ranging from analytical models to machine learning-based methods. Furthermore, we intend to provide an overview of the current state-of-the-art for graph performance modelling.

4.3 Initial search

Our initial search was systematically carried out using Google Scholar, Semantic Scholar, and the University of Amsterdam's multi-database search engine[5]. Each search engine was queried for the same sets of keywords. We use the first set to describe our main subject as "graph" or "network". The second generically describes the type of models as "predict", "project", or "estimate", and the third set uses terms related to performance like "execution time" or "runtime". The exact queries differ per search engine due to a difference in query formulation. We only include papers from the first two search engine result pages, aiming to include the papers judged to be the most relevant while still keeping the initial selection limited in size.

4.4 Application of selection criteria

We apply a set of inclusion and exclusion criteria to our initial selection to refine the current list. We prioritize studies that directly address performance prediction in graph processing, excluding those not written in English, duplicate occurrences, or obviously irrelevant to the topic of predictive modeling for graph processing.

4.5 Snowballing

Following the initial selection, we use a snowballing technique to increase our list of likely relevant literature. We briefly examined the titles and abstracts to find promising papers and then read their related works section to discover more related papers. We also made use of the articles suggested as related by Google and Semantic Scholar. This method helps find additional relevant studies, previously overlooked in the initial search, significantly increasing our literature selection to approximately one hundred papers, excluding duplicates.

4.6 Filtering

The next step involved an automatic filtering process, where we sorted the expanded set of papers by keyword occurrences. Papers mentioning our targeted keywords[6] less than our threshold of five times were determined to be likely less relevant and excluded from further reading. This filtering approach reduced our selection to twenty-eight papers. A brief manual review of the paper abstracts allowed us to discard an additional 16 papers that were similar to our research topic but failed to cover our subject in detail. For example, a number of papers passed our automated selection on the 'graph' keyword. However, on further investigation, these papers used 'graph' in the context of figures, not networks. This final

[5] https://lib.uva.nl/

[6] ('graphs' or 'networks') and ('predict' or 'project' or 'estimate') and ('performance' or 'execution time' or 'runtime')

filtering process left us with a collection of ten studies, which we expected to be highly relevant to our work.

4.7 Data Extraction

We analyzed the ten remaining papers, focusing on sections related to performance prediction. This process involved categorizing the different models and accompanying frameworks to create an overview of the process of creating a prediction model and for what purpose the model is used in the article, as most models are predictive for a specific task, making them not interchangeable and requiring a more general way of comparing their results.

4.8 Data Synthesis

We fully read the selected papers in order to identify the different environments to which the models are suitable and to find out what their limitations are. This process aims to cluster the findings across multiple studies and is used to highlight common approaches, challenges, and performance characteristics. This results in the overview of predictive modeling techniques for graph processing discussed in Section 5.2.6 and 6.5.

4.9 Study Replicability

To ensure study applicability, we outline the generic search query applied to Google Scholar[7], Semantic Scholar[8], and the UvA's CataloguePlus search engine[9], as well as the subsequent selection and categorization of papers. We formulate the search query as follows:

"*(graph OR network) AND (predict OR project OR estimate) AND ("execution time" OR runtime OR performance)*"

This search query is changed according to each search engine's operator requirements (e.g., '|' instead of 'OR'). Furthermore, when supported, we further narrow the search by setting the discipline to "Computer Science', language to 'English', and earliest publication date to '2000' to ensure recency.

We document the resulting selection of papers in a spreadsheet[10]. This document outlines our preliminary observations on research relevancy, publication date, type, prediction model specifics when applicable, and any notes on the topic.

4.10 Threats to Validity

This subsection outlines potential limitations and concerns regarding the validity of the study's findings.

4.10.1 External Validity. Our findings focus on predictive models for graph processing. However, some insights, like those concerning hardware and overhead parameters, could be relevant to topics unrelated to graph processing, such as performance modeling for distributed systems.

4.10.2 Internal Validity. Our analysis is limited to the existing models, which vary significantly across computing environments and methodologies. Consequently, direct comparisons between models often involve differences in multiple variables. For example, comparing an analytical model with a machine learning model may also introduce a difference in execution time target.

4.10.3 Construct Validity. Our strict literature selection criteria might exclude potentially relevant studies. Our focus on performance models related to graph processing may exclude models whose performance predictions are not limited to a specific use case, like graph processing.

4.10.4 Conclusion Validity. The study's conclusions are based on a selection of ten papers, which limits the generalizability of observed patterns. This small scale means that findings should be considered indicative of potential trends in current research rather than a definitive framework on the usefulness of graph features under different conditions.

5 PAPER ANALYSIS

We analyze the literature selection presented in Section 4 and present an initial clustering of papers in Table 1. The table clusters papers based on their disc type(s) of graph processing prediction models as Machine Learning, Non-Machine Learning (denoted as Analytical), or both.

5.1 Machine Learning-based Modelling

5.1.1 Mix-and-Match: A Model-Driven Runtime Optimisation Strategy for BFS on GPUs [23]. In [23], Verstraaten et al. implements a performance model as part of their optimization efforts to speed up the Breadth-First Search (BFS) for GPUs. Specifically, they propose and implement the 'mix-and-match BFS' algorithm, which is designed to select the most efficient algorithm for each graph level during processing, thereby aiming to boost performance. However, the algorithm relies on predictions of performance for each graph level in order to switch to the right algorithm for the task. This is a nontrivial task, as there currently is no clear understanding of how structural properties of input graphs affect BFS performance.

The performance model formulated by Verstraaten et al. to address this prediction challenge characterizes the workload using performance data from real-world graphs in the KONECT repository [13]. The data is used to train a binary decision tree (BDT), which serves as the decision-making mechanism for algorithm selection. A BDT model is chosen to balance prediction accuracy and speed, as it allows for real-time performance prediction to inform algorithm choices.

The BDT's decisions are based on the training and validation of 32 BFS implementations, each implementing a different variation of neighbourhood iteration primitives. These include two edge-centric, two vertex-centric, and one virtual warp-based primitive. The experiments show that there is indeed a significant difference between the performance of these primitives, emphasizing the importance of choosing the right algorithm for each graph level.

Verstraaten et al. finds that the most important features relied upon by the BDT model include the graph size, frontier size, discovered vertex count, and degree distribution. Using these features coupled with the performance data from all BFS implementations on KONECT graphs, the authors form a dataset that associates each BFS level with its graph's structural properties. This dataset is split into a 60% training set and a 40% validation set. The mix-and-match

[7] https://scholar.google.com/
[8] https://semanticscholar.org/
[9] https://lib.uva.nl/
[10] https://docs.google.com/spreadsheets/

Table 1: An overview of the selected literature and its initial clustering by model type.

Title	Author	Ref	Model Type
A yoke of oxen and a thousand chickens for heavy lifting graph processing	Gharaibeh et al.	[8]	Analytical
Direction-optimizing Breadth-First Search	Beamer et al.	[4]	Analytical
Optimizations and Analysis of BSP Graph Processing Models on Public Clouds	Redekopp et al.	[21]	Analytical
GraphPi High Performance Graph Pattern Matching through Effective Redundancy Elimination	Shi et al.	[22]	Analytical
Partitioning-Aware Performance Modeling of Distributed Graph Processing Tasks	Presser and Siqueira	[20]	Analytical
Performance modelling and cost effective execution for distributed graph processing on configurable VMs	Li et al.	[14]	Analytical & Machine Learning
HeteroMap: A Runtime Performance Predictor for Efficient Processing of Graph Analytics on Heterogeneous Multi-Accelerators	Ahmad et al.	[2]	Analytical & Machine Learning
Predictive modeling and scalability analysis for large graph analytics	Medya et al.	[17]	Machine Learning
Mix-and-Match: A Model-Driven Runtime Optimisation Strategy for BFS on GPUs	Verstraaten et al.	[23]	Machine Learning
A Performance and Recommendation System for Parallel Graph Processing Implementations	Pollard et al.	[19]	Machine Learning

BFS trained on this dataset achieves about 100% slowdown when compared to the optimal algorithm choice, but this is still a 40% improvement over the best non-switching BFS implementation.

One challenge noted is the need for different graph representations in memory required by the vertex- and edge-centric implementations. To address this, the authors opt to keep these representations in memory, instead of accessing the required representation on the fly. The chosen in-memory representation is a combination of the Compressed Sparse Row (CSR) (vertex-centric) and edge-list (edge-centric) formats.

The final BDT is implemented in C++ based on the implementation of the CART algorithm in Python's scikit-learn library[11], and the resulting mix-and-match BFS outperforms existing methods like Gunrock [25] and LonestarGPU 2.0 [12] on average by a factor of 4.5 and 45, respectively.

5.1.2 A Performance and Recommendation System for Parallel Graph Processing Implementations [19]. With their paper, Pollard et al. addresses the difficulty of selecting the graph package most suitable for parallel and distributed graph processing. This challenge is due to the varying hardware requirements and performance characteristics of different packages, which depend heavily on the graph's properties. Pollard et al. proposes a solution which makes use of a prediction model that utilizes regression models and binary classification to label packages as well-performing and select the best.

To this end, the authors present the EPG* framework, which simplifies the comparison, installation, and performance analysis of four common graph algorithms: Breadth-First Search (BFS), Single Source Shortest Path (SSSP), PageRank (PR), and Triangle Counting (TC). EPG* simplifies the recommendation process for software packages by performing performance and scalability analyses based on the properties of the input graph, without requiring users to understand the packages being compared.

The package suggestion system of EPG* is based on downloading or generating graph datasets, running experiments on them, and using the results to train a recommender system. This system predicts either runtime or classifies configurations as well-performing based on twelve features: threads, edges, vertices, average clustering coefficient, triangle count (TC), fraction of closed triangles, diameter, 90th percentile effective diameter, and the fractions of nodes and edges in largest strongly and weakly connected components (SCC and WCC). The results are normalized using z-score normalization for regression and Traversed Edges Per Second (TEPS) for classification normalization. This prevents a skew in the models caused by negligible runtimes of small graphs.

The recommender system splits its predictive models in two: one for suggesting the best package using linear regression and binary classification, and another for predicting runtime with logistic regression and random forests for improved accuracy. The classification model, especially with random forests, proved more effective than linear regression, achieving up to 97% classification accuracy and demonstrating mean performance improvements ranging from 7% for PR to 700% for BFS.

In summary, EPG* offers a simplified framework for easy package selection for parallel and distributed graph processing tasks. The presented system judges the suitability of packages based on predictions of its runtime, and selects the best package using a classification system.

5.1.3 Predictive modeling and scalability analysis for large graph analytics [17]. Medya et al. introduces a regression-based approach to predict the performance and scalability of processing large distributed graphs. The authors show the necessity of this solution with the difference between the testing and real-world environments where graph processing applications are deployed. Namely, programmers often only have access to smaller clusters for experimentation, which makes a method to predict application behaviour on larger clusters important. This work highlights the scalability issues encountered as the cluster size increases, especially how the increase in communication can negatively impact application scalability and reduce performance. The proposed solution combines experiments with an "interconnect bandwidth throttling" tool [24] to analyze the impact of bandwidth on application performance. Using these observations, Medya et al. sets up an ensemble/collection of analytical models to analyze scalability and performance.

The ensemble models aim to predict the completion time (CT), starting with a linear regression model that evaluates CT based on the size of the cluster and the number of processes involved. At first, the model estimates the CT by accounting for how both the processing workload and communication between processors affect performance. It describes the relationship with Equation 1.

$$CT(p) = C_1^p * 1/p + C_2^p * 1/\sqrt{p} \quad (1)$$

[11] https://scikit-learn.org/stable/modules/tree.html#tree-algorithms-id3-c4-5-c5-0-and-cart

C_1^p and C_2^p are coefficients derived using linear regression that show the impact of increasing the number of processors on completion time, with p being the processor count. The term $\frac{1}{p}$ shows that more processors can reduce the workload per processor, while $\frac{1}{\sqrt{p}}$ shows how the communication overhead increases as more processors are added.

This basic model is then refined by accounting for the dataset size and the interconnect bandwidth. The dataset size is included to account for variations in the amount of data processed, and the bandwidth is included to describe the associated impact on the time it takes for processors to communicate with each other. This process uses data from the Graph500 benchmark [3], measuring BFS traversal performance.

Through the use of an "interconnect bandwidth throttling" tool, the models can analyze how varying bandwidth levels influence application performance, leading to a more accurate analytical model. This refined model includes the previous effects of processor count and dataset size on completion time, and also considers how available bandwidth can affect communication time.

In summary, Medya et al. provides a model based on basic assumptions, which is extended multiple times to incorporate different variables which are observed to affect performance. This method allows for predictions of large graph processing application completion time for different cluster configurations, dataset sizes, and bandwidth conditions.

5.1.4 HeteroMap: A Runtime Performance Predictor for Efficient Processing of Graph Analytics on Heterogeneous Multi-Accelerators [2]. In their work, Ahmad et al. introduces HeteroMap, a performance prediction framework for heterogeneous multi-accelerator architectures, which addresses the challenge of real-time optimization of graph analytics processing. The challenge with real-time optimization is the increased variability in computational resources, together with algorithm combinations that perform more efficiently on different hardware. Some algorithms benefit from GPUs with their high concurrency and bandwidth, while others are better suited to multi-core processors with increased data caching capabilities. To this end, Ahmad et al. presents HeteroMap, a method which uses benchmarks, hardware, and graph characteristics to select parameters that optimize graph processing efficiency on these heterogeneous resources.

The HeteroMap framework includes three models that map input graph characteristics and graph benchmarks to the optimal inter- and intra-accelerator machine configuration. Using the resource recommendations of HeteroMap during execution, a speedup ranging from 5% to 3.8× over traditional single-accelerator setups can be achieved.

The three models that recommend configuration choices are 1) a decision tree analytical model, which uses performance and input variables to reduce the choice space of the inter- and intra-accelerator configurations, providing a simple and easily tunable method for defining optimal settings. 2) A deep learning prediction model based on a feed-forward four-layer neural network with 32 neurons per layer, which is used to learn non-linear relationships between benchmarks and machine configurations. 3) A regression prediction model that captures non-linear performance trends with a 7th-order polynomial, providing a simpler method, compared to a neural network, for identifying optimal configurations with an 85% prediction accuracy.

The evaluation compares the resource configurations predicted by HeteroMap against manual tuning and baseline setups. The results show that the deep learning prediction model outperforms the decision tree and regression prediction model with a classification accuracy of 90.5% and an overhead of 3.48 ms. While the decision tree achieves a lower overhead compared to the other models, it does not provide a speedup comparable to the deep learning model's 31%. Additionally, using the best-performing prediction model, HeteroMap consistently achieves within 10% of the ideal manual optimization, which is a significant improvement in accuracy over prior work. However, in cases when runtime is very short, the HeteroMap's overhead becomes more noticeable.

In summary, [2] presents a real-time optimization technique for machine choices based on benchmark and input graph characteristics. To this end, HeteroMap presents a decision tree model, a deep learning model, and a regression model. Based on an evaluation of all three models, the deep learning model is chosen as the most suitable for the authors' setup due to its high classification accuracy and relatively smaller runtime compared to the other models.

5.2 Analytical Performance Modelling

5.2.1 Partitioning-Aware Performance Modeling of Distributed Graph Processing Tasks [20]. In their study, Presser and Siqueira [20] address the challenges of accurately predicting performance for graph processing jobs in distributed parallel computing environments. The paper begins by pointing out the limitations of general-purpose parallel processing frameworks such as Hadoop/MapReduce [7], and Spark [26] for graph-based tasks. These limitations led to the development of specialized graph processing frameworks, including Pregel [15], PowerGraph [9], and GraphX [10], with Pregel's Bulk Synchronous Parallel (BSP) framework being widely adopted.

The focus of this paper specifically is GPS, an open-source framework based on Pregel. Presser and Siqueira introduce a performance prediction model specifically designed for GPS, in which special attention is paid to graph partitioning. Namely, the common method of Hash Partitioning distributes vertices and edges randomly, which suffers from inefficient edge cuts. To address this problem, the authors present an alternative approach called *over-partitioning*, which involves creating more partitions than there are workers, and then combining these partitions into a single logical partition per worker. This method improves estimation accuracy by providing insight into the workload per worker based on the number of edges and vertices it handles.

Using these partitioning techniques, Pregel job performance can be estimated by splitting the total computation time into three phases: loading the input graph, executing the user-defined function at each superstep, and saving the results ($T_{total} = T_{load} + T_{superstep} + T_{save}$). While estimating loading and saving times is relatively simple using regression methods, estimating the supersteps requires more in-depth analysis, which differs depending on the chosen graph distribution algorithm.

For hash-based partitioning, an even distribution of vertices and edges is assumed, simplifying the calculation of the maximum number of messages. In contrast, the over-partitioning approach bases

estimations on the statistics of the specific partitioning algorithm, focusing on the workload of the largest logical partition and its communication overhead to estimate worst-case performance.

The resulting superstep estimation model bases its predictions on statistics of the input graph and three variables describing the computation time, communication overhead, and edge distribution differences. A fourth variable is added when using hash partitioning, which accounts for imbalances in the partitioned graph. These variables are calibrated on historical job data, after which the superstep model can be used for job execution time estimation by summing the job's expected number of supersteps and adding the load and save times.

In summary, Presser and Siqueira present an approach to performance modelling targeted at Pregel-like frameworks, which accounts for graph distribution, workload, and communication overhead. However, in the process, it also addresses challenges that apply outside of GPS, like graph distribution among workers.

5.2.2 A yoke of oxen and a thousand chickens for heavy lifting graph processing [8]. In [8][12], Gharaibeh et al. explores the use of heterogeneous computing platforms, combining CPUs and GPUs, to address the challenges of processing large, complex graphs. These challenges include extensive memory requirements, poor algorithm locality, and inefficient load-balancing due to graphs' low diameter.

To this end, Gharaibeh et al. proposes the use of heterogeneous platforms as a cost-effective solution. As part of their solution, they present a performance model, as well as TOTEM, a BSP processing engine that simplifies graph algorithm implementation on heterogeneous platforms.

The quality of TOTEM is determined using a specialized performance model, which assesses the result of distributing graph processing tasks across CPUs and GPUs. This model is based on the Bulk Synchronous Parallel (BSP) processing model and simplifies the computing environment to a single CPU and GPU for analysis purposes, but is extendable to many. The model calculates the processing time of a partition $G_p = (V_p, E_p) \subseteq G$ of the graph $G = (V, E)$ on a processing unit p with Equation 2.

$$t(G_p) = \frac{\left|E_p^b\right|}{c} + \frac{|E_p|}{r_p} \tag{2}$$

$|E_p^b|$ represents boundary edges, c is the communication speed, and r_p is the processing rate of processor p in edges per second. This formula shows the time required to process a partition as the sum of the communication time of the boundary edges and the processing time of the edges in processor p's partition. Additionally, the makespan $m_p(G)$ of workload G on platform P is defined as Equation 3, which indicates that the total processing time is limited by the slowest partition.

$$m_p(G) = \max_{p \in P} \left\{ t\left(G_p\right) \right\} \tag{3}$$

This performance model is used to evaluate the performance of the TOTEM graph processing engine, a framework designed for heterogeneous, dual-socket, multi-GPU, single-node systems implemented in C and CUDA. TOTEM uses a Compressed Sparse Rows (CSR) graph representation and consequently only allows fixed structured graphs due to the expensive cost of updating the graph during execution. These features and limitations result in TOTEM achieving close to the predicted speedup when offloading graphs to the GPU and negatively deviating from the communication overhead prediction when graph density is reduced.

In summary, this work highlights the usefulness of heterogeneous platforms for graph processing, providing a detailed performance model and an accompanying framework, TOTEM, for simplified implementation.

5.2.3 Performance modelling and cost effective execution for distributed graph processing on configurable VMs [14]. Li et al. presents an analytical model to estimate the runtime of distributed graph processing tasks of GPS, a framework similar to Pregel [15]. This performance prediction model uses features like the graph's structure, the complexity of graph algorithms, the implementation details of the Graph Processing System (GPS), and the specifications of the underlying virtual machines (VMs). The analytical model is compared against two popular machine learning alternatives, Artificial Neural Networks (ANN) and Support Vector Machines (SVM), and shows improved accuracy with its predictions. Using these predictions, a VM execution scenario is chosen that optimizes cost efficiency.

The authors highlight that a single machine is unable to efficiently process the increasingly large graph sizes and the associated increase in algorithm complexity. For this reason, they explore cost-efficient distributed graph processing based on a predictive model. However, poor memory locality and frequent synchronization among VMs complicate the prediction of execution times.

To handle this challenge, Li et al. sets up a prediction model that splits the runtime prediction of GPS for various execution scenarios into three parts: 1) the *computation workload*, 2) the *communication traffic*, and 3) the *synchronization overhead*. This division allows for a more accurate representation of GPS, including the parsing and processing of messages between vertices, which significantly impacts memory locality and processing efficiency.

The *computation workload* considers both the delivery of messages along edges (E) to vertices (N) and the processing of these messages (p_1 to p_4), calculated using Equation 4.

$$T_{comp} = \frac{N}{M} \times p_3 + \frac{E}{M} \times q_1 \times p_1 \times p_4 \tag{4}$$

Where q_1 to q_3 are features of the GPS implementation. The *Communication workload* is assessed based on the transmission of messages between workers, with Equation 5.

$$T_{comm} = \frac{p_1 \times p_2 \times q_1 \times E \times (M-1)}{M^2 \times B} \tag{5}$$

Lastly, *Synchronization overhead*, which ensures consistency across VMs at the end of a superstep, is represented as Equation 6, where M is VM count.

$$T_{syn} = q_2 \times M^{q_3} \tag{6}$$

The total execution time calculation of a superstep (T_{step}) can then be computed by combining these parts as Equation 7.

$$T_{step} = K_{comp} \times T_{comp} + K_{comm} \times T_{comm} + T_{syn} \tag{7}$$

[12]The title refers to a quote by Seymour Cray about the future of supercomputers being a single fast processor instead of many small processors running in parallel: https://en.wikipedia.org/wiki/Seymour_Cray#SRC_Computers

K_{comp} and K_{comm} account for the partially overlapping computation and communication, respectively. Furthermore, when the total superstep count is known, the total job runtime can be predicted. Lastly, the values of the parameters are calibrated using Generalized Reduced Gradient (GRG) for different execution scenarios.

Experiments using synthetic and real-world graphs demonstrate the model's predictive accuracy, achieving up to 95% accuracy and outperforming the two machine learning models. This precision allows for close-to-optimal execution scenario recommendations for cost efficiency.

In conclusion, Li et al. presents a system-aware performance prediction model for distributed graph processing on VMs that outperforms popular machine learning methods. This allows users to make better-informed decisions about the most cost-efficient execution scenarios. However, the parameters of the analytical model require calibration for each new scenario.

5.2.4 Direction-optimizing Breadth-First Search [4]. In their study, Beamer et al. introduces a hybrid Breadth-First Search (BFS) algorithm suited to low-diameter, scale-free graphs commonly found in social networks. This hybrid method combines the traditional top-down search with a new bottom-up approach, which significantly reduces the number of edges examined. The motivation behind this work is the lack of locality and low computational intensity of BFS graph processing applications, which are often memory-bound on shared-memory systems or communication-bound on clusters.

Scale-free graphs, whose degree distribution follows a power-law, present significant load-balancing challenges due to the highly variable work per node. Low-diameter graphs worsen the problem with the many cross-edge communications, which makes partitioning difficult. To solve these problems, the authors propose a hybrid BFS search that minimizes the edge examinations by dynamically switching between top-down and bottom-up BFS based on the size of the frontier.

The implementation of the hybrid algorithm is based on a predictive model that determines the optimal level to switch between the two BFS approaches. The model takes into account the number of edges to check from the current frontier (m_f), the number of vertices within the frontier (n_f), and the edges to check from unvisited vertices (m_u). The model switches to bottom-up search following Equation 8, where α is a tuning parameter that adjusts the threshold for switching.

$$m_f > \frac{m_u}{\alpha} = C_{TB} \tag{8}$$

However, when the frontier becomes too small, the model switches back to top-down search following Equation 9.

$$n_f < \frac{n}{\beta} = C_{BT} \tag{9}$$

The parameter β needs to be tuned, similar to α. These two models allow the algorithm to efficiently transition between top-down and bottom-up search at runtime, reducing edge examinations.

The evaluation of the hybrid BFS algorithm, implemented in C++ using OpenMP and storing graphs in Compressed Sparse Row (CSR) format, shows an average speedup of 3.9 and a minimum speedup of 2.4 for all tested graphs. These performance improvements are due to the method picking the search strategy that is expected to be optimal for each graph frontier. In this case, the values of model parameters α and β that provided the greatest speedup were 14 and 24, respectively.

In conclusion, Beamer et al. presents a hybrid BFS algorithm suited for low-diameter graphs that uses a simple model to predict the optimal search strategy for a graph level. The algorithm in this paper discusses a single-node implementation, but this is extended in future work [5].

5.2.5 Optimizations and Analysis of BSP Graph Processing Models on Public Clouds [21]. In [21], Redekopp et al. presents a method to optimize Bulk Synchronous Parallel (BSP) graph processing models in public cloud environments, focusing on Microsoft's Azure Cloud. They address the challenges of resource utilization, graph partitioning effects, and dynamic scaling in cloud-based BSP frameworks, which significantly affect the performance of graph analysis algorithms. Their contributions include a heuristic for vertex scheduling to maximize resource utilization on cloud VMs, an analysis of graph partitioning with BSP, and a model that can dynamically scale the number of BSP workers, achieving an improvement in performance and reduction in cost.

The presented method uses adaptive and generalizable vertex and superstep scheduling heuristics within a .NET/C# BSP graph processing framework. This approach aims to predict required resource levels before execution and balance the load during execution. Redekopp et al. uses Azure cloud to deploy their model, including a web role for job submission, a job manager for BSP coordination, and worker roles for vertex tasks. Additionally, following the BSP method, all vertices must complete their execution, before the computation can move to the next step.

The optimization strategy introduces a vertex-scheduling model that only initiates computation for a subset of vertices (called 'swaths') at the same time to maximize resource utilization. This strategy uses performance prediction for determining swath size and optimal moment of initiation. These strategies aim to improve memory use, reduce resource spikes, and flatten resource usage over time.

The prediction of swath size and initiation is performed using two prediction models. The first model predicts swath size based on runtime observations and graph properties, including sampling small swaths for peak memory usage and adapting swath size based on previous memory usage. For swath initiation, Redekopp et al. proposes static and dynamic approaches to determine when to start the next swath. The static approach initiates a new swath at a predetermined interval, aiming to maintain memory usage within physical limits based on prior insights on the graph's structure. Conversely, the dynamic approach adjusts the initiation interval based on real-time observations of system conditions, such as message traffic and memory utilization, to optimize resource use throughout the computation process.

Redekopp et al. evaluate their approach using real-world datasets, compare their heuristics against baseline methods, and explore the impact of graph partitioning on BSP performance. The results show that the dynamic scaling heuristic with a percentage of active vertices achieves performance equal to an 8-worker setup using Google's web graph dataset, and greater performance when using a patent citation network. Additionally, by dynamically scaling, the presented method has a monetary cost lower than or comparable

to a 4-worker setup while offering a performance similar to an 8-worker setup.

In summary, Redekopp et al. develops a strategy for improving BSP graph processing on public clouds. Their work includes scheduling heuristics, graph partitioning analysis, and a dynamic scaling model. Their findings show significant improvements in cloud-based graph processing performance and cost reductions.

5.2.6 GraphPi High Performance Graph Pattern Matching through Effective Redundancy Elimination [22]. In their work, [22] introduces GraphPi, a high-performance distributed system for graph pattern matching, targeting the reduction of redundant computations caused by pattern symmetry and the challenge of selecting optimal matching orders. GraphPi's contributions include a 2-cycle based automorphism elimination algorithm, a 2-phase computation-avoid schedule generator, and a performance prediction model to efficiently determine the optimal matching order.

The graph pattern matching problem involves identifying all subgraphs (called embeddings) within a graph that are isomorphic[13] to a given pattern. This problem is worsened by automorphism[14] of the graph, which leads to unnecessary computations, and by the vast number of potential combinations of matching orders and restrictions, which significantly impact performance but are challenging to predict. GraphPi addresses these issues by introducing an approach that removes automorphisms and uses a performance prediction model to select the most efficient 'schedule' (the order in which each vertex is searched) and the set of restrictions.

In order to decrease the automorphism count and remove redundant computations, Shi et al. describes the following four approaches. Firstly, it uses a restriction generation algorithm that uses 2-cycle permutations to remove automorphisms, reducing their count to one. Secondly, GraphPi introduces a 2-phase computation-avoid schedule generator that identifies efficient schedules by removing schedules with redundant computations. Thirdly, a performance prediction model is added to estimate the performance of each configuration, which allows for the selection of the optimal combination of schedule and restrictions. Lastly, a method for counting embeddings based on the Inclusion-Exclusion Principle[15] is included, further improving its performance.

The prediction model predicts the performance of a graph pattern matching based on a schedule and a set of restrictions. The previously introduced algorithm uses nested loops, so the authors use a nested performance model, shown in Equation 10.

$$\text{cost}_i = \begin{cases} l_i \times (1 - f_i) \times (c_i + \text{cost}_{i+1}), & \text{for } 1 \leq i < n - 1 \\ l_i \times (1 - f_i), & \text{for } i = n \end{cases} \quad (10)$$

The model computes the execution cost of the ith loop ($cost_i$) using the number of pattern vertices (n), the loop size (l_i), the probability of filtering the embedding according to a restriction (f_i), and the computational overhead of intersection operations (c_i).

GraphPi is compared against GraphZero and Fractal to find all embeddings of 6 patterns in 5 graphs. The results show that GraphPi outperforms GraphZero and Fractal by an order of magnitude. Additionally, the performance model is compared against an oracle which shows that GraphPi's schedule predictions are 32% slower than the oracle's. Shi et al. mentions that prediction accuracy can be improved by including more information on the data graphs.

In summary, [22] introduces GraphPi, a framework which allows for high-performance graph pattern matching by utilizing a new restriction generation algorithm and a performance model to determine the matching order based on . The resulting framework outperforms existing frameworks by an order of magnitude.

[13] Graph isomorphism is the property of a graph to match *another graph* by rearranging its structure without changing any vertices or edges.

[14] Graph automorphism is the property of a graph to match *itself* by rearranging its structure without changing any vertices or edges.

[15] https://mathworld.wolfram.com/Inclusion-ExclusionPrinciple.html

6 COMPARATIVE ANALYSIS

In this analysis, we examine the ten main models presented in the summarized literature and five alternative approaches which did not yield the most accurate results. We separate the analysis of the frameworks from the classification of the input parameters in Tables 2 and Table 3. This structured examination aims to highlight the differences and similarities between the models and to predictive modelling of various graph processing-related tasks.

6.1 Model Overview

In Table 2, we present a structured comparison of fifteen predictive models across five main categories: environment, methodology, limitations, and model input/output. The operational environment encompasses the hardware, programming models, and architecture setup for each model, indicating the setup requirements for each model. For instance, models utilizing CPUs differ in approach to accounting for parallelism from those on GPUs, impacting their setup. Methodology encompasses the model types, execution targets, and measured accuracy, offering insights into the workings and effectiveness of each model. Limitations focus on the generalizability of each model based on the types of problems the model can be applied to and the restrictions on the input graph, e.g., Scale-Free (SF) or Small-World (SW). The input/output category outlines the complexity of each model based on the number of parameters involved and the nature of their output, distinguishing between time-based predictions indicated with a 'T'-symbol, and recommendations, like optimal algorithm or framework, indicated with an 'R'-symbol. The parameters are divided into three categories: 'Few' for less than five parameters, 'Moderate' for 5 to 15, and 'Many' for more.

6.2 Parameter Clustering

Table 3 summarizes the input parameters on which the models base their predictions, categorized under graph structural properties, BSP-specific algorithm features, hardware characteristics, and overhead considerations. Graph structural properties such as edge and vertex counts, graph diameter, and maximum degree, reflect the graph features that are deemed predictive for the given task. BSP-related parameters show algorithmic features unique to this graph processing model, while the hardware parameters summarize the hardware characteristics that are expected to impact performance. Finally, the Overhead parameters encompass a broader set of factors, from runtime to synchronization, originating from the applied algorithm and environment that are expected to impact performance.

Table 2: An overview of the predictive models presented in the selected literature based on 12 metrics. Empty fields indicate unspecified information.

Article Overview			Environment			Methodology			Limitations		Model Input / Output	
Ref	Year	Type	Hardware	Processing Model	Architecture	Model Type	Execution	Accuracy	Problem	Topology	Parameters	Metric
[8]	2012	A	Hybrid	BSP	Single	Analytical	Offline	?		SF	Moderate (7)	T
[4]	2012	A	GPU		Multi	Analytical	Online	75.0%	BFS		Moderate (6)	R
[21]	2013	A	CPU	BSP (GPS)	Multi	Analytical	Online	?		SF, SW	Moderate (8)	R
[22]	2020	A	CPU		Multi	Analytical	Online	?	Pat. Match.		Moderate (5)	T
[20]	2023	A	CPU	BSP (GPS)	Multi	Analytical	Offline	90.0%			Moderate (11-12)	T
		A	CPU	BSP (GPS)	Multi	Analytical	Online	95.0%		SF, Regular	Moderate (11)	T
[14]	2017	ML	CPU	BSP (GPS)	Multi	Artificial Neural Network	Online	77.3%		SF, Regular	Few (4)	T
		ML	CPU	BSP (GPS)	Multi	Support Vector Machine	Online	50.8%		SF, Regular	Few (4)	T
[17]	2017	ML	CPU		Multi	Linear Regression	Offline	?		SF	Few (3-4)	T
[23]	2018	ML	GPU		Multi	Binary Decision Tree	Online	?	BFS		Few (4)	R
[19]	2019	ML	Hybrid		Multi	Linear Regression, Binary Classification	Offline	97.0%	BFS, PR, SSSP, TC		Moderate (12)	R
		ML	Hybrid		Multi	Logistic Regression, Random Forest	Offline	?	BFS, PR, SSSP, TC		Moderate (12)	T
		A	Hybrid		Multi	Decision Tree	Online	86.2%			Many (17)	R
[2]	2019	ML	Hybrid		Multi	Deep Learning (FFNN)	Online	90.5%			Many (17)	R
		ML	Hybrid		Multi	Multi Regression	Online	85.4%			Many (17)	R

6.3 Legend

The categorization of Tables 2 and 3 use the following symbols to indicate full or partial matches, or absent information:

- ● Indicates a direct match to the category, reflecting a precise inclusion of the variable in the model.
- ○ Denotes a broader interpretation or an indirect match of a variable to fit a category.
- ? Marks the absence of specified information or clearly defined numeric values (e.g., accuracy is often denoted as "high").

6.4 Analysis of Patterns and Characteristics

Examining Tables 2 and 3 together shows that hybrid CPU-GPU configurations, as seen in [19], [2], and [8], appear to prefer machine learning models for prediction tasks. Analytical models are commonly used for predicting BSP execution. Furthermore, in our literature selection, execution time predictions by analytical models are only made for BSP-based frameworks, with the exception of [22], which is specifically designed for graph pattern matching. In regard to the execution target, we observe eight input parameters that are unique to prediction models with offline execution targets. These parameters include the boundary edges, closed triangles, weakly and strongly connected components, and edge density, among others.

Regarding the graph's structural properties, nearly all models judge the number of edges and vertices as predictive. Only [4] and [21] do not directly include the number of edges. [21] is a recommendation predictive model that schedules vertices on public cloud nodes by monitoring system conditions at runtime. This includes message traffic, which we expect to indirectly provide a similar metric to edge count. [4] is a direction-optimizing BFS algorithm that includes edges at the BFS frontier scale, instead of the global scale.

Hybrid (CPU & GPU) models, with one exception, consistently include diameter and maximum degree as metrics. We suspect that the common inclusion of these parameters is likely due to the increased challenge of efficiently adapting graph data structures to diverse hardware capabilities. In this case, the diameter can provide insight into the complexity of graph traversal and determine the upper bound on communication latency between workers. Conversely, the maximum degree can help with the workload distribution, where denser parts of the graph may be better suited to the low latency of a CPU, while the GPU processes large portions of sparsely connected vertices. The hybrid implementation that excludes these metrics is [8]'s single-node setup. The omission is likely due to their focus on a generic model that predicts execution times through processing rates and inter-component communication volumes, without relying on algorithm specific metrics.

Diameter metrics are included in the BFS models for hybrid hardware in [19]. However, the results show that in models specifically designed for the prediction of Breadth First Search (BFS) tasks, such as [4] and [23], the diameter is excluded. This is likely because the models focus on predicting BFS performance at the frontier level in both cases. Global metrics, like diameter, could prove indicative of total worst-case execution time, but [4] and [23] specialize in recommendations for the local graph structure.

Lastly, models applied to scale-free graphs, including [17], [8], and [14], consistently emphasize network bandwidth. Based on these three models, we assume this to be an indicator of the increased challenge of managing data distribution across nodes, potentially resulting in skewed data placement. This distribution is expected to necessitate a significant amount of network traffic, highlighting bandwidth as a potential limiting factor.

Table 3: The input parameters of the predictive models presented in the selected literature.

Article Overview		Graph Structure														BSP		Hardware					Overhead							
Ref	Model Type	# Edges	# Vertices	Diameter	Max. Degree	# Boundary Edges	# Frontier Vertices	# Frontier Edges	# Triangles	# Closed Triangles	Largest SSC	Largest WCC	Avg. Cluster Coef.	Edge Density	# Visited Vertices	# Supersteps	# Active Vertices	# Workers	# Threads	Utilization	Network Bandwidth	Memory Size	Data Type	Computation	Memory Access	Data Movement	Communication	Synchronization	Skip Work Prob.	Runtime
[8]	Analytical	●	●			●												●			●			●			●			
[4]	Analytical		●				●	●							○															
[21]	Analytical		●													●	●	●		●		●					●	●		
[22]	Analytical	●	●						●															●					●	
[20]	Analytical	●	●											●		●	●	●									●			○
[14]	Analytical	●	●													○		○			●			●		●		●		
	ANN	●	●													○		○			●									
	SVM	●	●													○		○			●									
[17]	LR	●	●															●			○									●
[23]	BDT	●	●				●								●															
[19]	LR, BC	●	●	●					●	●	●	●	●						●											
	LR, RF	●	●	●					●	●	●	●	●						●											
[2]	Analytical	●	●	●	●																		●	●	●	●		●		
	DL	●	●	●	●																		●	●	●	●		●		
	MR	●	●	●	●																		●	●	●	●		●		

6.5 Implications of Analyzed Patterns

Machine learning models excel at complex hardware setups. Based on the analysis of four machine-learning models, we conclude that their preference for hybrid CPU-GPU configurations is an indicator of their effectiveness in modeling complex hardware setups. This suitability is likely due to the high level at which the models describe the hardware and its impact on graph processing performance. By abstracting these interactions, machine learning models can be adapted to various hardware configurations without requiring detailed, component-level knowledge.

Analytical models excel at predicting subsystem or component-level behavior. Our observations show that the six analytical models are mainly used to describe smaller components of graph processing execution, such as predictions for local structures, scheduling, or initial resource allocations. These are cases where the impact of unknown variables is minimal. Following these observations, we conclude that analytical modeling is suited to precise performance predictions in environments with well-defined parameters, making them effective tools for understanding the impact of individual components on overall system behavior.

Hybrid models describe complex hardware by relying on upper bounds or averages. Hybrid models frequently integrate metrics such as diameter and maximum degree. This approach implies a strategy of managing uncertainties of complex hardware configurations by basing decisions on worst- or average-case scenarios rather than precise predictions. These insights indicate that initial model development could prove more effective when focusing on these upper bounds and averages.

Network bandwidth signifies expected communication deficiencies. The consistent inclusion of network bandwidth in models targeting scale-free graphs highlights the underlying challenge of managing data distribution and communication in distributed environments. This focus indicates that for graphs whose vertex connectivity varies significantly, optimizing for communication efficiency is of similar importance as computational efficiency. From these observations, we infer an expectation of inefficient data placement or task assignment in these models, emphasizing the need for the development of more advanced data distribution strategies that minimize network traffic and reduce communication bottlenecks.

7 CONCLUSION

This literature review focused on predictive models for graph processing, aiming to categorize and analyze current methodologies to help developers and researchers develop prediction models. We systematically searched and analyzed ten relevant papers and identified 15 predictive models, including analytical and machine learning models applicable to various computational environments.

Our findings indicate a preference for machine learning models in hybrid CPU-GPU environments due to their adaptability in complex settings. Analysis shows that edges and vertices are considered essential predictive parameters across models, with graph diameter and maximum degree primarily included in hybrid computing environments. Furthermore, models targeting scale-free graphs consistently prioritize network bandwidth, highlighting the increased challenge of data distribution and network traffic management.

Based on this analysis, we discussed four main implications, including the suitability of machine learning models for complex hardware setups, the usefulness of analytical models for predicting subsystem or component-level behaviors within larger systems, the tendency of hybrid models to describe complex hardware by focusing on general metrics like upper bounds or averages, and the emphasis on network bandwidth in models for scale-free graphs, which suggests expectations of communication deficiencies in these scenarios.

In conclusion, this study provides insight into the current state of predictive models for graph processing, showcasing their operational environments, methodologies, and common input parameters.

7.1 Considerations and Scope of Review

We purposefully limit our review to papers published within the last 20 years to ensure relevancy and focus exclusively on graph processing. We expect this approach to capture the majority of predictive models for graph processing. However, these criteria will exclude studies of models applicable to areas outside of graph processing that may still prove predictive for our purposes.

Additionally, the categorization we formulate aims to systematically organize the landscape of predictive models. Due to the generic formulation of the model environments in Table 2, we expect this categorization to fit all classes of models. However, the input parameters in Table 3 may not cover all classifications. As this is an ongoing area of study with no standardized method of predicting graph processing performance, we expect the set of parameters describing the graph's structural properties to be expanded with unique or derived properties in future iterations of predictive models that would better describe graph processing behavior.

7.2 Future Work

In future work, a greater selection of predictive models is needed to discover patterns and stronger relationships between the environments and input parameters of predictive models, potentially by including models outside the scope of graph processing. A selection of similarly specialized models would improve the comparability and insights that can be gathered from the models.

REFERENCES

[1] Tanuj Kr Aasawat, Tahsin Reza, and Matei Ripeanu. 2018. How Well do CPU, GPU and Hybrid Graph Processing Frameworks Perform?. In *2018 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. 458–466. https://doi.org/10.1109/IPDPSW.2018.00082

[2] Masab Ahmad, Halit Dogan, Christopher J. Michael, and Omer Khan. 2019. HeteroMap: A Runtime Performance Predictor for Efficient Processing of Graph Analytics on Heterogeneous Multi-Accelerators. In *2019 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. 268–281. https://doi.org/10.1109/ISPASS.2019.00039

[3] James Alfred Ang, Brian W Barrett, Kyle Bruce Wheeler, and Richard C Murphy. 2010. *Introducing the graph 500.* Technical Report. Sandia National Laboratories (SNL), Albuquerque, NM, and Livermore, CA https://graph500.org/

[4] Scott Beamer, Krste Asanovic, and David Patterson. 2012. Direction-optimizing Breadth-First Search. In *SC '12: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*. 1–10. https://doi.org/10.1109/SC.2012.50

[5] Scott Beamer, Aydin Buluç, Krste Asanovic, and David Patterson. 2013. Distributed Memory Breadth-First Search Revisited: Enabling Bottom-Up Search. In *2013 IEEE International Symposium on Parallel & Distributed Processing, Workshops and Phd Forum*. 1618–1627. https://doi.org/10.1109/IPDPSW.2013.159

[6] Victor R Basili1 Gianluigi Caldiera and H Dieter Rombach. 1994. The goal question metric approach. *Encyclopedia of software engineering* (1994), 528–532.

[7] Jeffrey Dean and Sanjay Ghemawat. 2008. MapReduce: simplified data processing on large clusters. *Commun. ACM* 51, 1 (jan 2008), 107–113. https://doi.org/10.1145/1327452.1327492

[8] Abdullah Gharaibeh, Lauro Beltrão Costa, Elizeu Santos-Neto, and Matei Ripeanu. 2012. A yoke of oxen and a thousand chickens for heavy lifting graph processing. In *Proceedings of the 21st International Conference on Parallel Architectures and Compilation Techniques* (Minneapolis, Minnesota, USA) *(PACT '12)*. Association for Computing Machinery, New York, NY, USA, 345–354. https://doi.org/10.1145/2370816.2370866

[9] Joseph E. Gonzalez, Yucheng Low, Haijie Gu, Danny Bickson, and Carlos Guestrin. 2012. PowerGraph: Distributed Graph-Parallel Computation on Natural Graphs. In *10th USENIX Symposium on Operating Systems Design and Implementation (OSDI 12)*. USENIX Association, Hollywood, CA, 17–30. https://www.usenix.org/conference/osdi12/technical-sessions/presentation/gonzalez

[10] Joseph E. Gonzalez, Reynold S. Xin, Ankur Dave, Daniel Crankshaw, Michael J. Franklin, and Ion Stoica. 2014. GraphX: Graph Processing in a Distributed Dataflow Framework. In *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)*. USENIX Association, Broomfield, CO, 599–613. https://www.usenix.org/conference/osdi14/technical-sessions/presentation/gonzalez

[11] Chuang-Yi Gui, Long Zheng, Bingsheng He, Cheng Liu, Xin-Yu Chen, Xiao-Fei Liao, and Hai Jin. 2019. A Survey on Graph Processing Accelerators: Challenges and Opportunities. *Journal of Computer Science and Technology* 34, 2 (01 Mar 2019), 339–371. https://doi.org/10.1007/s11390-019-1914-z

[12] Milind Kulkarni, Martin Burtscher, Calin Cascaval, and Keshav Pingali. 2009. Lonestar: A suite of parallel irregular programs. In *2009 IEEE International Symposium on Performance Analysis of Systems and Software*. 65–76. https://doi.org/10.1109/ISPASS.2009.4919639

[13] Jérôme Kunegis. 2013. : the Koblenz network collection. In *Proceedings of the 22nd International Conference on World Wide Web* (Rio de Janeiro, Brazil) *(WWW '13 Companion)*. Association for Computing Machinery, New York, NY, USA, 1343–1350. https://doi.org/10.1145/2487788.2488173

[14] Zengxiang Li, Bowen Zhang, Shen Ren, Yong Liu, Zheng Qin, Rick Siow Mong Goh, and Mohan Gurusamy. 2017. Performance Modelling and Cost Effective Execution for Distributed Graph Processing on Configurable VMs. In *2017 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*. 74–83. https://doi.org/10.1109/CCGRID.2017.85

[15] Grzegorz Malewicz, Matthew H. Austern, Aart J.C Bik, James C. Dehnert, Ilan Horn, Naty Leiser, and Grzegorz Czajkowski. 2010. Pregel: a system for large-scale graph processing. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data* (Indianapolis, Indiana, USA) *(SIGMOD '10)*. Association for Computing Machinery, New York, NY, USA, 135–146. https://doi.org/10.1145/1807167.1807184

[16] Robert Ryan McCune, Tim Weninger, and Greg Madey. 2015. Thinking Like a Vertex: A Survey of Vertex-Centric Frameworks for Large-Scale Distributed Graph Processing. *ACM Comput. Surv.* 48, 2, Article 25 (oct 2015), 39 pages. https://doi.org/10.1145/2818185

[17] Sourav Medya, Ludmila Cherkasova, and Ambuj Singh. 2017. Predictive modeling and scalability analysis for large graph analytics. In *2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*. 63–71. https://doi.org/10.23919/INM.2017.7987265

[18] Doekemeijer Niels and Ana Lucia. 2014. *A survey of parallel graph processing frameworks.* Technical Report. Delft University of Technology.

[19] Samuel D. Pollard, Sudharshan Srinivasan, and Boyana Norris. 2019. A Performance and Recommendation System for Parallel Graph Processing Implementations: Work-In-Progress. In *Companion of the 2019 ACM/SPEC International Conference on Performance Engineering* (Mumbai, India) *(ICPE '19)*. Association for Computing Machinery, New York, NY, USA, 25–28. https://doi.org/10.1145/3302541.3313097

[20] Daniel Presser and Frank Siqueira. 2023. Partitioning-Aware Performance Modeling of Distributed Graph Processing Tasks. *International Journal of Parallel Programming* 51, 4 (01 Oct 2023), 231–255. https://doi.org/10.1007/s10766-023-00753-w

[21] Mark Redekopp, Yogesh Simmhan, and Viktor K. Prasanna. 2013. Optimizations and Analysis of BSP Graph Processing Models on Public Clouds. In *2013 IEEE 27th International Symposium on Parallel and Distributed Processing*. 203–214. https://doi.org/10.1109/IPDPS.2013.76

[22] Tianhui Shi, Mingshu Zhai, Yi Xu, and Jidong Zhai. 2020. GraphPi: high performance graph pattern matching through effective redundancy elimination. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis* (Atlanta, Georgia) *(SC '20)*. IEEE Press, Article 100, 14 pages.

[23] Merijn Verstraaten, Ana Lucia Varbanescu, and Cees de Laat. 2018. Mix-and-Match: A Model-Driven Runtime Optimisation Strategy for BFS on GPUs. In *2018 IEEE/ACM 8th Workshop on Irregular Applications: Architectures and Algorithms (IA3)*. 53–60. https://doi.org/10.1109/IA3.2018.00014

[24] Qi Wang, Ludmila Cherkasova, Jun Li, and Haris Volos. 2015. InterSense: Interconnect Performance Emulator for Future Scale-out Distributed Memory Applications. In *2015 IEEE 23rd International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems*. 122–125. https://doi.org/10.1109/MASCOTS.2015.13

[25] Yangzihao Wang, Andrew Davidson, Yuechao Pan, Yuduo Wu, Andy Riffel, and John D. Owens. 2016. Gunrock: a high-performance graph processing library on the GPU. In *Proceedings of the 21st ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming* (Barcelona, Spain) *(PPoPP '16)*. Association for Computing Machinery, New York, NY, USA, Article 11, 12 pages. https://doi.org/10.1145/2851141.2851145

[26] Matei Zaharia, Reynold S. Xin, Patrick Wendell, Tathagata Das, Michael Armbrust, Ankur Dave, Xiangrui Meng, Josh Rosen, Shivaram Venkataraman, Michael J. Franklin, Ali Ghodsi, Joseph Gonzalez, Scott Shenker, and Ion Stoica. 2016. Apache Spark: a unified engine for big data processing. *Commun. ACM* 59, 11 (oct 2016), 56–65. https://doi.org/10.1145/2934664

[27] Jianlong Zhong and Bingsheng He. 2014. Medusa: Simplified Graph Processing on GPUs. *IEEE Transactions on Parallel and Distributed Systems* 25, 6 (2014), 1543–1552. https://doi.org/10.1109/TPDS.2013.111