

A Distributed Memory 1/2 Approximation Algorithm for Vertex-Weighted Matching

BY

Saliyah Shuraym Alotaibi

A Thesis Presented to the

COLLEGE OF GRADUATE AND INTERDISCIPLINARY STUDIES

KING FAHD UNIVERSITY OF PETROLEUM & MINERALS

DHAHRAN, SAUDI ARABIA

In Partial Fulfillment of the
Requirements for the Degree of

MASTER OF SCIENCE

In

COMPUTER SCIENCE

MAY 2024

A DISTRIBUTED MEMORY $1/2$
APPROXIMATION ALGORITHM FOR
VERTEX-WEIGHTED MATCHING

BY

SALIYAH SHURAYM ALOTAIBI

A Thesis Presented to the
DEANSHIP OF GRADUATE STUDIES

In Partial Fulfillment of the Requirements
for the Degree of
MASTER OF SCIENCE

IN

COMPUTER SCIENCE
KING FAHD UNIVERSITY
OF PETROLEUM & MINERALS

Dhahran, Saudi Arabia

MAY 2024

KING FAHD UNIVERSITY OF PETROLEUM & MINERALS
DHAHRAN 31261, SAUDI ARABIA

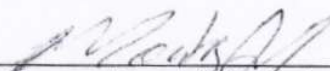
DEANSHIP OF GRADUATE STUDIES

This thesis, written by **SALIYAH SHURAYM ALOTAIBI** under the direction of her thesis adviser and approved by her thesis committee, has been presented to and accepted by the Dean of Graduate Studies, in partial fulfillment of the requirements for the degree of **MASTER OF SCIENCE IN COMPUTER SCIENCE**.

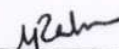
Thesis Committee



Dr. Ahmed Ibrahim Al-Herz (Adviser)



Dr. Moataz Ahmed (Member)



Dr. Md Mahfuzur Rahman (Member)



Dr. Motaz Alfarraj
Department Chairman



Dr. Suliman Al-Homidan
Dean of Graduate Studies

Date



©Saliyah Shuraym Alotaibi
2024

Dedication

To my advisor, Dr. Ahmed Al-Herz

ACKNOWLEDGMENTS

I would like to express my gratitude to God for the life He has blessed me with. I am thankful for the opportunities I have been given and the strength to make the most of them. I am grateful for His guidance through difficult times and for giving me the courage to face whatever comes my way. I want to convey my profound gratitude to my mentor, Dr. Ahmed Al-Herz, for his remarkable encouragement, unwavering support, and extensive knowledge. With his guidance and the considerable time he invested in me and this research, this dissertation came to fruition. To the person who has always been proud of me, no matter what path I choose or which endeavors I pursue, I dedicate this achievement to you, my father. Your unwavering pride and belief in me have been a constant source of motivation and strength. With each step forward, I strive to make you even prouder, knowing that your support is a guiding force propelling me towards success. This accomplishment is a testament to your influence and my mother's encouragement, and the values you have instilled in me. Thank you for being my pillar of strength, and I hope to continue making you proud in all that I do. I thank my thesis committee members, Dr. Moataz Ahmed and Dr. Md Mahfuzur Rahman, for dedicating their time to reviewing my dissertation and offering valuable insights and feedback. My heartfelt thanks go out to my family. My

*siblings Fawziah, Rainz, Areej, Manal, Lama, Bandar, Mohammed, Khalid, and
Fayes have been a continuous source of prayers and encouragement. I am grateful to
my friends, Bshayer, Nouf, and Maha, for their enduring patience and support
throughout this academic journey.*

TABLE OF CONTENTS

ACKNOWLEDGMENTS	vi
LIST OF TABLES	x
LIST OF FIGURES	xi
ABSTRACT (ENGLISH)	xii
ABSTRACT (ARABIC)	xiv
CHAPTER 1 INTRODUCTION	1
1.1 Motivation	2
1.2 Methods for Addressing the MVM Problem	4
1.3 Preliminaries	6
CHAPTER 2 LITERATURE REVIEW	11
2.1 Exact Algorithms	11
2.1.1 Maximum Cardinality Matching	12
2.1.2 Maximum Edge-Weighted Matching	15
2.1.3 Maximum Vertex-Weighted Matching	23
2.2 Approximation Algorithms	28
2.2.1 Edge-Weighted Matching	30
2.2.2 Vertex-Weighted Matching	39
2.3 Gap analysis	46

CHAPTER 3	PARALLEL 1/2 APPROXIMATION ALGORITHM AND HEURISTIC FOR MVM	47
3.1	Shared Memory 1/2 Approximation Algorithm [1]	47
3.2	Distributed Memory 1/2 Approximation Algorithm for MVM	48
3.3	A Heuristic for MVM	57
CHAPTER 4	EXPERIMENTS AND RESULTS	62
4.1	Experimental Setup	62
4.2	Distributed Memory Heuristic and Suitor Algorithms Results	65
CHAPTER 5	CONCLUSION	79
REFERENCES		82
VITAE		90

LIST OF TABLES

2.1	Approximation algorithms for Vertex (MVM) and Edge (MEM) weighted matching.	45
4.1	Input graphs for evaluating the algorithms	63
4.2	Percentage of edges that are between compute nodes where the number of compute nodes is 10	64
4.3	Running time in seconds using different number of cores on shipsec8 graph	69
4.4	Running time in seconds using different number of cores on fe_ocean graph	70
4.5	Running time in seconds using different number of cores on tx2010 graph	71
4.6	Running time in seconds using different number of cores on kron_g500-logn21 graph	72
4.7	Running time in seconds using different number of cores on rgg_n_2_23_s0 Graph	73
4.8	Running time in seconds using different number of cores on nlpkkt200 graph	74
4.9	Running time in seconds using different number of cores on hugebubbles-00010 graph	75
4.10	Running time in seconds using different number of cores on europe graph	76
4.11	Percentage of obtained matching weight and cardinality with respect to the optimal results	77
4.12	Number of communications initiated by nodes when nodes number is 10	77
4.13	Number of messages exchanged by nodes when nodes number is 10 . .	78

LIST OF FIGURES

4.1	Average running time in seconds using different number of cores on shipsec8 graph	69
4.2	Average running time in seconds using different number of cores on fe_ocean graph	70
4.3	Average running time in seconds using different number of cores on tx2010 graph	71
4.4	Average running time in seconds using different number of cores on kron_g500-logn21 graph	72
4.5	Average running time in seconds using different number of cores on rgg_n_2_23_s0 Graph	73
4.6	Average running time in seconds using different number of cores on nlpkkt200 graph	74
4.7	Average running time in seconds using different number of cores on hugebubbles-00010 graph	75
4.8	Average running time in seconds using different number of cores on europe graph	76

THESIS ABSTRACT

NAME: Saliyah Shuraym Alotaibi

TITLE OF STUDY: A Distributed Memory $1/2$ Approximation Algorithm for Vertex-Weighted Matching

MAJOR FIELD: COMPUTER SCIENCE

DATE OF DEGREE: MAY 2024

The maximum vertex-weighted matching (MVM) problem in a graph with weighted vertices involves finding vertex-disjoint edges that optimize the total sum of weights, where the weight of a matching is calculated as the sum of the weights of the vertices at the ends of the matched edges. This type of matching is relevant in various fields including Internet advertising, scientific computing, facility allocation, etc. The MVM problem can be solved both exactly and approximately in polynomial time. Additionally, a parallel approximation algorithm using shared memory was developed to utilize multi-core processors effectively. However, this algorithm does not scale beyond the number of available processor cores. In this thesis, we introduce the first parallel distributed memory $1/2$ -approximation algorithm for this problem. The new algorithm relies on identifying augmenting paths and weight-increasing paths of lengths one or two, both

locally and remotely, continuing this process until no further such paths are found, to achieve the approximation. The time and message complexity using n processors are $O(\Delta^2)$ and $O(\Delta^2 n)$, respectively, where n is the number of vertices and Δ is the maximum degree. Furthermore, we introduce a more efficient heuristic for MVM that guarantees every remote vertex is examined a single time. The time and message complexity of the heuristic, using n processors, are $O(\Delta^2)$ and $O(m)$, respectively, where m is the number of edges. Although the MVM problem can be addressed using a maximum edge weighted approximation (MEM) algorithm, this thesis demonstrates that solving the problem directly using the heuristic is more effective. Experimental evaluations demonstrate that our heuristic surpasses the fastest MEM approximation algorithm, the Suitor algorithm, in terms of both matching quality and computational efficiency.

ملخص الرسالة

الاسم: ساليه شريم العتيبي

عنوان الدراسة: الخوارزمية التقريبية

التخصص: علوم الحاسب الآلي

تاريخ الدرجة العلمية: مايو ٢٠٢٤

مشكلة المطابقة ذات الأوزان القصوى على الرؤوس (MVM) في الرسم البياني الذي يحتوي على رؤوس ذات أوزان، تتضمن إيجاد أضلاع غير مشتركة في الرؤوس بحيث تحقق مجموع أوزان كلي أمثل، حيث يتم حساب وزن المطابقة كمجموع أوزان الرؤوس عند نهايات الأضلاع المطابقة. هذا النوع من المطابقة تكمن أهميته في العديد من المجالات بما في ذلك الإعلانات عبر الإنترنت، والحوسبة العلمية، وتخصيص الموارد، وغيرها. يمكن حل مشكلة MVM في زمن كثير الحدود الدقيق و زمن كثير الحدود التقريبي . بالإضافة إلى ذلك، تم تطوير خوارزمية تقريبية متوازية باستخدام الذاكرة المشتركة للاستفادة بشكل فعال من المعالجات متعددة النوى. ومع ذلك، فإن هذه الخوارزمية لا تتوسع بما يتجاوز عدد نوى المعالج المتاحة. في هذه الرسالة، نقدم أول خوارزمية تقريبية متوازية باستخدام الذاكرة الموزعة بنسبة $1/2$ لحل هذه المشكلة. تعتمد هذه الخوارزمية الجديدة على تحديد المسارات المعززة والمسارات التي تزيد الأوزان بأطوال واحد أو اثنين، سواءً محلياً أو عن بعد، وتستمر في هذه العملية حتى لا يمكن اكتشاف مسارات أخرى . تكون تعقيدات الزمن والرسائل باستخدام n من المعالجات هي $O(\Delta^2)$ و $O(\Delta^2 n)$ على التوالي، حيث n هو عدد الرؤوس و Δ هو الدرجة القصوى.

علاوة على ذلك، صممنا خوارزمية تقريبية أكثر كفاءة لمشكلة MVM حيث أنها تضمن فحص كل رأس بعيد مرة واحدة فقط. بناءً على ذلك فإن تعقيدات الزمن والرسائل للخوارزمية التقريبية، باستخدام n من المعالجات، هي $O(\Delta^2)$ و $O(m)$ على التوالي، حيث m هو عدد

الأضلاع. على الرغم من أن مشكلة MVM يمكن معالجتها باستخدام خوارزمية تقريب الأوزان القصوى للأضلاع (MEM) إلا أن هذا البحث

سيظهر أن حل المشكلة باستخدام الخوارزمية التقريبية MVM أكثر فعالية.

تظهر النتائج التجريبية أن الخوارزمية التقريبية الجديدة تتفوق على أسرع خوارزمية تقريبية Suitor, من حيث جودة المطابقة وكفاءة الحوسبة.

CHAPTER 1

INTRODUCTION

Given a graph $G = (V, E)$ with a set of vertices V , and a set of edges E , a matching M is a subset of edges such that no two edges in M meet at the same vertex. The matching problem is a well-researched area in graph algorithms. Initially, the problem was formulated to optimally allocate personnel to tasks [2]. Since then, matching problems have been explored across a variety of applications. A graph can be unweighted, edge-weighted or vertex-weighted. Hence, there are three basic variations of the matching problem: 1- A Maximum Cardinality Matching (MCM), 2- An Edge-Weighted Matching (MEM) and 3- A Vertex-Weighted Matching (MVM). In this thesis, we study the MVM variant, in particular, we study distributed memory approximation algorithms for the MVM problem. The MVM problem emerged in many applications such as internet advertising [3], switch scheduling [4], scientific computing [5], [6], [7] and resources allocation [8].

1.1 Motivation

To motivate the problem, we present three applications: switch scheduling, scheduling training sessions, and finally addressing the problem of finding the sparsest column-space basis.

An emerging use of MVM is in the management of network switch schedules [9], where a switch is represented by a bipartite graph. Here U is a set of input ports and V is set of output ports. An edge (u, v) exists when a packet from node $u \in U$ must be forwarded to $v \in V$. The weight of $u \in U$ corresponds to the number of packets present at port u , while the weight of $v \in V$ indicates the number of packets requiring delivery to v . The aim of the scheduling policy is to optimize the number of packets transmitted in a single time slot. This is accomplished by identifying a maximum vertex-weighted matching, wherein for each paired edge (u, v) , the packet is directed from port u to port v . It has been shown in [9] that representing network switch scheduling as a vertex-weighted matching yields superior results compared to an edge-weighted matching model. This is because the vertex-weighted approach completely clears all packets from the system in the shortest time possible, assuming no new packets are arriving. Additionally, vertex-weighted models achieve maximum throughput efficiency.

Vertex-weighted matching is applicable for scheduling training sessions for astronauts [10]. Consider T as a set of time intervals, R as a set of astronauts' resource requests, and S as a set of shared resources. Every astronaut submits a subset of time periods during which they are available, along with a subset of required shared

resources. The aim is to schedule the maximum number of astronauts without any conflicts over shared resources. The solution involves two main phases. During the first phase, a graph is created with the following structure: R denotes a set of vertices corresponding to requests, S represents a set of vertices for shared resources, and an edge is drawn between $r \in R$ and $s \in S$ whenever there exists at least one time slot where both r and s can be accommodated at the same time. Determining the maximum vertex-weighted matching yields a set of matching edges and unmatched vertices, referred to as the group G .

During the second phase, a graph is made with vertices from the sets G and T . Here, g represents a vertex belonging to G , and t represents a vertex in T . An edge exists between g and t if g can be assigned to time t . Each group in the graph is given a vertex weight reflecting its priorities. The objective is to find a maximum vertex-weight matching, which aligns groups with time slots in a way that maximizes the overall weight.

Lastly, we will explore an MVM application for addressing the sparsest column-space basis (SCB) problem [11]. Let A be a matrix with k rows and n columns, with $n > k$, and full row rank k . The maximum number of linearly independent columns (or rows) of A is the numerical rank of A . The maximum number of non-zeroes in a diagonal is the *structural rank* of A . The numerical rank of a matrix is less than or equal to its structural rank of A . A *basis* for the column-space of A is a linearly independent set of columns of maximum cardinality. A *sparsest basis* for the column-space of A is a basis with the fewest non-zeroes. An SCB problem can be modeled as an MVM problem. An n by k A matrix can be represented as a bipartite graph

$G = (S, T, E, \phi)$ with weight function $\phi : S \mapsto R_{\geq 0}$, where set S represents the n columns, set T represents the k rows, and there is an edge $(s, t) \in E$ if the entry $A(s, t)$ is nonzero. The vertex weights in S are given by $\phi(s) = k + 1 - d(s)$, where $d(s)$ equals the number of nonzero elements in column s . A matching M in G is equivalent to a subset of nonzero elements in A , such that no two nonzero elements share a column and a row. The subset of non-zeroes that corresponds to matched edges can be positioned on the diagonal of A by permuting the rows and columns of A . Assuming that the numerical rank of A is equal to the structural rank of A and based on the weights specified earlier, the sparsest basis is achieved through a maximum vertex-weighted matching in G .

1.2 Methods for Addressing the MVM Problem

The MVM problem can be reduced to an MEM problem by assigning each edge a weight obtained by summing the weights at its endpoints. While MEM algorithms can be used to address the MVM problem, more efficient algorithms are achieved by tackling the problem directly. The fastest exact MVM algorithm runs in $O(m\sqrt{n} \log n)$ [12], where m is the number of edges and n is the number of vertices. Two recent exact algorithms were proposed by Al-Herz et al. [13], [1] with a time complexity of $O(mn)$ and $O(\Delta mn)$ where Δ is the maximum degree. One exact algorithm is based on searching for augmenting paths until no more augmenting path can be found. The other exact algorithm is based on solving the maximum cardinality matching problem then searching for increasing paths iteratively until no more increasing paths can be

found. Despite the fact that MVM problems can be solved exactly in polynomial time, on nowadays applications which involve graphs with millions and even billions of vertices and edges, exact algorithms are impractical and very slow. In addition, exact algorithms are hard to parallelize, due to their complexities. Recently, many researchers have been proposing approximation algorithms that provide simplicity and high concurrency. Al-Herz et al. [13], [1] proposed approximation algorithms for the MVM problem with approximation ratios of $1/2$ and $2/3$ that are based on augmenting and increasing paths. On a shared memory machine, parallel approximation algorithms for vertex-weighted matching have been designed and implemented [13]. While the shared-memory approximation algorithm achieved a good speed-up, the number of processors that can be utilized by the algorithm is limited. In this thesis, we design a parallel algorithm that computes a $1/2$ approximation for a maximum vertex-weighted matching in distributed memory machines. The approximation algorithm employs a parallel process similar to the shared memory algorithm, specifically by continuously searching for augmenting or increasing paths until none remain. In addition, the distributed algorithm checks a remote neighbor from some vertex $u \in V$ by sending a request message then waiting for a response from the remote vertex. The algorithm terminates when one iteration does not discover any augmenting or increasing paths from every unmatched vertex. The approximation algorithm might result in inefficiencies because of the large volume of messages required to verify remote neighbors. Despite the algorithm guaranteeing a $1/2$ approximation, it is crucial to reduce the number of messages exchanged between compute nodes in distributed memory systems since this significantly impacts the execution time. To resolve this

issue we introduce a practical heuristic for the MVM problem. This issue is addressed by ensuring that remote vertices are checked only once. Also, we propose an order of operations, which is first checking for paths that end with local vertices then looking for paths that end with remote vertices, to reduce number of messages required to be sent between compute nodes.

In the remaining of Chapter 1, we define fundamental concepts and terms in graph theory and matching within the specified scope.

1.3 Preliminaries

In this section we will give basic foundations of matching algorithms. We will introduce definitions, notation, and basic theorems in matching.

Let $G = (V, E)$ be a simple graph where V is a set of vertices and E is a set of edges; each edge is an unordered binary relation on V . We will use the notations (u, v) or uv for an undirected edge with endpoints vertices u and v . The number of vertices and edges are denoted by n and m , respectively. The degree of a vertex u is the number of edges that are incident on u , denoted by $d(u)$. The maximum degree of a vertex in a graph will be denoted by Δ . The set of neighbors of a vertex u will be denoted by $N(u)$. A graph can be unweighted or weights could be associated with its edges or vertices. Weights on vertices can be represented as $\phi : V \mapsto R_{\geq 0}$ for weighted vertices and $\phi : E \mapsto R_{\geq 0}$ for weighted edges. A bipartite graph $G = (S, T, E)$ is a graph in which the set of vertices $S \cup T$ can be partitioned into two disjoint sets S and T , such that no edge joins any two vertices in S , and no edge joins any two vertices in

T . Since edges only connect vertices in S and vertices in T , bipartite graphs do not contain a cycle of odd length. A matching M in a graph $G = (V, E)$ is a subset of edges such that no two edges in the subset meet at the same vertex. A matching can be viewed as an independent set of edges. If u is matched to v in M let $Mate(u) = v$ where $(u, v) \in M$; otherwise $Mate(u) = \text{NULL}$. The cardinality of a matching M is the number of edges in M and is denoted by $|M|$. Based on whether the graph is weighted or not, we have three types of maximum matching problems, which are defined below.

Definition 1.1 *Given a graph $G = (V, E)$, the Maximum Cardinality Matching problem is to find a matching M of maximum cardinality in G .*

Definition 1.2 *Given a graph $G = (V, E, \phi)$ and a weight function $\phi : E \mapsto R_{\geq 0}$, we define the weight of a matching as $\sum_{e \in M} \phi(e)$, the sum of the weights of the matching edges. In the Maximum Edge-Weighted Matching problem we find a matching M of maximum weight in G .*

Definition 1.3 *Given a graph $G = (V, E, \phi)$ and a weight function $\phi : V \mapsto R_{\geq 0}$, we define the weight of a matching as $\sum_{e \in M, e=(u,v)} (\phi(u) + \phi(v))$, the sum of the weights on the endpoints of matching edges. The Maximum Vertex-Weighted Matching problem is to find a matching M of maximum vertex weight in G .*

There are several variants of matching problems. A maximal matching is a matching such that another edge cannot be added to it without violating the matching constraints on the vertices. A perfect matching is a matching in which every vertex

in the graph is matched. A maximum (or minimum) weighted perfect matching is a perfect matching with maximum (or minimum) weight. A maximal matching is not necessarily a maximum cardinality matching but a perfect matching is always a maximum cardinality matching. A graph might not have a perfect matching. Note that a maximum edge-weighted matching is not necessarily a maximum cardinality matching, whereas a maximum vertex-weighted matching can be chosen to be a maximum cardinality matching when the vertex weights are non-negative.

A *path* P in G is a finite sequence of distinct vertices $\{v_i, v_{i+1}, \dots, v_{i+k}\}$ such that $(v_j, v_{j+1}) \in E$ for $i \leq j \leq i+k-1$. The length of a path $|P|$ is the number of edges in the path. A *cycle* is a path with an edge joining its first and last vertices.

A subgraph induced by a subset of vertices X is the graph that includes all edges that join vertices in X . A subgraph induced by a subset of edges F is the graph obtained from the edges in F and all vertices which are its endpoints. We refer to the set of edges of a subgraph H as $E(H)$ and similarly its set of vertices as $V(H)$.

Definition 1.4 *An alternating path P with respect to M is a path whose edges alternate between edges in the matching M and edges not in the matching.*

Definition 1.5 *An augmenting path P with respect to M is an M -alternating path of odd length where the first and last vertices on the M -alternating path P are unmatched.*

Definition 1.6 *In the context of vertex-weighted matching, a weight-increasing path P with respect to M is an M -alternating path of even length where one endpoint is unmatched and the other endpoint is matched such that the weight of the unmatched vertex is heavier than that of the matched vertex.*

Definition 1.7 *A vertex v on a path P is an outer vertex if the number of edges from an unmatched vertex to v is even, and an inner vertex if the number is odd. All unmatched vertices are considered as outer vertices.*

The matching M can be augmented by matching the edges in the symmetric difference $M' = M \oplus P$, which is a matching of cardinality $|M| + 1$. Augmenting paths may be used to find a maximum cardinality matching by repeatedly searching an augmenting path from an unmatched vertex, if it exists. When no augmenting path can be found with respect to M , then the cardinality of the matching is maximum. The following theorem from Berge [14] is a cornerstone to many matching algorithms.

Theorem 1.1 *A matching M in a graph G is a maximum matching if and only if there is no M -augmenting path in G .*

Proof. If there exists an augmenting path P in G with respect to M , then the cardinality of M can be increased by matching the edges in $M \oplus P$, and hence M cannot be a maximum matching. Conversely, assume M is not a maximum matching and M' is a maximum matching. Consider the symmetric difference $M \oplus M'$, which results in $M \oplus M'$ -alternating paths and even cycles. In the case of an alternating path or an alternating cycle of even lengths, both M and M' match the same number of edges. Since $|M'| > |M|$, there must exist an $M \oplus M'$ -alternating path P of an odd length such that $|P \cap M'| > |P \cap M|$ where the first and the last edges belong to M' . Thus, P is augmenting path with respect to M in G . ▮

The rest of the thesis is structured as follows: In Chapter 2, a thorough literature review is conducted, summarizing existing research in the MCM, MEM and MVM

problem. In addition, a critical evaluation of the strengths and weaknesses of previous research is performed, with a specific emphasis on identifying gaps or limitations in the existing literature that the thesis aims to address. In Chapter 3, we will describe the new $1/2$ -approximation algorithm, and the heuristic. We will prove their time and message complexity, and the approximation algorithm correctness. Following this, Chapter 4 describes the experiment setup and results, providing an overview of the methodologies employed and the subsequent analysis of obtained results, shedding light on the heuristic and Suitor algorithm [15] performance. The concluding chapter summarizes the findings and presents the overall conclusion for this work.

CHAPTER 2

LITERATURE REVIEW

This chapter provides a comprehensive review of both exact and approximate algorithms for the maximum vertex weighted problem. Additionally, it covers exact and approximate algorithms for cardinality and edge weighted matching to ensure thoroughness. It is important to note that edge weighted matching algorithms can be adapted to address the maximum vertex weighted matching problem by transforming the graph into an edge weighted graph, where the weights of the vertices at both ends of an edge are summed and assigned to that edge. Historically, the focus has predominantly been on the edge-weighted matching problem due to its broader applicability. However, recent studies have demonstrated that employing specialized algorithms for the vertex-weighted problem yields superior outcomes [13], [1], [16].

2.1 Exact Algorithms

In this section, we provide general overviews of the three varieties of maximum matching. Additionally, we will outline the fundamental combinatorial algorithms for

both cardinality and vertex-weighted matching, as well as primal-dual methods for edge-weighted matching. We omit discussions on scaling and randomized algebraic approaches. [17], [18], [19], [20], [21], which are excluded from this discussion. Initially, we will explore a standard bipartite maximum matching algorithm due to its simplicity, followed by an explanation of the non-bipartite maximum matching algorithm.

2.1.1 Maximum Cardinality Matching

Bipartite Graphs

Consider a bipartite graph $G = (S, T, E)$ with an initially empty matching M . The procedure selects an unmatched vertex s from S and attempts to find an augmenting path P . If such a path P is discovered, M is updated by combining it with P using the operation $P \oplus M$. This process is repeated for every vertex in S . Should the search fail to locate an augmenting path for any unmatched vertex, that vertex is no longer considered in subsequent iterations. The time required to search for an augmenting path is $O(m)$, and with n vertices in the graph, the overall time complexity of the algorithm is $O(mn)$.

Hopcroft and Karp [22] introduced an approach that significantly reduces search iterations by identifying a maximal collection of vertex disjoint shortest augmenting paths in a single $O(m)$ search. The strategy is to organize the searches into stages, where in each stage, a maximal collection of shortest non-overlapping M -augmenting paths is identified. The number of stages can be demonstrated to be limited by $O(\sqrt{n})$. Given that each stage requires $O(m)$ time, the overall time complexity for this method

Algorithm 1 Exact Algorithm for MCM on Bipartite Graphs.

```
1: procedure EXACT-MCM-BIP( $G = (S, T, E)$ )  
  
2:    $M \leftarrow \emptyset$ ;  
  
3:   for all  $s \in S$  do  
  
4:     Starting from  $s$  search for an augmenting path  $P$ ;  
  
5:     if  $P$  is found then  
  
6:        $M \leftarrow M \oplus P$ ;
```

in bipartite graphs is $O(m\sqrt{n})$. For further details and proof, the reader is directed to [22].

Non-bipartite Graphs

Determining the maximum cardinality matching in non-bipartite graphs presents increased complexity. Utilizing the bipartite algorithm mentioned previously on a non-bipartite graph may lead to unsuccessful searches for augmenting paths due to the presence of odd length cycles. These odd length cycles, possessing the maximum number of matched edges, are referred to as blossoms.

Definition 2.1 *Let M be a matching in G and B be an odd set of vertices. B is a blossom if the set of vertices in B form a cycle C and the number of matching edges in C is $(|B| - 1)/2$, that is, $|C \cap M| = (|B| - 1)/2$.*

Edmonds [23] unveiled an innovative approach to address the issues posed by odd cycles. He proposed to compress a blossom into a single super vertex v_B , and substitute the edge set E_1 , which connects vertices within B , with a new set $E_2 = \{(v_B, j) | \exists (i, j) \in E_1, i \in B, j \notin B\}$.

We can now outline the maximum matching algorithm. Consider a graph $G = (V, E)$ with an initially empty matching M . The algorithm selects an unmatched vertex u and initiates a search for an augmenting path. Upon encountering a blossom, the algorithm compresses it, updates G , and recursively continues the search for an augmenting path starting from u . When an augmenting path is located, the algorithm recursively unfolds all blossoms within that path to restore the augmenting path in the original graph. Subsequently, the matching is augmented. This process is repeated until every vertex has been addressed.

Identifying an augmenting path requires $O(m)$ time, similarly, modifying a graph due to the identification of a blossom also takes $O(m)$ time, with a maximum of $O(n)$ recursive shrunken blossoms possible. Consequently, each search within Edmonds' algorithm demands $O(mn)$ time, and with n stages, the overall time complexity reaches $O(mn^2)$. Gabow [24] enhanced the time complexity to $O(n^3)$ by circumventing the explicit shrinking of blossoms and the reconstruction of the graph. To determine the blossom to which a vertex is associated, a blossom membership array is utilized, and upon the detection of a blossom, the internal vertices are added to a queue for subsequent searches, which also comprises the external vertices. An additional enhancement involves the use of a union-find structure to manage blossom memberships [24]. The process of identifying the blossom a vertex is part of requires $O(m\alpha(n, m))$ per stage, leading to a total time complexity of $O(nm\alpha(n, m))$, where α represents the inverse of Ackermann's function. By employing the incremental tree set union algorithm [25], the cost per stage can be minimized to $O(m)$.

Similar to the bipartite scenario, the MCM problem in non-bipartite graphs can

Algorithm 2 Exact Algorithm for MCM on Non-Bipartite Graphs.

```
1: procedure EXACT-MCM( $G = (V, E)$ )  
  
2:    $M \leftarrow \emptyset$ ;  
  
3:   for all  $u \in V$  do  
  
4:     Search for an augmenting path  $P$  starting from  $u$ ;  
  
5:     if a blossom  $B$  is found then  
  
6:       Shrink  $B$  and continue the search for an augmenting path  $P$  from  $u$ ;  
  
7:     if  $P$  is found then  
  
8:       Expand all blossoms recursively that  $P$  goes through;  
  
9:        $M \leftarrow M \oplus P$ ;
```

be addressed in $O(\sqrt{nm})$ time through $O(\sqrt{n})$ phases [26]. Each phase involves identifying a maximal set of augmenting paths with the shortest vertex-disjoint lengths. For an in-depth explanation, please consult [26], [27].

2.1.2 Maximum Edge-Weighted Matching

The MEM problem can be expressed as a linear programming problem, utilizing duality theory. This section explores primal-dual approaches to the MEM problem within both bipartite and non-bipartite graphs. For a more comprehensive exploration of linear programming and primal-dual techniques, see [28].

Bipartite Graphs

The Hungarian method, a primal-dual approach to the MEM problem in bipartite graphs, was initially used for the assignment problem as suggested by Harold W.

Kuhn [2].

Consider the bipartite graph $G = (S, T, E, \phi)$, where the weight function $\phi : E \mapsto R_{\geq 0}$ is defined.

The formulation of the primal-dual for the MEM problem is presented as follows:

Primal problem:

$$\begin{aligned}
& \max && \sum_{st \in E} \phi(st)x(st) \\
& \text{s.t.} && 0 \leq x(st) \leq 1 \quad \text{for } \forall st \in E, \\
& && \sum_{t \in T} x(st) \leq 1 \quad \text{for } \forall s \in S, \\
& && \sum_{s \in S} x(st) \leq 1 \quad \text{for } \forall t \in T.
\end{aligned}$$

Dual problem:

$$\begin{aligned}
& \min && \sum_{s \in S} y(s) + \sum_{t \in T} y(t) \\
& \text{s.t.} && y(s) + y(t) \geq \phi(st) \quad \text{for } \forall st \in E, \\
& && y(s) \geq 0 \quad \text{for } \forall s \in S, \\
& && y(t) \geq 0 \quad \text{for } \forall t \in T.
\end{aligned}$$

Each edge $st \in E$ is associated with a primal variable $x(st)$, which can be either 1 (indicating a matching edge) or 0 (indicating a non-matching edge). Each vertex $v \in S \cup T$ is allocated a dual variable $y(v)$.

Dual variables facilitate the graph search process. Define $\pi(st)$ as a slack variable for each edge $st \in E$, where $\pi(st) = y(s) + y(t) - \phi(st)$. An edge st is considered tight when $\pi(st) = 0$. The optimality of a primal-dual solution is determined if the subsequent complementary slackness conditions are satisfied:

1. $\pi(st) \geq 0, \forall st \in E$.
2. If st is a matching edge, then $\pi(st) = 0$.
3. If s is an unmatched vertex, then $y(s) = 0$.

The fundamental concept of the primal-dual algorithm involves beginning with an initial setup that adheres to conditions 1 and 2, yet breaches condition 3. Consider this initial setup as an For each $s \in S$, $y(s)$ is set to the highest weight of an edge connected to s , and for each $t \in T$, $y(t)$ is set to 0. This configuration meets conditions 1 and 2 but not condition 3. Throughout the algorithm's progression, it systematically reduces the instances where condition 3 is not met, while ensuring conditions 1 and 2 remain satisfied.

The algorithm selects an unmatched vertex s and proceeds to look for an augmenting path originating from s that utilizes tight edges. Should such a path be discovered, it is used to enhance the existing matching. In the absence of tight edges, the dual variables are modified by the smallest positive slack $\delta_{min} = \min(\pi(st))$, ensuring $\pi(st) > 0$ for all $st \in E$:

- $y(s) \leftarrow y(s) - \delta_{min}$.
- $y(t) \leftarrow y(t) + \delta_{min}$.

After the duals are adjusted, new tight edges could cause an augmenting path to be found.

The procedure continues until vertex s is either matched or the dual variable $y(s)$ reaches zero. The efficiency of the algorithm depends on how the minimum

slack δ_{min} is determined and the duals are updated. Employing a simple array search results in a time complexity of $O(n^3)$ [29]. Utilizing a binary heap can lower this to $O(mn \log n)$ [30], [31], while a Fibonacci heap achieves a time complexity of $O(mn + n^2 \log n)$ [32]. The most efficient known exact MEM algorithm for bipartite graphs with integer weights, which achieves a time complexity of $O(\sqrt{nm} \log(W))$, is documented in [33], where W represents the maximum edge weight.

Algorithm 3 Exact Algorithm for MEM on Bipartite Graphs.

```

1: procedure EXACT-MEM-BIP( $G = (S, T, E, \phi)$ )

2:    $M \leftarrow \emptyset$ ;

3:    $\forall s \in S \ y(s) = \max(\phi(st))$ ;

4:    $\forall t \in T \ y(t) = 0$ ;

5:   for all  $s \in S$  do

6:     while  $\exists$  a tight edge do

7:       Search for an augmenting path  $P$  starting from  $s$  using tight edges;

8:       if  $P$  is found then

9:          $M \leftarrow M \oplus P$ ;

10:        break;

11:      else

12:         $\delta_{min} \leftarrow$  minimum positive slack;

13:         $y(s) \leftarrow y(s) - \delta_{min}$ ;

14:         $y(t) \leftarrow y(t) + \delta_{min}$ ;

```

Non-bipartite Graphs

Applying the primal-dual approach of bipartite graphs to non-bipartite problems could lead to inaccurate matchings, as the linear programming solution might include fractional values. Consider a triangle graph and assume the weight of each edge is 1. In this case, $\sum_{uv \in E} x(uv) = 1.5$ is an optimal solution to the linear program, where each $x(uv) = 0.5$. Edmonds [34] introduced a clever method to address this issue by incorporating constraints such as

$$\sum_{uv \in E(B)} x(uv) \leq (|B| - 1)/2, \quad \forall B \in V_{\text{odd}},$$

where V_{odd} is the set of all odd size subsets of V . The primal-dual formulation is as follows:

Primal problem:

$$\begin{aligned} \max \quad & \sum_{uv \in E} \phi(uv)x(uv) \\ \text{s.t.} \quad & 0 \leq x(uv) \leq 1 \quad \text{for } \forall uv \in E \\ & \sum_{uu' \in E} x(uu') \leq 1 \quad \text{for } \forall u \in V \\ & \sum_{uv \in E(B)} x(uv) \leq (|B| - 1)/2 \quad \text{for } \forall B \in V_{\text{odd}}. \end{aligned}$$

Dual problem:

$$\begin{aligned}
\min \quad & \sum_{u \in V} y(u) + \sum_{B \in V_{odd}} z(B)(|B| - 1)/2 \\
\text{s.t.} \quad & y(u) + y(v) + \sum_{B \in V_{odd}: uv \in B} z(B) \geq \phi(uv) \quad \text{for } \forall uv \in E, \\
& y(u) \geq 0 \quad \text{for } \forall u \in V, \\
& z(B) \geq 0 \quad \text{for } \forall B \in V_{odd}.
\end{aligned}$$

The slack is given by $\pi(uv) = y(u) + y(v) + \sum_{B \in V_{odd}: uv \in B} z(B) - \phi(uv)$. The primal-dual solution achieves optimality when the subsequent complimentary slackness conditions are satisfied:

1. $\pi(uv) \geq 0 \forall uv \in E$.
2. If uv is a matching edge, then $\pi(uv) = 0$.
3. If u is an unmatched vertex, then $y(u) = 0$.
4. $z(B) > 0$ for all shrunk blossoms.

Define V_{outer} as the set of outer vertices (vertices at even distance from an unmatched vertex on an alternating path) and V_{inner} as the set of inner vertices (vertices at odd distance from an unmatched vertex on an alternating path), with $V_{non} = V \setminus V_{outer} \cup V_{inner}$. Also, define B_{outer} as the set of outer blossoms and B_{inner} . The procedure operates over $O(n)$ phases. During each phase, the search for an augmenting path involves using tight edges. Should an augmenting path be identified, the matching is then augmented. Upon encountering a blossom B , it is shrunk and $z(B)$

is set to 0. When an inner blossom B is encountered with a dual $z(B)$ of 0, it is expanded. Absence of tight edges prompts an update of the dual variables by δ_{min} to facilitate a new tight edge.

$$\delta_{min} = \min\{\delta_1, \delta_2, \delta_3\}, \text{ where}$$

- $\delta_1 = \min(\pi(uv))$, if $u \in V_{outer}$ and $v \in V_{non}$.
- $\delta_2 = \min(\pi(uv)/2)$, if $u, v \in V_{outer}$.
- $\delta_3 = \min(z(B))$, if $B \in B_{inner}$.

Then we update the duals as follows:

- $y(u) \leftarrow y(u) - \delta_{min}$, if $u \in V_{outer}$.
- $y(u) \leftarrow y(u) + \delta_{min}$, if $u \in V_{inner}$.
- $z(B) \leftarrow z(B) + 2\delta_{min}$, if $B \in B_{outer}$.
- $z(B) \leftarrow z(B) - 2\delta_{min}$, if $B \in B_{inner}$.

If $\delta_{min} = \delta_3$ then expand all inner blossoms with $z(B) = 0$. If $\delta_{min} = \delta_1$ or δ_2 then the search can be expanded or a new blossom can be identified using the tight edges. Similarly to the bipartite algorithm, the most resource-intensive aspect involves determining the minimum slack and modifying the dual variables. The most efficient exact MEM algorithm for non-bipartite graphs with integer weights is presented in [33] and has a time complexity of $O(\sqrt{nm} \log(nW))$, where W is the maximum edge weight.

Algorithm 4 Exact Algorithm for MEM on Non-Bipartite Graphs.

```
1: procedure EXACT-MEM( $G = (V, E, \phi)$ )  
2:    $M \leftarrow \emptyset$ ;  
3:    $\forall u \in V \ y(u) = \max(\phi(e))/2$ ;  
4:   for all  $u \in V$  do  
5:     while  $\exists$  a tight edge  $e = uv$  such that  $u \in V_{outer}$  do  
6:       One of the following steps is done;  
7:       Growing a search tree step, if  $v \in V_{non}$ ;  
8:       Shrinking a blossom step, if  $v \in V_{outer}$  and  $v$  is matched;  
9:       Augmenting a path step, if  $v \in V_{outer}$  and  $v$  is unmatched;  
10:      Expanding a blossom step, if and  $v \in B_{inner}$  and  $z(B) = 0$ ;  
11:      if  $\nexists$  a tight edge then  
12:         $\delta_{min} \leftarrow \min\{\delta_1, \delta_2, \delta_3\}$ ;  
13:         $y(u) \leftarrow y(u) - \delta_{min}$  for  $\forall u \in V_{outer}$ ;  
14:         $y(u) \leftarrow y(u) + \delta_{min}$  for  $\forall u \in V_{inner}$ ;  
15:         $z(B) \leftarrow z(B) + \delta_{min}$  for  $\forall B \in B_{outer}$ ;  
16:         $z(B) \leftarrow z(B) - \delta_{min}$  for  $\forall B \in B_{inner}$ ;
```

2.1.3 Maximum Vertex-Weighted Matching

Bipartite Graphs

Tabatabaee et al. [35] introduced an approach for the design of network switches. This method involves initially calculating a maximum cardinality matching, followed by arranging the unmatched vertices according to their weights in a descending order. For each vertex in this sequence, the algorithm looks for a path that increases the weight. If such a path is identified, the matching is updated; otherwise, the search moves to the subsequent unmatched vertex. Computing a maximum cardinality matching requires $O(m\sqrt{n})$ time, while the search for increasing paths demands $O(mn)$ time, leading to an overall time complexity of $O(mn)$.

Dobrian et al. [36] and Halappanavar [37] introduced a method leveraging bipartite graph structures. The approach involves arranging the vertices by decreasing weight order and splitting the issue into two separate 'one-side-weighted' challenges. Subsequently, each problem is addressed independently by identifying augmenting paths for each vertex. These solutions are then merged into a complete matching through the application of the Mendelsohn-Dulmage Theorem [38]. The process of determining a maximum vertex-weighted matching in a one-side-weighted graph requires $O(mn)$ time, and merging these matchings into a final solution requires $O(n)$ time. Therefore, the overall computational complexity of this method is $O(mn)$.

Algorithm 5 Exact Algorithm for MVM on Bipartite Graphs (Dobrian et. al.).

```

1: procedure MVM-BIP( $G = (S, T, E, \phi)$ )

2:    $M \leftarrow \emptyset; M_S \leftarrow \emptyset; M_T \leftarrow \emptyset;$ 

3:    $Q \leftarrow S;$ 

4:   while  $Q \neq \emptyset$  do

5:      $u \leftarrow \text{heaviest}(Q);$ 

6:      $Q \leftarrow Q - u;$ 

7:     Find an augmenting path  $P$  starting at  $u;$ 

8:     if  $P$  found then

9:        $M_S \leftarrow M_S \oplus P;$ 

10:     $Q \leftarrow T;$ 

11:    while  $Q \neq \emptyset$  do

12:       $u \leftarrow \text{heaviest}(Q);$ 

13:       $Q \leftarrow Q - u;$ 

14:      Find an augmenting path  $P$  starting at  $u;$ 

15:      if  $P$  found then

16:         $M_T \leftarrow M_T \oplus P;$ 

17:     $M \leftarrow \text{MENDELSONDULMAGE}(M_S, M_T, M);$ 

```

Non-bipartite Graphs

Spencer and Mayer [12] developed an exact algorithm for addressing the MVM problem. They arranged the vertices by decreasing weight and converted a non-bipartite graph into a bipartite one by shrinking all blossoms, assigning to each condensed blossom the weight of its lightest vertex. The solution to the bipartite weighted matching problem was then approached by treating it as two separate 'one-side-weighted' problems. These solutions were merged according to the Mendelsohn-Dulmage Theorem [38]. They employed a divide and conquer approach to repeatedly divide the one-side-weighted bipartite graph into smaller segments, finding a maximum cardinality matching in each segment. This strategy segmented the graph into $\log n$ parts, with each matching computation taking no more than $O(m\sqrt{n})$ time, leading to an overall time complexity for the algorithm of $O(m\sqrt{n} \log n)$.

Dobrian et al. [36] and Halappanavar [37] introduced a more straightforward MVM approach. It begins by arranging vertices in descending order based on their weights. Initially, the algorithm works with an empty matching M and tries to pair the heaviest unmatched vertex u . It then looks for another heavy unmatched vertex v , reachable via an augmenting path P . If P is discovered, the matching is increased by taking the symmetric difference between the existing matching M and P , subsequently removing u and v from the pool of unmatched vertices. Should it not locate an augmenting path starting from u , u is then excluded from the unmatched vertices. The process ends once there are no unmatched vertices left. The computational complexity of this method is $O(mn)$.

Al-Herz et al. [13] designed an exact algorithm for MVM called Direct Increasing algorithm. The algorithm has two phases, the first phase of the exact algorithm, as shown in Algorithm 6, starts with an empty matching, then sorts the vertices in non-increasing order of weights and inserts the vertices in a queue Q . In each iteration the algorithm attempts to match a heaviest unmatched vertex u . From u the algorithm searches for an unmatched vertex v that it can reach via an augmenting path P . If it finds P , then the matching is augmented by forming the symmetric difference of the current matching M with P , and the vertices u and v are removed from the set of unmatched vertices. If it fails to find an augmenting path from u , then u is removed from the set of unmatched vertices and inserted into a queue Q' , as we need to search for an increasing path from u in the next phase. When all the unmatched vertices have been processed, the algorithm proceeds to the next phase. In the second phase Q' is the set of unmatched vertices from phase one, and in each iteration, the algorithm attempts to match a heaviest unmatched vertex u . From u , the algorithm searches for a lightest matched vertex v it can reach by an increasing path P such that $\phi(u) > \phi(v)$. If it finds P , then the matching is reversed by forming the symmetric difference of the current matching M with P , and the vertex u is removed from the set of unmatched vertices Q' . If it fails to find an increasing path from u , then u is removed from the set of unmatched vertices. When all the unmatched vertices in Q' have been processed, the algorithm terminates. The time complexity of the Direct-Increasing algorithm is $O(mn + n \log n)$.

Another exact algorithm was proposed by Al-Herz et al. [1]. The algorithm is based on the iterative approach where a vertex may be processed several times. The

Algorithm 6 The Direct-Increasing Exact Algorithm for MVM.

```

1: procedure DIRECT-INCREASING( $G = (V, E, \phi)$ )

2:    $M \leftarrow \phi$ ;

3:    $Q \leftarrow V$ ;

4:   while  $Q \neq \emptyset$  do

5:      $u \leftarrow \text{heaviest}(Q)$ ;

6:      $Q \leftarrow Q - u$ ;

7:     Find an augmenting path  $P$  from  $u$  to an unmatched vertex  $v$ ;

8:     if  $P$  found then

9:        $M \leftarrow M \oplus P$ ;

10:       $Q \leftarrow Q - v$ ;

11:     else

12:        $Q' \leftarrow Q' \cup u$ ;

13:     while  $Q' \neq \emptyset$  do

14:        $u \leftarrow \text{heaviest}(Q')$ ;

15:        $Q' \leftarrow Q' - u$ ;

16:       Find an increasing path  $P$  from  $u$  reaching a lightest matched vertex  $v$ ;

17:       if  $P$  found then

18:          $M \leftarrow M \oplus P$ ;

```

algorithm consists of two phases: in the first phase, a maximum cardinality matching is computed, and, in the second phase, highest gain increasing paths are sought. Phase two is repeated until no increasing path is found. In the first phase of the exact algorithm, as shown in Algorithm 7, the algorithm processes unmatched vertices in an arbitrary order, and in each iteration the algorithm attempts to match an unmatched vertex u . From u , the algorithm searches for an unmatched vertex v that it can reach by an augmenting path P . If it finds P , then the matching is augmented by forming the symmetric difference of the current matching M with P . When all the unmatched vertices have been processed, the algorithm proceeds to the next phase.

In the second phase the algorithm searches for increasing paths. The algorithm processes unmatched vertices in an arbitrary order, and in each iteration it attempts to match an unmatched vertex u . From u , the algorithm searches for a lightest matched vertex v that it can reach by an increasing path P such that $\phi(u) > \phi(v)$. If it finds P , then the matching is reversed by forming the symmetric difference of the current matching M with P , and now v becomes unmatched. The algorithm terminates when it fails to find an increasing path during one pass over the unmatched vertices. The time complexity of the Iterative algorithm is $O(\Delta mn)$.

2.2 Approximation Algorithms

An α -approximation matching algorithm finds a matching that is within a factor of α of the weight of the exact matching. If M_α is a matching that is computed by an α -approximation algorithm and M_{opt} is the optimal matching, then

Algorithm 7 The Iterative Exact Algorithm for MVM.

```
1: procedure ITER( $G = (V, E, \phi)$ )  
  
2:    $M \leftarrow \phi$ ;  
  
3:   for each unmatched vertex  $u$  do  
  
4:     Find an augmenting path  $P$  from  $u$ ;  
  
5:     if  $P$  found then  
  
6:        $M \leftarrow M \oplus P$ ;  
  
7:   repeat  
  
8:     done = true;  
  
9:     for each unmatched vertex  $u$  do  
  
10:      Search for an increasing path  $P$  from  $u$  reaching a lightest matched  
      vertex  $v$ ;  
  
11:      if  $P$  found then  
  
12:         $M \leftarrow M \oplus P$ ;  
  
13:        done = false;  
  
14:   until not done
```

$\sum_{u \in M_\alpha} \phi(u) \geq \alpha \sum_{u \in M_{opt}} \phi(u)$. Approximation algorithms are generally designed for NP-hard problems [39], [40], [41], [42]. However, polynomial time exact matching algorithms are very slow for applications involving massive graphs. and ones that require especially fast computations. For many applications (e.g., big data and scientific computing) a matching needs to be computed fast on massive graphs, and the optimality of the matching is not crucial. This is one motivation for the development of fast approximation algorithms. Another motivation is the necessity for parallel algorithms on massive graphs, when a processor is not able to store the graph in memory; often the exact algorithms do not possess much concurrency but the approximation algorithms do.

A comprehensive review of approximation algorithms for various types of maximum matching problems (including cardinality, edge-weighted matching, vertex-weighted matching, and b -matching) as well as associated minimum edge cover problems (cardinality, edge-weighted, and b -edge cover) is documented in [16]. An earlier review focusing on cardinality and edge-weighted matching approximation algorithms can be found in [43]. This section will provide a concise overview of the latest advancements in approximation algorithms for both edge- and vertex-weighted matching.

2.2.1 Edge-Weighted Matching

Avis [44] introduced a straightforward $1/2$ -approximation method for solving the maximum edge-weighted matching problem. In a graph $G = (V, E, \phi)$, where $\phi : E \mapsto R_{\geq 0}$ is the weight function, the edges are evaluated in descending order of their weights.

The method selects the heaviest edge not yet in the matching M and incorporates it. Subsequently, it removes all edges connected to the vertices of the newly added edge. This procedure is iterated until no edges remain unexamined. Sorting the edges incurs a cost of $O(m \log n)$, resulting in an overall computational time of $O(m \log n + m)$.

Algorithm 8 The Greedy 1/2-Approximation Algorithm for MEM.

```

1: procedure GREEDY( $G = (V, E, \phi)$ )

2:    $M \leftarrow \emptyset$ ;

3:   while  $E \neq \emptyset$  do

4:     Pick a heaviest edge  $uv \in E$ ;

5:      $M \leftarrow M \cup uv$ ;

6:     Delete all edges incident on vertices  $u$  and  $v$ ;

```

The locally-dominant-edge approximation algorithm, introduced by Preis [45], ensures a 1/2-approximation for maximum edge-weighted matching and operates in linear time $O(m)$. For a graph $G = (V, E, \phi)$ where $\phi : E \mapsto R_{\geq 0}$ is the weight function, this algorithm selects an arbitrary non-matching edge $uv \in E$ and examines the edges adjacent to u and v . An edge uv qualifies as locally-dominant if it is heavier or equal in weight compared to any other adjacent edges of u and v . If a heavier adjacent edge ux (or vy) is discovered, where $\phi(ux) > \phi(uv)$ (or $\phi(vy) > \phi(uv)$), the algorithm shifts focus to ux (or vy) and continues recursively until it identifies a locally-dominant edge, which it then includes in the current matching. Subsequently, it eliminates all edges connected to the matched edge. This process is repeated until no edges remain. The algorithm runs in linear time if depth first search is used for locating and removing

dominating edges [16]. Nonetheless, depth-first search is not suitable for a parallel algorithm, so a straightforward alternative that simply inspects the neighboring vertices is employed. This approach runs in $O(m \log \Delta)$, provided that the neighbors of each vertex are sorted.

Algorithm 9 The Locally Dominant Edge 1/2-Approximation Algorithm for MEM.

```

1: procedure LOCAL-DOM( $G = (V, E, \phi)$ )

2:    $M \leftarrow \emptyset$ ;

3:   while  $E \neq \emptyset$  do

4:     Pick an arbitrary edge  $uv \in E$ ;

5:     if  $uv$  is locally dominant edge then

6:        $M \leftarrow M \cup uv$ ;

7:       Delete all edges incident on  $u$  and  $v$ ;

8:     else

9:       Searching from  $u$  and  $v$ , find a locally dominant edge  $xy \in E$ ;

10:       $M \leftarrow M \cup xy$ ;

11:      Delete all edges incident on  $x$  and  $y$ ;

```

Hoepman [46] and Manne et al. [47] designed distributed memory parallel algorithm version of the locally-dominant-edge approximation algorithm.

Drake and Hougardy [48] introduced a more straightforward algorithm utilizing the concept of expanding a path within a graph. This path-growing algorithm ensures a 1/2-approximation for the maximum edge-weighted matching. The procedure is as follows: Consider a graph $G = (V, E, \phi)$, where $\phi : E \mapsto R_{\geq 0}$ is the weight function, and M_1 and M_2 are two initially empty matching sets. The algorithm initiates by

selecting a random unmatched vertex u .

The procedure identifies the heaviest edge $uv \in E$ connected to u and incorporates it into the matching set M_1 . Subsequently, it removes u and its connected edges from G . The method then moves to vertex v , applying similar operations but adds the heaviest edge $vw \in E$ to the set M_2 . This pattern of alternately adding edges to M_1 and M_2 continues. Once all edges are removed, the larger of the two matchings, M_1 or M_2 , is selected as the final result. The computational complexity of this path-growing algorithm is $O(m)$, as it scans each vertex's adjacent edges once.

Algorithm 10 The Path Growing 1/2-Approximation Algorithm for MEM.

```

1: procedure PG( $G = (V, E, \phi)$ )

2:    $M \leftarrow \emptyset$ ;

3:    $M_1 \leftarrow \emptyset$ ;  $M_2 \leftarrow \emptyset$ ;

4:   while  $E \neq \emptyset$  do

5:      $i = 1$ ;

6:     Pick an arbitrary vertex  $u \in V$  of degree at least 1;

7:     while  $u$  has degree at least 1 do

8:       Let  $uv$  be a heaviest neighbor of  $u$ ;

9:        $M_i \leftarrow M_i \cup uv$ ;

10:      Delete all edges incident on  $u$ ;

11:       $u = v$ ;

12:       $i = 3 - i$ 

13:    $M \leftarrow \max \{M_1, M_2\}$ ;

```

Dynamic programming can be applied to identify the best matching edges for each

path developed using the Drake-Hougardy algorithm. Another $1/2$ -approximation method, known as the Global Paths algorithm (GPA), was introduced by Maue and Sanders [49]. This algorithm arranges the edges in a descending order based on their weights. It forms groups of paths and cycles *with even lengths* by evaluating the edges in descending weight order. For each path and cycle, it calculates an optimal weight matching through dynamic programming, removes the matched edges and their neighboring edges. This procedure is repeated until no edges remain. The computational complexity of the GPA is $O(m \log n)$.

Another $1/2$ -approximation method, known as the Suitor algorithm [15], utilizes a proposal-driven technique akin to the traditional stable matching algorithm [50]. In this approach, each vertex u makes a proposal to the heaviest vertex v which has not yet received a superior proposal. If v has previously received a weaker proposal from another vertex w , v rejects this earlier proposal and accepts the one from u . Subsequently, the rejected vertex w must seek a new partner. A match is established when both involved vertices mutually propose to each other. The computational complexity of the Suitor algorithm is $O(\Delta m)$ in the worst-case scenario. The Suitor algorithm's run time improves to $O(m \log \Delta)$ when the neighbors of each vertex are initially sorted.

Two methods have been implemented to surpass approximation ratios of $1/2$. The initial method utilizes short augmenting paths and cycles continuously until meeting a particular threshold. The other method relies on the primal-dual schema. We will outline two $(2/3 - \epsilon)$ -approximation algorithms that are based on the first method. Following this, a $(1 - \epsilon)$ -approximation algorithm utilizing the primal-dual approach

Algorithm 11 The Suitor 1/2-Approximation Algorithm for MEM.

```
1: procedure SUITOR( $G = (V, E, \phi)$ )

2:    $M \leftarrow \emptyset$ ;

3:    $suitor(u) \leftarrow NULL \ \forall u \in V$ ;

4:    $ws(u) \leftarrow 0 \ \forall u \in V$ ;

5:   for each  $u \in V$  do

6:      $x = u$ ;

7:     done = False;

8:     while not done do

9:        $partner = suitor(x)$ ;

10:       $heaviest = ws(x)$ ;

11:      for each  $v \in N(x)$  do

12:        if  $\phi(vx) > heaviest$  and  $\phi(vx) > ws(v)$  then

13:           $partner = v$ ;

14:           $heaviest = \phi(vx)$ ;

15:      done = True;

16:      if  $heaviest > 0$  then

17:         $y = suitor(partner)$ ;

18:         $suitor(partner) = x$ ;

19:         $ws(partner) = heaviest$ ;

20:        if  $y \neq NULL$  then

21:           $x = y$ ;

22:          done = False;
```

will be described.

To describe the $(2/3 - \epsilon)$ -approximation algorithms, we need to explain a 2-augmentation on vertex u . An arm associated with u can be either an edge $\{u, v\}$ or a path $\{u, v, v'\}$, with (u, v) being a non-matching edge and (v, v') a matching edge. The gain from an arm is determined by subtracting the weight of the matching edge from the weight of the non-matching edge; if the arm consists only of a non-matching edge, its gain is equal to the weight of that edge. Two cases need to be addressed:

Case 1) If u is unmatched, identify an arm of u that yields the highest positive gain.

Case 2) If u is matched to u' , ascertain the highest positive gain by evaluating the gains from the following paths or cycles: (1) Alternating cycles of length four that incorporate the edge (u, u') . (2) Alternating paths: this involves locating two vertex disjoint arms of u , with the highest gains P and P' , followed by finding an arm of u' with the highest gain Q . If P and Q are vertex disjoint, then $P \cup (u, u') \cup Q$ forms the highest gain alternating path; otherwise, $P \cup (u, u') \cup Q$ is selected as the highest gain alternating path. We now turn our attention to the $(2/3 - \epsilon)$ -approximation algorithms. The random matching algorithm (RAMA) [51] selects a vertex u at random and conducts a 2-augmentation at u targeting the maximum gain. This process is executed $k = n \frac{1}{3} \ln \frac{1}{\epsilon}$ times. The random ordered matching algorithm (ROMA) [49] randomizes the sequence of vertices, and each vertex u in this new sequence undertakes a 2-augmentation aimed at achieving the highest gain at u . This is carried out over $k = \frac{1}{3} \ln \frac{1}{\epsilon}$ phases. The algorithm concludes if no additional enhancements are found at the end of a phase. Hovland [52] designed and implemented shared-memory parallel ROMA algorithm.

The same technique has been used to achieve $(3/4 - \epsilon)$ -approximation [53], [54] by extending the length of augmentations.

Algorithm 12 The Random $(2/3 - \epsilon)$ -Approximation Algorithm for MEM.

```

1: procedure RAMA( $G = (V, E, \phi)$ ,  $k$ )

2:    $M \leftarrow \emptyset$  (or initialized with any matching);

3:   for  $i = 1$  to  $k$  do

4:     Randomly pick a vertex  $u \in V$ ;

5:      $M \leftarrow M \oplus 2\text{-augmentation}(u)$ ;
```

Algorithm 13 The Random Ordered $(2/3 - \epsilon)$ -Approximation Algorithm for MEM.

```

1: procedure ROMA( $G = (V, E, \phi)$ ,  $k$ )

2:    $M \leftarrow \emptyset$  (or initialized with any matching);

3:   for  $i = 1$  to  $k$  do

4:     Permute the order of vertices;

5:     for each  $u \in V$  in the permuted order do

6:        $M \leftarrow M \oplus 2\text{-augmentation}(u)$ ;

7:     if there is no improvement then

8:       break;
```

A $(1 - \epsilon)$ -approximation algorithm has been introduced [55], [53], utilizing the scaling method, the primal-dual approach to the problem, along with relaxed feasibility and complementary slackness conditions. Define W as the maximum weight of an edge, set $\epsilon' = \Theta(\epsilon)$, $\delta_0 = W/\epsilon'$, $\delta_i = \delta_0/2^i$, $\phi_i(e) = \delta_i \lfloor \phi(e)/\delta_i \rfloor$, and $\gamma = \log 1/\epsilon$. A root blossom refers to a blossom that is not part of a larger blossom. The dual variables y are assigned to vertices and z to blossoms. The edge variable yz for an

edge $e = uv$ is defined as

$$yz(e) = y(u) + y(v) + \sum_{uv \in E(B)} z(B),$$

where B represents a blossom.

The approximation method is composed of $\log W + 1$ levels. For each level i , the conditions of relaxed feasibility and complementary slackness are maintained:

1. $z(B)$ is a non-negative multiple of $\delta_i \ \forall B \in V_{odd}$ and $y(u)$ is a non-negative multiple of $\delta_i/2 \ \forall u \in V$.
2. $z(B) > 0$ for all root blossoms.
3. $yz(e) \geq \phi_i(e) - \delta_i, \ \forall e \in E$.
4. $yz(e) \leq \phi_i(e) + (\delta_j - \delta_i), \ \forall e \in M$, where e becomes a matching edge in scale j and $j \leq i$.

At each scale only eligible edges are considered. At scale i an edge e is eligible if at least one of the following hold:

- e is in a blossom.
- If e is a matching edge, $yz(e) - \phi_i(e)$ is a non-negative integer multiple of δ_i , and $\log \phi(e) \geq i - \gamma$.
- If e is a non-matching edge, $yz(e) = \phi_i(e) - \delta_i$ and $\log \phi(e) \geq i - \gamma$.

The algorithm starts with an empty matching and sets $\delta_0 = \epsilon'W$ and $y(u) = W/2 - \delta_0/2$ for every $u \in V$. During the i -th scale, the algorithm executes four main

actions: it identifies and augments a maximal set of non-overlapping augmenting paths using eligible edges; it shrinks found blossoms and resets their dual values to zero; it updates the dual values of vertices and blossoms that are attained by unmatched vertices during the search; and it expands internal blossoms with dual values of zero. These steps are carried out repeatedly until the duals of unmatched vertices reach $W/2^{i+2} - \delta_i/2$, or becoming zero in the final scale. Following each scale i , except for the last, the dual values of all vertices increase by $\delta_i/2$. The time complexity of this algorithm is $O(m\epsilon^{-1} \log \epsilon^{-1})$.

2.2.2 Vertex-Weighted Matching

Dobrian et al. [36], [37] introduced a 2/3-approximation algorithm for MVM on bipartite graphs. This method involves sorting vertices by decreasing weight and splitting the problem into two problems where only one side has weights. Each problem is addressed separately by limiting augmenting paths to a maximum length of three. Subsequently, the two solutions are merged into a single matching using the Mendelsohn-Dulmage Theorem. The computational complexity of this approach is $O(m + n \log n)$.

In [13], the first 2/3-approximation algorithm for non-bipartite graphs, named the direct approximation algorithm, was presented. Described in Algorithm 16, this method organizes vertices by descending weight and places them into a queue Q . Starting with an empty matching, the algorithm tries to match vertices from Q in the order they appear. It removes each unmatched vertex u from Q , and from vertex u , it looks for the heaviest vertex v that can be reached via an augmenting path

Algorithm 14 $(1 - \epsilon)$ -Approximation Algorithm for MEM.

```
1: procedure SCALING( $G = (V, E, \phi), \epsilon$ )

2:    $M \leftarrow \emptyset$ ;

3:    $\delta_0 \leftarrow \epsilon' W$ ; //  $W$  is the maximum edge weight, and  $\epsilon' = \Theta(\epsilon)$ 

4:    $y(u) \leftarrow W/2 - \delta_0/2$  for all  $u \in V$ ;

5:   for Scale  $i = 0$  to  $\log W$  do

6:     while there are unmatched vertices  $u$  with  $y(u) > W/2^{i+2} - \delta_i/2$  or  $(y(u) \neq$ 
       0 and  $i = \log W)$  do

7:       Find a maximal set  $\mathcal{P}$  of vertex-disjoint augmenting paths using eligible
       edges;

8:        $M \leftarrow M \oplus \mathcal{P}$ ;

9:       Shrink blossoms found in step 6;

10:       $z(B) \leftarrow 0$  for all blossoms  $B$  found in step 6;

11:      Update the duals:

12:         $y(u) \leftarrow y(u) - \delta_i/2$  for all  $u \in V_{outer}$ ;

13:         $y(u) \leftarrow y(u) + \delta_i/2$  for all  $u \in V_{inner}$ s;

14:         $z(B) \leftarrow z(B) + \delta_i$  for all  $B \in B_{outer}$  and  $B$  is a root blossom;

15:         $z(B) \leftarrow z(B) - \delta_i$  for all  $B \in B_{inner}$  and  $B$  is a root blossom;

16:        Expand all inner root blossoms whose  $z(B) = 0$ ;

17:      if  $i < \log W$  then

18:         $\delta_{i+1} \leftarrow \delta_i/2$ ;

19:         $y(u) \leftarrow y(u) + \delta_{i+1}$  for all  $u \in V$ ;
```

Algorithm 15 2/3-Approximation Algorithm for MVM on Bipartite Graphs.

```

1: procedure 2/3-MVM-BIP( $G = (S, T, E, \phi)$ )

2:    $M \leftarrow \emptyset; M_S \leftarrow \emptyset; M_T \leftarrow \emptyset;$ 

3:    $Q \leftarrow S;$ 

4:   while  $Q \neq \emptyset$  do

5:      $u \leftarrow \text{heaviest}(Q);$ 

6:      $Q \leftarrow Q - u;$ 

7:     Find a shortest augmenting path  $P$  of length at most 3 starting at  $u;$ 

8:     if  $P$  found then

9:        $M_S \leftarrow M_S \oplus P;$ 

10:     $Q \leftarrow T;$ 

11:    while  $Q \neq \emptyset$  do

12:       $u \leftarrow \text{heaviest}(Q);$ 

13:       $Q \leftarrow Q - u;$ 

14:      Find a shortest augmenting path  $P$  of length at most 3 starting at  $u;$ 

15:      if  $P$  found then

16:         $M_T \leftarrow M_T \oplus P;$ 

17:     $M \leftarrow \text{MENDELSONDULMAGE}(M_S, M_T, M);$ 

```

no longer than three. If such a path is discovered, the matching is updated with the path leading to the heaviest vertex v , which is then also removed from Q . If an augmenting path of length at most three is not found, the search continues with the next heaviest unmatched vertex, despite the possibility of longer paths being available. The process ends once every vertex has been processed. The approximation algorithm operates with a time complexity of $O(m \log \Delta + n \log n)$. Limiting the length of augmenting paths to one results in a $1/2$ -approximate matching, with a time complexity of $O(m + n \log n)$.

Algorithm 16 The Direct $2/3$ -Approximation Algorithm for MVM.

```

1: procedure 2/3-DIR( $G = (V, E, \phi)$ )

2:    $M \leftarrow \emptyset$ ;

3:    $Q \leftarrow V$ ;

4:   while  $Q \neq \emptyset$  do

5:      $u \leftarrow \text{heaviest}(Q)$ ;

6:      $Q \leftarrow Q - u$ ;

7:     Let  $v$  denote a heaviest unmatched vertex reachable from  $u$  by an
        augmenting path  $P$  of length at most three;

8:     if  $P$  is found then

9:        $M \leftarrow M \oplus P$ ;  $Q \leftarrow Q - v$ ;
```

Al-Herz et al. [1] proposed another $1/2$ -approximation algorithm referred to as the iterative approximation algorithm. Initially, the algorithm either begins with no matching or uses a pre-computed one. It repeatedly executes the following steps as long as it identifies either an augmenting or increasing path. The vertices are

processed in a random sequence. For each unmatched vertex u , the algorithm looks for an adjacent unmatched vertex v . If such a vertex v is found, they are matched. Otherwise, the search extends to finding an increasing path of length two leading to the vertex v with the least weight. Upon discovering such a path, the matching is adjusted by inverting this path. The computational complexity of the approximation algorithm is $O(\Delta m)$, where Δ represents the highest degree. Limiting the lengths of augmenting and increasing paths to 3 and 4, respectively, results in a $2/3$ -approximate matching. This iterative approach was also applied in the development of a shared memory parallel algorithm that attains $1/2$ and $2/3$ -approximation ratios [1]. A more comprehensive description of this parallel algorithm will be discussed in chapter 3.

Algorithm 17 The Initialized Iterative 1/2-Approximation Algorithm for MVM.

```
1: procedure 1/2-ITE( $G = (V, E, \phi)$ )

2:    $M \leftarrow \emptyset$ ;

3:   for each  $u \in V$  do

4:     Find from  $u$  an unmatched neighbor  $v$ ;

5:     if  $v$  is found then

6:        $M \leftarrow M \cup (u, v)$ ;

7:   repeat

8:     done = true;

9:     for each  $u \in V$  do

10:      if  $u \notin M$  then

11:        Find from  $u$  an unmatched neighbor  $v$ ;

12:        if  $v$  is found then

13:           $M \leftarrow M \cup (u, v)$ ;

14:          done = false;

15:        else

16:          Find a highest gain increasing path  $P'$  from  $u$  s.t.  $|P'| = 2$ ;

17:          if  $P'$  is found then

18:             $M \leftarrow M \oplus P'$ ;

19:            done = false;

20:  until not done
```

Table 2.1: Approximation algorithms for Vertex (MVM) and Edge (MEM) weighted matching.

Algorithm	Problem	Approx. ratio	Time Complexity	Parallel
Direct	MVM	1/2	$O(m + n \log n)$	NO
Direct	MVM	2/3	$O(m \log \Delta + n \log n)$	NO
Iterative	MVM	1/2	$O(\Delta m)$	shared memory
Iterative	MVM	2/3	$O(\Delta^2 m)$	shared memory
Greedy	MEM	1/2	$O(m \log n)$	NO
Locally Dominant	MEM	1/2	$O(m \log \Delta)$	shared and distributed memory
Path Growing	MEM	1/2	$O(m)$	NO
Global Paths algorithm	MEM	1/2	$O(m \log n)$	NO
Suitor	MEM	1/2	$O(m \log \Delta)$	shared and distributed memory
RAMA and ROMA	MEM	$2/3 - \epsilon$	$O(m \log \epsilon^{-1})$	shared memory
Scaling	MEM	$1 - \epsilon$	$O(m \epsilon^{-1} \log \epsilon^{-1})$	NO

2.3 Gap analysis

Upon examining existing literature which is summarized in Table 2.1, it was discovered that:

1. Although distributed memory parallel MEM approximation algorithms exist for addressing the MVM problem, none specifically tackle the problem by leveraging the structure of the vertex-weighted graph. Furthermore, it is unknown whether such a targeted algorithm would yield better practical results.
2. No existing distributed memory parallel MEM or MVM approximation algorithms can achieve an approximation ratio greater than $1/2$.
3. There are no MEM or MVM approximation algorithms capable of achieving a $1/2$ -approximation ratio in $O(m \log \Delta)$ without the need to sort edges or vertices.

This thesis proposes to resolve the first issue by developing a new distributed memory parallel algorithm that directly addresses the MVM problem. Furthermore, a more effective heuristic is suggested to provide a sub-optimal solution to the MVM problem. Moreover, the practical effectiveness of this heuristic will be assessed by implementing it and experimentally comparing its execution time and matching weights with those of the Suitor algorithm.

CHAPTER 3

PARALLEL $1/2$

APPROXIMATION

ALGORITHM AND HEURISTIC

FOR MVM

In this chapter we will describe the shared memory approximation algorithm, then introduce the new distributed memory approximation algorithms, and demonstrate its time and message complexity as well as its correctness. Lastly, we will present a heuristic for vertex weighted matching and establish its time and message complexity.

3.1 Shared Memory $1/2$ Approximation Algorithm [1]

In this section, the shared memory approximation algorithm will be described. First, as shown in Algorithm 18, the algorithm starts with an empty matching. The

algorithm iterates over the unmatched vertices repeatedly in parallel until in one iteration no augmenting or increasing paths are found. Once a thread finds one such path, it locks vertices on the path such that no other thread should augment or update the matching on these vertices since augmenting and increasing paths discovered by two threads could overlap. If a thread cannot acquire all locks needed, then it releases all locks and proceeds to search from other unmatched vertices. There is an implicit synchronization barrier across all threads at the end of each iteration of the for loop. Now we discuss how the test and set locks are employed in the parallel algorithm. The lock is free if its value is zero and, not free otherwise. If a thread reads a value of zero for a lock, then it has atomic access to the lock variable and can set it to a nonzero value. If a thread reads a nonzero value for a lock, then it is unavailable. We allow an augmenting path joining the vertices u and v to augment the matching only if $u < v$, and in this way we prevent two threads from attempting to acquire locks and augmenting the same path from opposite directions. For matching an edge (u, v) on an augmenting path, we lock its lower-numbered endpoint and then the other endpoint; for an increasing path with edges (u, v_i) and (v_i, v_j) , we need to lock u first then the lower-numbered endpoint of the matching edge (v_i, v_j) .

3.2 Distributed Memory 1/2 Approximation Algorithm for MVM

In this section, we will present and describe our new distributed memory parallel algorithm, derived from Algorithm 18.

Algorithm 18 Shared Memory Parallel 1/2-Approximation Algorithm for MVM.

```

1: procedure SHARED-MEMORY 1/2-APPROX( $G = (V, E, \phi)$ )
2:    $M \leftarrow \emptyset$ ;
3:   repeat
4:     done = true;
5:     for all unmatched  $u \in V$  do in parallel
6:       find an aug. path  $P$  of length 1;
7:       if  $P$  is found and  $u < v$  then
8:         if LOCK( $P, u$ ) = true then
9:            $M \leftarrow M \oplus P$ ; done = false;
10:        release all locks;
11:       else
12:         continue;
13:     else
14:       find a increasing path  $P'$  of length 2;
15:       if  $P'$  is found then
16:         if LOCK( $P', u$ ) = true then
17:            $M \leftarrow M \oplus P'$ ; done = false;
18:         release all locks;
19:       else
20:         continue;
21:   until done = true

```

Algorithm 19 Locking procedure.

```
1: procedure LOCK( $P, u$ )
2:   if lock( $u$ )=0 then
3:     lock( $u$ )=1;
4:     if  $P$  is an augmenting path then
5:       if lock( $v$ )=0 then
6:         lock( $v$ )=1;
7:       else
8:         release any locks;
9:       return false;
10:    let  $e = (v_i, v_j)$  be a matching edge on  $P'$ 
11:    min_ $v = \min(v_i, v_j)$ ;
12:    if lock(min_ $v$ )=0 then
13:      lock(min_ $v$ )=1;
14:    else
15:      release any locks;
16:    return false;
17:    if  $P$  has not changed then
18:      return true;
19:    else
20:      release any locks;
21:      return false;
22:  else
23:    return false;
```

The initial step of the algorithm involves the distribution of the input graph $G = (V, E, \phi)$ among the compute nodes. Each compute node manages a local sub-graph $G' = (V', E', \phi)$. The set of vertices V' is composed of two sets: V_l , representing the vertices that are local, and V_r , representing the vertices that are remote. The set V_r is not exhaustive; instead, it includes vertices that are neighbors to the local vertices in a compute node. This selective subset of remote vertices called ghost vertices. The edges in the local node denoted by E' and ϕ corresponds to the weights function.

In the distributed memory 1/2-approximation algorithm, communication between compute nodes is facilitated by exchanging messages. We have four types of messages: 1) request for matching (REQM), 2) request for unmatching (REQU), 3) accept matching (ACC), 4) reject matching (REJ).

The algorithm is presented in Algorithm 20. While the algorithm is hybrid (shared and distributed) in the description we will omit calling locking procedure to avoid repetitions. The algorithm starts with an empty matching. The following steps are repeated as long as the matching is updated in one iteration. For each unmatched local vertex u , the algorithm searches for an augmenting path of length one or an increasing path of length two. If the augmenting path end is local then the edge is matched. Furthermore, if the increasing path end, say v , is local, then the matching is updated by reversing the increasing path. If the augmenting path end is a remote vertex then the algorithm creates a REQM message, sends it to the remote node and waits for a response. If an accept message is received then it matches u with v . If it is a reject message then the algorithm proceeds to examine the next neighbor. Lastly, if the increasing path that ends with a remote vertex v , then the algorithm updates

the matching, generates a **REQU** message, and sends it to the remote node. If a node receives a **REQM** message with a ghost vertex u (source) and a local v (destination), then if v is unmatched then (u, v) is matched and sends an **ACC** message to the remote node. If $u, v, \text{Mate}(v)$ is an increasing path and $\text{Mate}(v)$ is local, then the algorithm unmatched $\text{Mate}(v)$, matches (u, v) and sends an **ACC** message to the remote node. If $u, v, \text{Mate}(v)$ is an increasing path and $\text{Mate}(v) = v'$ is a ghost vertex, then the algorithm unmatched v' , matches (u, v) and sends an **ACC** message to the remote node and a **REQU** message to the node having v' . If $u, v, \text{Mate}(v)$ is not an increasing path then the algorithm sends a **REJ** message to the remote node. If the received message is a **REQU** message with a ghost vertex u (source) and a local v (destination), then the algorithm unmatched v .

The algorithm terminates when in one iteration no augmenting paths or increasing paths can be found and all attempts to match a ghost vertex are rejected.

Time Complexity

Theorem 3.1 *The time complexity of the distributed memory approximation algorithm using n processors is $O(\Delta^2)$, where Δ is the maximum degree.*

Proof. In each iteration of the for loop, each processor chooses an unmatched vertex u and examine all neighbors of u to find an augmenting path or increasing path. Thus, for a vertex u , the algorithm examines at most a number of vertices equal to $d(u)$. Since a vertex can be unmatched at most $O(\Delta)$ times and $d(u)$ is upper bounded by Δ , a total of $O(\Delta^2)$ vertices are searched in parallel.

I

Algorithm 20 Distributed Memory 1/2 Approximation Algorithm for MVM

```
1: procedure DISTRIBUTED MEMORY 1/2 APPROX. ( $G' = (V', E', \phi)$ )
2:    $M \leftarrow \emptyset$ ;
3:   repeat
4:     done = true;
5:     for unmatched  $u \in V_l$  in parallel do
6:       find  $v$  (an end of aug. or inc. path  $P$ )
7:       if  $v$  is local unmatched neighbor then
8:          $M \leftarrow M \oplus P$ ;
9:       else if  $v$  is a local end of an increasing path then
10:         $M \leftarrow M \oplus P$ ;
11:       else if  $v$  is a remote neighbor then
12:         send a REQM message to the node having  $v$ ;
13:         if an ACC message is received then
14:            $M \leftarrow M \oplus P$ ;
15:         else if a REJ message is received then
16:           continue;
17:       else if  $v$  is a remote end of an increasing path then
18:         send a REQU message to the node having  $v$ ;
19:          $M \leftarrow M \oplus P$ ;
20:       if a message  $m$  is received then
21:         HANDLE_MSG_PROC( $m$ );
22:       if  $M$  is updated then
23:         done = false;
24:   until done = true
```

Algorithm 21 Procedure for handling a received message

```
1: procedure HANDLE_MSG_PROC( $m$ )
2:   let  $u$  be the remote vertex and  $v$  the local vertex in  $m$ ;
3:   if  $m$  is a REQM message then
4:     if  $v$  is unmatched then
5:        $M \leftarrow M \oplus P$ ;
6:       send an ACC message
7:     else if  $v$  is matched and  $\text{Mate}(v)$  is lighter than  $u$  then
8:       if  $\text{Mate}(v)$  is a remote vertex then
9:         send a REQU message to the node having  $\text{Mate}(v)$ ;
10:       $M \leftarrow M \oplus P$ ;
11:      send an ACC message
12:     else if  $v$  is matched and  $\text{Mate}(v)$  is heavier than  $u$  then
13:       send a REJ message;
14:   else if  $m$  is a REQU message then
15:     unmatched  $v$ 
```

Message complexity

Theorem 3.2 *The number of messages transmitted by n processors when employing the distributed $1/2$ -approximation algorithm is $O(\Delta^2 n)$.*

Proof. Each processor is assigned a vertex u , and for each neighbor that needs to be examined the processor sends a request for matching message. Therefore, in each iteration of the for loop, each processor sends at most $O(d(u))$ request messages or $O(d(u))$ reject messages. Since a vertex can be unmatched at most $O(\Delta)$ times and $d(u)$ is upper bounded by Δ , the total number of transmitted messages is $O(\Delta^2 n)$ by n processors. |

Proof of Correctness

In this sub-section, we will prove that the distributed approximation algorithm computes 1/2-approximation for MVM.

Lemma 3.1 *Let M_{opt} be an optimal matching and M_A be an approximate matching in a graph $G = (V, E, \phi)$ such that there does not exist an augmenting path of length one and an increasing path of length two with respect to M_A . For every M -unmatched vertex u that is matched in M_{opt} , there is one distinct M -matched vertex that is at least as heavy as u .*

Proof. Let O be the set of vertices that are matched in M_{opt} but unmatched in M_A . Consider the symmetric difference of the two matchings $M_{opt} \oplus M_A$, which consists of vertices with degrees 0, 1 or 2. Hence this subgraph is a union of alternating paths and alternating cycles. In each alternating cycle all vertices are matched in both M_{opt} and M_A ; hence we need consider only alternating paths. Each vertex $u \in U$ has degree one in the subgraph, and therefore must be an end point of an alternating path. Since there does not exist an augmenting path of length one, the length of each alternating path is at least two. Since there are no weight-increasing paths of length two, the vertex at distance two from u on the symmetric difference path must be at least as heavy as u . Note that the symmetric difference paths are vertex disjoint, and, thus, for every unmatched vertex u , we have found one distinct matched vertex in M_A that are at least as heavy as u . ■

Theorem 3.3 *Let $G = (V, E, \phi)$ be a graph and $\phi : V \mapsto R_{\geq 0}$ a weight function.*

Then the distributed approximation algorithm computes a $1/2$ - approximation for the maximum vertex-weighted matching problem on G .

Proof. Let M_A , M_{opt} and O all be as defined in the Lemma 3.1, and consider paths in the symmetric difference between M_A and M_{opt} . Each vertex $u \in O$ must be an endpoint of an alternating path in the symmetric difference. Thus

$$\phi(M_{opt}) \leq \phi(M_A) + \phi(O).$$

By Lemma 3.1, we have $\phi(O) \leq \phi(M_A)$. Hence,

$$\begin{aligned} \phi(M_{opt}) &\leq \phi(M_A) + \phi(M_A) \\ \Rightarrow \phi(M_A) &\geq \frac{1}{2}\phi(M_{opt}). \end{aligned}$$

I

3.3 A Heuristic for MVM

In this section, we will present and describe our new distributed memory parallel heuristic. To surpass the performance of distributed memory parallel MEM approximation algorithms, it is essential to utilize the matching on a vertex-weighted graph. It is important to note that for the weights of two neighboring vertices to contribute to the matching weight, these vertices must be matched, though the specific matching edge involved is not important. Therefore, our approach prioritizes the order in which neighbors are considered, focusing first on local vertices followed by remote vertices. Additionally, to enhance the algorithm's efficiency, remote vertices are examined and checked a single time.

Like the $1/2$ -approximation algorithm, the initial step of the algorithm is distributing the input graph $G = (V, E, \phi)$ among the nodes. Each node manages a local sub-graph $G' = (V', E', \phi)$.

As in the distributed memory $1/2$ -approximation algorithm, communication between compute nodes is facilitated by exchanging messages, however, sending messages is done in batches rather than individually. Each compute node maintains separate buffers for sending (SB) and receiving (RB) messages from neighboring compute nodes. We have two types of messages: 1) request for matching (REQ), 2) reject matching (REJ).

In each compute node, we have a queue Q that has the local vertices in arbitrary order and an empty queue Q' to maintain vertices that are unmatched during one iteration to be processed again in the next iteration. When Q' is empty at the end of

an iteration, the algorithm terminates.

The heuristic is presented in Heuristic 22. The heuristic is hybrid (shared and distributed) in the description we will omit calling locking procedure to avoid repetitions. The heuristic starts with an empty matching. The following steps are repeated as long as Q' is not empty. For each unmatched vertex u in Q , the heuristic searches for an augmenting path of length one that ends with a local vertex v of a heaviest weight. If it is found, then u and v are matched. If not, then it searches for an increasing path of length two that reaches a local lightest vertex v . If an increasing path is found, then the matching is updated by reversing the increasing path and v is inserted in Q' . If no augmenting path or increasing path within local vertices are found, then the heuristic looks for an augmenting path of length one that reaches a heaviest ghost neighbor v and if it is found it updates the matching and generates a REQ message, and adds it to the appropriate send buffer SB. Lastly, if no ghost neighbor is found, then the heuristic looks for an increasing path that ends with a lightest ghost vertex v , and if it is found it updates the matching and generates a REJ message, and adds it to the appropriate send buffer SB. Once all vertices in Q have been processed, the heuristic dispatches all collected messages from the iteration to their corresponding remote nodes. Note that in Algorithms 18 and 20, an augmenting or increasing path is sought by examining the neighboring vertices sequentially from the first neighbor until one such path is discovered, each time a vertex become unmatched, however, in an efficient distributed algorithm we cannot afford checking remote vertices repeatedly. So, the heuristic will check remote neighbors only one time while local neighbors are checked repeatedly.

Afterward, the compute node remains idle until it gets incoming messages stored in the receive buffer RB from any of the remote nodes. Once it receives an incoming message from a remote node, it begins processing the message immediately. For a REQ message containing a ghost vertex u (source) and a local vertex v (destination), if v is currently unmatched, then (u, v) becomes a matched edge. Otherwise, if the path $u, v, \text{Mate}(v)$ is an increasing path, then the heuristic unmatcheds $\text{Mate}(v)$, adds it to Q' , and matches (u, v) . If the path $u, v, \text{Mate}(v)$ is not an increasing path, then the heuristic creates a REJ message and places it in the send buffer. For a received REJ message with a ghost vertex u (source) and a local vertex v (destination), the heuristic unmatcheds v and adds it to Q' .

Time Complexity

Theorem 3.4 *The time complexity of the heuristic using n processors is $O(\Delta^2)$, where Δ is the maximum degree.*

Proof. In each iteration of the **for** loop, each processor chooses an unmatched vertex u and examines all neighbors of u . Once a remote neighbor is selected as an end of a potential augmenting or increasing path it will not be examined again in the future. Since a vertex can be unmatched at most $O(d(u))$ times, a total of $\sum_{i=1}^{d(u)} i = O(d(u)^2)$ which is upper bounded by $O(\Delta^2)$. ■

Message complexity

Theorem 3.5 *The number of messages transmitted by n processors when employing the heuristic is $O(m)$.*

Algorithm 22 Heuristic for MVM

```
1: procedure MVM HEURISTIC( $G' = (V', E', \phi)$ )
2:    $M \leftarrow \emptyset$ ;  $Q \leftarrow V_l$ ;  $Q' \leftarrow \emptyset$ ;
3:   all buffers are initially empty;
4:   while ( $Q$ )  $\neq \emptyset$  do
5:     for unmatched  $u \in Q$  in parallel do
6:       find  $v$  (an end of aug. or inc. path  $P$ ) in the following order
7:       1- a heaviest local unmatched neighbor
8:       2- a lightest increasing path that ends in a local vertex
9:       3- a heaviest unmatched ghost neighbor
10:      4- a lightest increasing path that ends in a ghost vertex
11:      if  $v$  is local unmatched neighbor then
12:         $M \leftarrow M \oplus P$ ;
13:      else if  $v$  is a local end of an increasing path then
14:        add  $v$  to  $Q'$ ;
15:         $M \leftarrow M \oplus P$ ;
16:      else if  $v$  is a remote neighbor then
17:         $M \leftarrow M \oplus P$ ;
18:        add a REQ message to the send buffer  $SB$ ;
19:      else if  $v$  is a remote end of an increasing path then
20:         $M \leftarrow M \oplus P$ ;
21:        add a REJ message to the send buffer  $SB$ ;
22:      REM_MSG_PROC( $SB, RB, Q'$ );
23:       $Q \leftarrow Q'$ 
24:       $Q' \leftarrow \emptyset$ 
```

Algorithm 23 Procedure for remote messaging

```
1: procedure REM_MSG_PROC( $SB, RB, Q'$ )
2:   for all neighboring nodes  $r$  do
3:     send messages from buffer  $SB$  to node  $r$ ;
4:   for all messages  $m \in RB$  do
5:     let  $u$  be the remote vertex and  $v$  the local vertex in  $m$ ;
6:     if  $m$  is a REQ message then
7:       if  $v$  is unmatched then
8:          $M \leftarrow M \oplus P$ ;
9:       else if  $v$  is matched and  $\text{Mate}(v)$  is lighter than  $u$  then
10:        add  $\text{Mate}(v)$  to  $Q'$ 
11:         $M \leftarrow M \oplus P$ ;
12:       else if  $v$  is matched and  $\text{Mate}(v)$  is heavier than  $u$  then
13:        add a REJ message to the send buffer  $SB$ ;
14:       else if  $v$  is an end of increasing path then
15:        unmatch  $v$  add it to  $Q'$ 
16:     else if  $m$  is a REJ message then
17:       unmatch  $v$  and add it to  $Q'$ 
```

Proof. Each processor is assigned a vertex u , and each one sends at most $d(u)$ request messages and $d(u)$ reject messages. Hence, the total number of transmitted messages is $\sum_{u \in V} 2d(u) = 4m = O(m)$. |

CHAPTER 4

EXPERIMENTS AND RESULTS

4.1 Experimental Setup

Machine Specifications

We conducted experiments using ALFAHD high performance computing machines which use Intel Xeon CPU E5-2680 v4 – 14 Cores @ 2.40GHz (Dual-processor). We had an access to maximum five nodes. The nodes in this system use a CPU architecture with 2×14 cores, forming a multicore processor setup. Each core has its own cache levels with a 32 KB L1 cache for quick access to frequently used data. The second tier consists of a 256 KB L2 cache per core to enhance data retrieval efficiency. Additionally, there is a shared L3 cache of 35 MB per socket, which serves all the cores in the CPU.

Code Specifications

The code for the experiments was developed using C++. Intel® MPI was used for inter-node communication, while OpenMP was used for intra-node multi-threading.

Table 4.1: Input graphs for evaluating the algorithms

Graph	Vertices	Edge	Max degree	Average degree
shipsec8	114919	3269240	131	56.90
fe_ocean	143437	409593	6	5.71
tx2010	914231	2228136	121	4.87
kron_g500logn21	2097152	91040932	213,904	117.92
rgg_n_2_23_s0	8388608	63501393	40	15.14
nlpkkt200	16240000	215992816	27	26.60
hugebubbles-00010	19458087	29179764	3	2.99
europe	50912018	54054660	13	2.12

We used the mpich-4.0.0 compiler with -O3 -OpenMP flags. The operating system is Red Hat Enterprise Linux Server release 7.3 (Maipo) – kernel 3.10.0514.el7.x86_64. We set the stack size for each OpenMP thread to 16 megabytes. We controlled the allocation of threads and pinning a thread to a core by setting KMP_AFFINITY to fine and compact and KMP_HW_SUBSET to 1t. This parameter means that threads are distributed across the available cores compactly and each thread is pinned to one core. This parameter directly affects performance by determining how threads are assigned to cores.

Problems Specifications:

Our experiment involved eight graphs from the Suite Sparse Matrix Collection¹, and the problems we used for the experiments are listed in Table 4.1. The maximum vertex count is around 50 million and the maximum edge count is roughly 215 million.

For each graph, we list the maximum, average vertex degrees. The average vertex degrees range from 2.12 to 117.92. Therefore, the graphs exhibit a variety of degree distributions. Integer weights of vertices were generated uniformly at random in the range $[1, 1000]$. We partition a graph by a straightforward partitioning method. Each partition contains $|V|/np$ vertices, and processor i owns the i th $|V|/np$ vertices, where np is the number of compute nodes. Table 4.2 shows the percentage of edges that are between compute nodes where the number of nodes is 10. The percentages range from 0.17% to 100%, so the partitions are various with respect to the number edges that connect remote vertices.

Table 4.2: Percentage of edges that are between compute nodes where the number of compute nodes is 10

Graphs	(edges between compute nodes/ $ E $)*100
shipsec8	12.73%
fe_ocean	17.43%
tx2010	2.78%
kron_g500logn21	90.00%
rgg_n_2_23_s0	0.17%
nlpkkt200	100.00%
hugebubbles00010	13%
europe	8.13%

Experiment Specifications

In our research, we conducted multiple experiments to compare between the new distributed memory heuristic for MVM and Suitor algorithm. The new $1/2$ -approximation algorithm was excluded from the comparison because its high message complexity which negatively impacts the overall execution time.

To assess the algorithms' performance, we used a varying number of nodes and processors across different experiments. For instance, we used a range of nodes from 2 to 10 to distribute vertices over 28, 42, 56, 70, 84, 98, 112, 126 and 140 processors. The results reported were gathered by executing 7 trials on each graph, recording the minimum, maximum, and average time, as well as the matching weight and cardinality.

4.2 Distributed Memory Heuristic and Suitor Algorithms Results

In this section, we compare our distributed memory heuristic for MVM with the Suitor algorithms, which is considered the fastest approximation algorithm for MEM problem, in term of running time [16]. We compare the algorithms' running time and scalability. Also, we compare weights and the matching edges cardinality.

Overall, the new heuristic outperforms the Suitor algorithm on every graph except nlpkkt200, where their performances are comparable. From this point onward, the average running times will be used in the comparisons of the running time. The results are shown in Tables 4.3-4.10. On fe_ocean graph, the MVM heuristic runs 2.4

times faster than the Suitor algorithm on average across all core counts, and 4.1 times faster when using 28 cores. On the nlpkkt200 graph, the new heuristic outperforms by being 1.01 times faster on average, and its speed advantage extends to 1.1 times when utilizing 70 cores. In the case of the tx2010 graph, the running time is 1.8 times faster on average, peaking at a 3.2 times improvement with 56 cores. Additionally, for the kron_g500-logn21 graph, the new heuristic exhibits a 4.0 times faster performance across all core counts, reaching a 4.5 times faster execution compared to the Suitor algorithm when 28 cores are employed. The new heuristic shows a 1.5 times speed improvement on average and is 1.8 times faster than the Suitor algorithm on the europe graph when utilizing 28 cores. On the rgg_n_2_23_s0 graph, the new algorithm is 9.1 times faster on average and achieves a 16.4 times speed increase on 28 cores. For the Shipsec8 graph, the heuristic obtains matchings 2.4 times faster on average and 3.8 times faster with 56 cores. Finally, in the case of the hugebubbles-00010 graph, the new algorithm demonstrates an average speed improvement of 1.1 times and achieves a 1.4 times increase in performance when utilizing 28 cores. The new heuristic recorded a maximum execution time of 127 seconds on the nlpkkt200 graph, while the Suitor algorithm took 139 seconds on the same graph. The high running times can be attributed to the vast number of edges in the graph, with each edge connecting two remote vertices, thereby increasing the number of transmitted messages. The heuristic reaches a minimum execution time of 0.001 seconds on the fe_ocean graph, while the Suitor algorithm has a minimum time of 0.01 seconds on the same graph. Referring to tables 4.11 and 4.12, on average, the new heuristic transmits 0.63 fewer messages than the Suitor algorithm when using 10 nodes which contributes to the heuristic's

improved speed. Additionally, the frequency of nodes initiating communications is lower by 0.93 times with the Suitor algorithm.

Based on the recorded running times, the new heuristic outperforms the Suitor algorithm, one reason for the fast running time is that the heuristic matches local vertices first then the remote vertices if no local neighbors are available for matching. On the other hand, Suitor algorithm sends a proposal to the heaviest available neighbor, regardless of whether that neighbor is local or remote.

Now we will discuss scalability of the new heuristic. Although the MVM algorithm is faster on all graphs, it did not scale up to 140 cores very well on most of the graphs. The speedup is measured as the running time using 28 cores over the running time using 140 cores. The data shows that the new heuristic scaled better on large graphs. On `nlpkkt200` graph, the heuristic achieved a speedup of 3.3 and Suitor achieved a speedup of 3.6, where the ideal speedup is 5. Suitor performed slightly better in term of scalability on most graphs and the biggest gap is on `rgg_n_2_23_s0` graph where Suitor speedup was 3.6 and the MVM heuristic was 1.6. In addition, the largest speedup was 3.6 on graph `rgg_n_2_23_s0` which was obtained by Suitor algorithm. Both algorithms, on `fe_ocean` and `tx2010`, exhibited slower performance when executed with 140 cores.

Next, we proceed to compare the matching weights and cardinalities when utilizing 140 cores. As shown in Table 4.11, the new heuristic obtained a higher matching weights and cardinality on all graphs with the exception of `nlpkkt200`, which showed marginal differences favoring the Suitor algorithm. On average, the new heuristic achieved a 1.7% greater matching weight and a 4.2% increase in cardinality compared

to the outcomes obtained by the Sutor algorithm. For the nlpkkt200 graph, the Sutor algorithm achieved 99.5% and 97.7% of the optimal weight and cardinality, respectively, whereas the heuristic algorithm reached 99.4% and 97.3% of the optimal weight and cardinality. In euro graph, the most significant difference can be shown, which benefits the new heuristic, is approximately 5% in terms of weight and 11% in terms of the number of edges. On average across all graphs, the new heuristic algorithm attained 97.2% of the optimal weight and 95.0% of the optimal cardinality, whereas the Sutor algorithm achieved 95.6% and 91.3% of the optimal weight and cardinality, respectively.

Table 4.3: Running time in seconds using different number of cores on shipsec8 graph

Cores	Statistics	28	42	56	70	84	98	112	126	140
New Heuristic	min	0.014	0.013	0.011	0.017	0.027	0.032	0.029	0.032	0.034
	max	0.074	0.044	0.026	0.040	0.059	5.666	9.738	8.309	14.802
	avg	0.027	0.022	0.017	0.027	0.026	0.037	0.034	0.039	0.037
	stdev	0.022	0.011	0.005	0.008	0.004	0.008	0.016	0.005	0.008
Suitor Algorithm	min	0.070	0.059	0.057	0.053	0.052	0.059	0.056	0.052	0.061
	max	0.082	0.077	0.071	0.078	0.065	0.074	0.070	0.070	0.077
	avg	0.073	0.068	0.065	0.063	0.060	0.067	0.061	0.062	0.070
	stdev	0.004	0.007	0.005	0.009	0.004	0.006	0.004	0.006	0.006

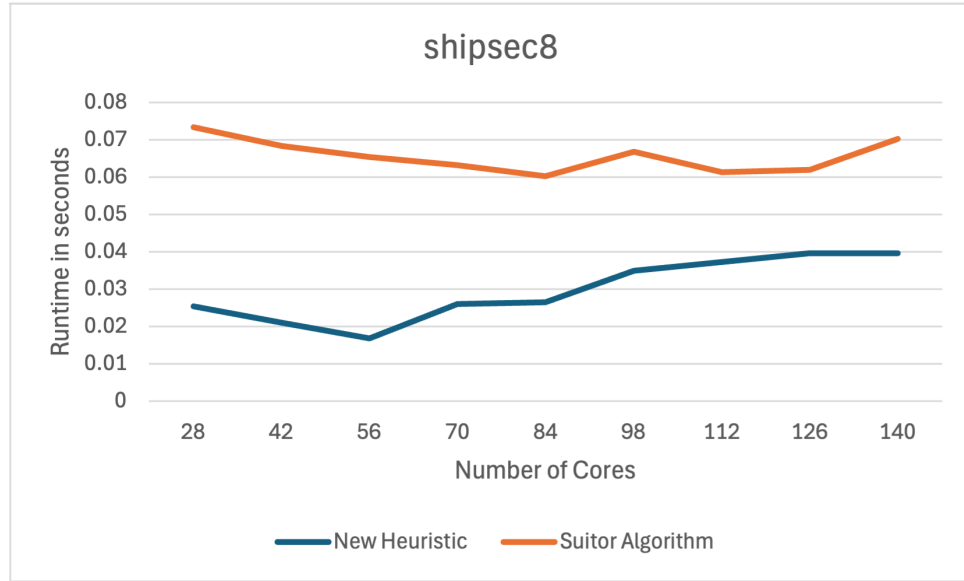


Figure 4.1: Average running time in seconds using different number of cores on shipsec8 graph

Table 4.4: Running time in seconds using different number of cores on fe __ocean graph

Cores	Statistics	28	42	56	70	84	98	112	126	140
New Heuristic	min	0.002	0.005	0.004	0.008	0.008	0.061	0.026	0.016	0.016
	max	0.004	0.006	0.013	0.021	0.019	0.094	0.092	0.094	0.085
	avg	0.003	0.006	0.009	0.014	0.014	0.078	0.068	0.068	0.067
	stdev	0.001	0.001	0.003	0.005	0.004	0.014	0.022	0.027	0.026
Suitor Algorithm	min	0.008	0.009	0.011	0.011	0.022	0.068	0.055	0.059	0.052
	max	0.013	0.015	0.058	0.028	0.138	0.117	0.098	0.139	0.094
	avg	0.010	0.012	0.029	0.019	0.084	0.084	0.071	0.095	0.072
	stdev	0.002	0.002	0.018	0.007	0.041	0.017	0.018	0.027	0.014

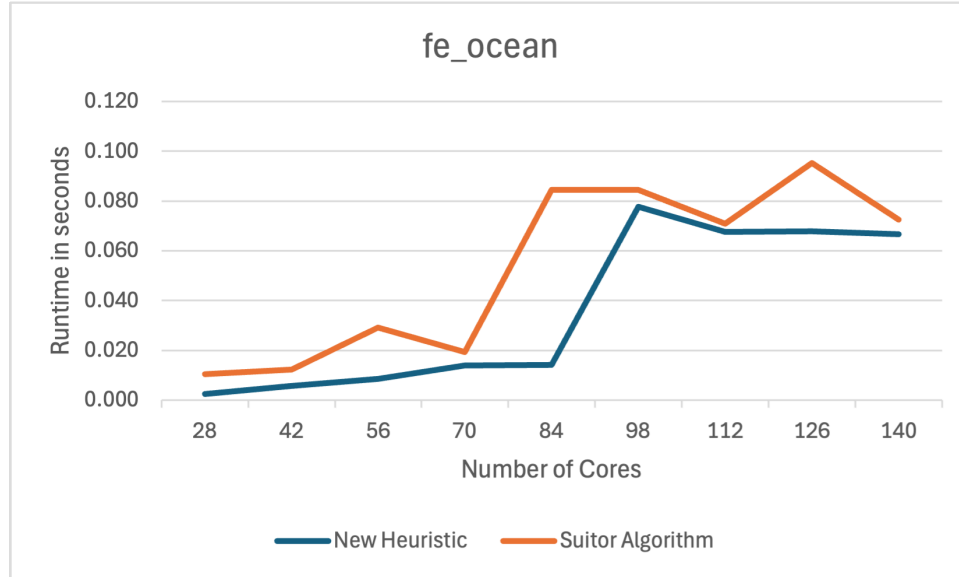


Figure 4.2: Average running time in seconds using different number of cores on fe __ocean graph

Table 4.5: Running time in seconds using different number of cores on tx2010 graph

Cores	Statistics	28	42	56	70	84	98	112	126	140
New Heuristic	min	0.008	0.007	0.006	0.008	0.009	0.021	0.011	0.033	0.019
	max	0.084	0.043	0.020	0.028	0.022	0.180	0.464	0.247	0.178
	avg	0.025	0.018	0.010	0.015	0.015	0.109	0.223	0.155	0.117
	stdev	0.027	0.012	0.004	0.007	0.005	0.063	0.165	0.087	0.069
Suitor Algorithm	min	0.032	0.025	0.025	0.019	0.027	0.032	0.027	0.059	0.031
	max	0.051	0.050	0.045	0.032	0.059	0.341	0.771	0.359	0.325
	avg	0.037	0.032	0.032	0.026	0.033	0.204	0.240	0.205	0.194
	stdev	0.006	0.008	0.007	0.004	0.011	0.130	0.254	0.107	0.121

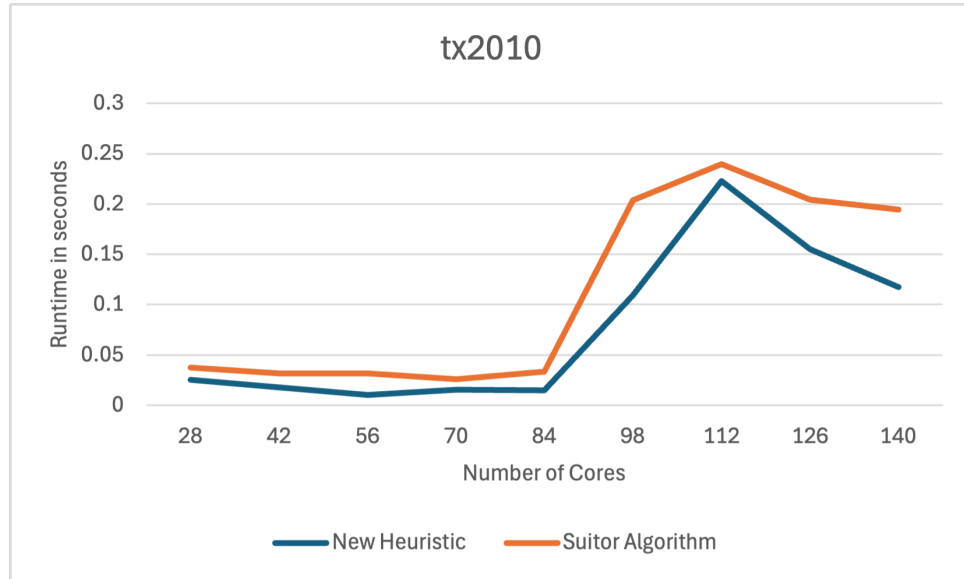


Figure 4.3: Average running time in seconds using different number of cores on tx2010 graph

Table 4.6: Running time in seconds using different number of cores on kron_g500-logn21 graph

Cores	Statistics	28	42	56	70	84	98	112	126	140
New Heuristic	min	0.543	2.897	2.884	2.818	2.532	2.455	2.350	2.277	2.087
	max	3.717	4.125	4.036	3.349	2.897	3.151	2.622	2.576	2.414
	avg	2.362	3.322	3.310	3.048	2.777	2.751	2.486	2.370	2.240
	stdev	0.972	0.412	0.424	0.185	0.128	0.265	0.093	0.111	0.131
Suitor Algorithm	min	9.879	11.853	11.542	11.172	10.329	9.935	9.254	9.098	8.682
	max	11.859	14.158	13.193	12.573	12.287	11.165	10.542	9.870	9.870
	avg	10.561	13.212	12.507	11.987	11.288	10.712	10.067	9.478	9.107
	stdev	0.656	0.960	0.677	0.536	0.611	0.455	0.546	0.258	0.473

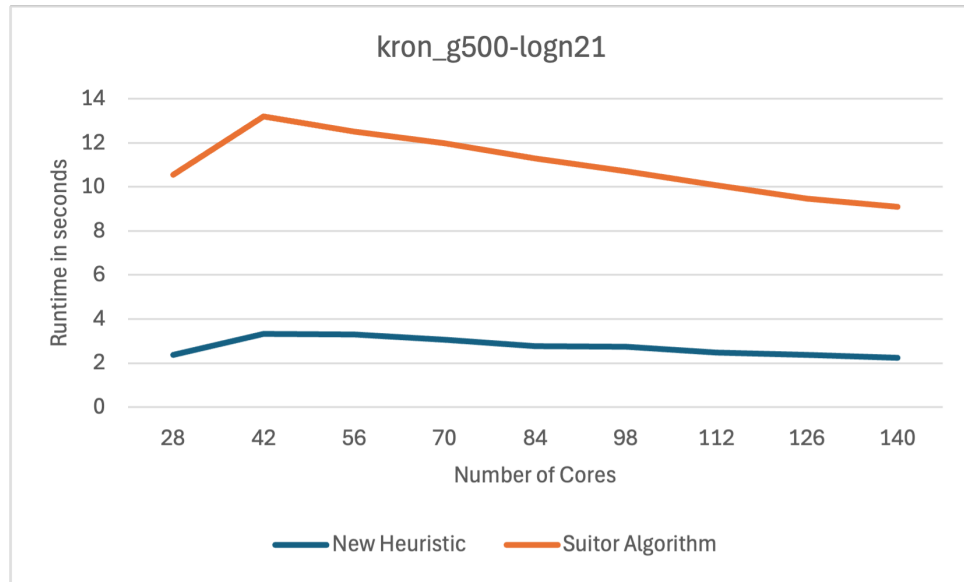


Figure 4.4: Average running time in seconds using different number of cores on kron_g500-logn21 graph

Table 4.7: Running time in seconds using different number of cores on rgg_n_2_23_s0 Graph

Cores	Statistics	28	42	56	70	84	98	112	126	140
New Heuristic	min	0.082	0.060	0.048	0.038	0.046	0.036	0.041	0.031	0.039
	max	0.100	0.092	0.091	0.149	0.142	0.099	0.112	0.216	0.072
	avg	0.088	0.084	0.067	0.081	0.099	0.081	0.066	0.097	0.055
	stdev	0.007	0.011	0.016	0.037	0.037	0.026	0.024	0.082	0.016
Suitor Algorithm	min	1.192	0.903	0.623	0.584	0.493	0.419	0.383	0.373	0.322
	max	1.709	1.065	1.096	0.827	0.744	0.662	0.559	0.495	0.488
	avg	1.446	0.984	0.868	0.686	0.618	0.523	0.507	0.435	0.398
	stdev	0.198	0.058	0.199	0.088	0.092	0.077	0.062	0.056	0.051

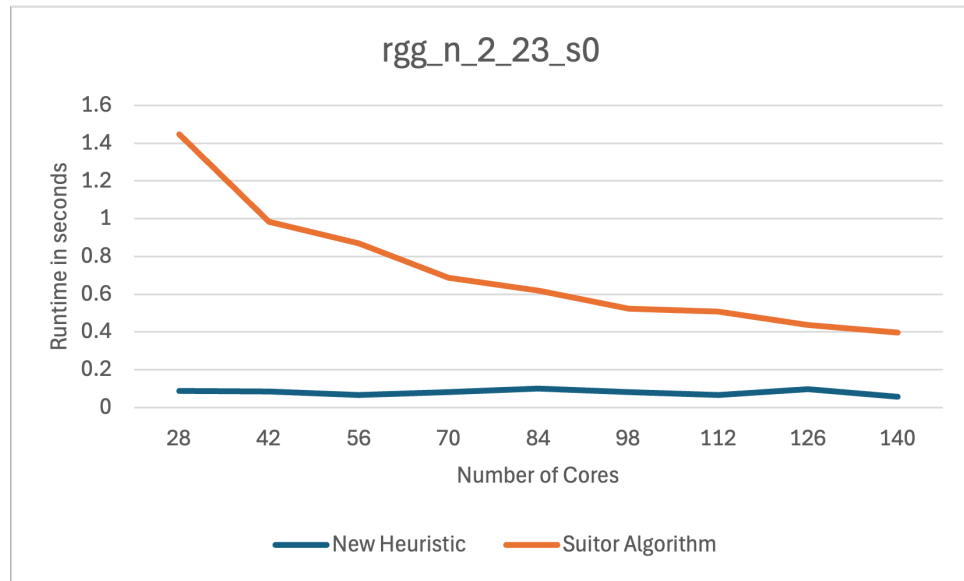


Figure 4.5: Average running time in seconds using different number of cores on rgg_n_2_23_s0 Graph

Table 4.8: Running time in seconds using different number of cores on nlpkkt200 graph

Cores	Statistics	28	42	56	70	84	98	112	126	140
New Heuristic	min	85.726	95.790	75.675	46.613	48.106	47.431	39.474	42.537	35.135
	max	127.141	100.094	82.627	68.289	57.772	49.635	44.511	42.884	36.912
	avg	117.531	98.082	79.646	62.309	55.524	48.613	43.328	42.691	36.272
	stdev	14.563	1.726	2.865	7.950	3.655	0.772	1.750	0.123	0.655
Suitor Algorithm	min	119.095	90.095	75.697	56.350	53.675	49.040	39.734	39.808	33.740
	max	139.329	104.153	84.417	84.410	59.460	54.705	51.595	42.282	37.065
	avg	126.683	94.907	78.474	67.058	55.436	50.965	43.601	40.952	35.362
	stdev	8.010	5.082	3.007	10.035	1.948	2.120	4.150	0.855	1.402

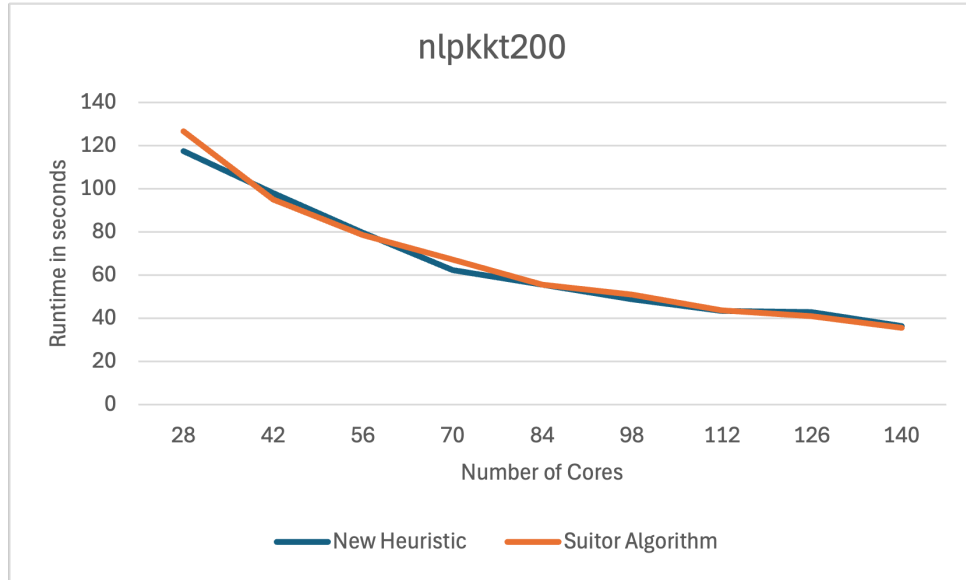


Figure 4.6: Average running time in seconds using different number of cores on nlpkkt200 graph

Table 4.9: Running time in seconds using different number of cores on hugebubbles-00010 graph

Cores	Statistics	28	42	56	70	84	98	112	126	140
New Heuristic	min	1.612	2.335	2.203	2.218	2.175	1.970	1.956	1.825	1.753
	max	1.996	2.736	2.938	2.861	2.527	2.448	2.320	2.382	1.991
	avg	1.769	2.542	2.621	2.553	2.373	2.278	2.125	2.069	1.849
	stdev	0.133	0.135	0.233	0.236	0.108	0.165	0.121	0.230	0.075
Suitor Algorithm	min	2.125	2.464	2.332	2.479	2.404	2.043	2.029	2.027	1.623
	max	2.910	2.912	3.027	3.445	2.653	2.626	2.334	2.267	2.234
	avg	2.502	2.683	2.750	2.794	2.530	2.341	2.176	2.157	1.977
	stdev	0.266	0.162	0.239	0.361	0.088	0.220	0.125	0.097	0.199

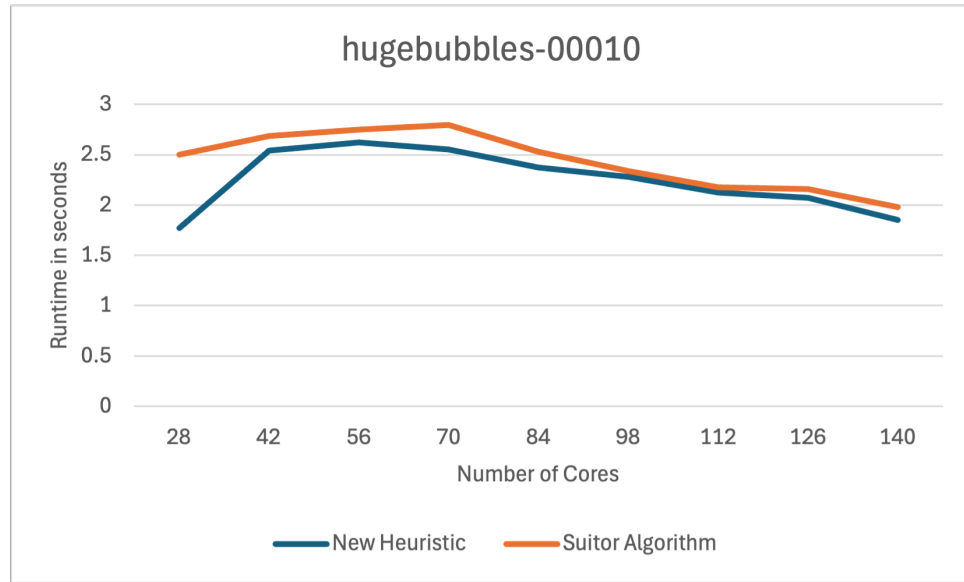


Figure 4.7: Average running time in seconds using different number of cores on hugebubbles-00010 graph

Table 4.10: Running time in seconds using different number of cores on europe graph

Cores	Statistics	28	42	56	70	84	98	112	126	140
New Heuristic	min	0.654	0.583	0.447	0.434	0.485	0.416	0.436	0.397	0.396
	max	1.188	0.800	0.721	0.735	0.642	0.649	0.601	0.795	0.591
	avg	0.818	0.654	0.616	0.582	0.543	0.536	0.523	0.523	0.503
	stdev	0.185	0.074	0.092	0.112	0.058	0.086	0.050	0.128	0.067
Suitor Algorithm	min	1.183	0.958	0.806	0.659	0.681	0.532	0.534	0.441	0.530
	max	2.000	1.153	2.003	0.929	1.008	1.013	0.813	0.746	0.636
	avg	1.477	1.031	1.248	0.813	0.795	0.736	0.648	0.626	0.571
	stdev	0.262	0.075	0.453	0.092	0.106	0.162	0.116	0.105	0.041

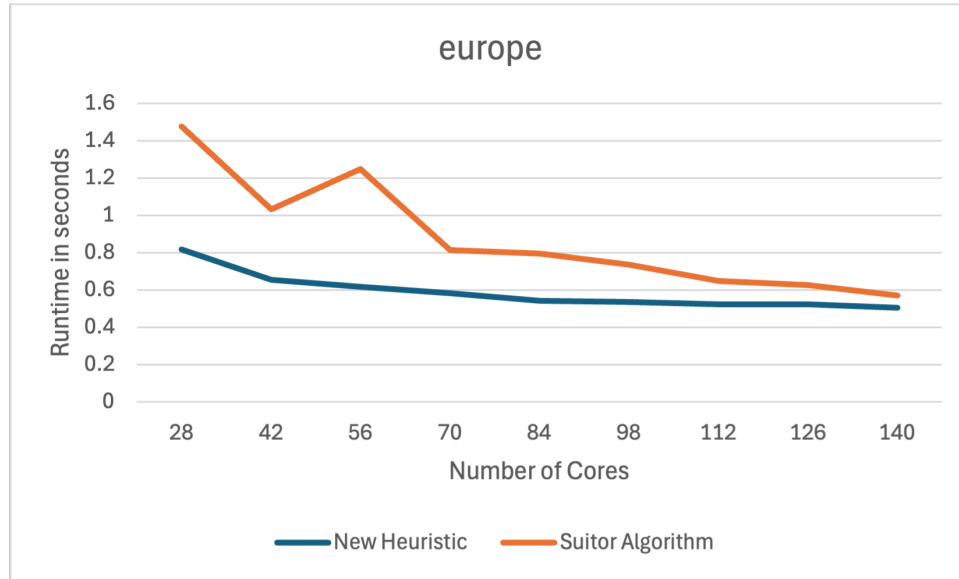


Figure 4.8: Average running time in seconds using different number of cores on europe graph

Table 4.11: Percentage of obtained matching weight and cardinality with respect to the optimal results

Graph	New Heuristic	New Heuristic	Suitor Algorithm	Suitor Algorithm
	matching weight	cardinality percentage	matching weight	cardinality percentage
shipsec8	99.98%	99.45%	99.95%	98.89%
fe_ocean	99.35%	96.87%	97.73%	91.76%
tx2010	96.08%	90.79%	93.59%	86.49%
kron_g500logn21	87.24%	86.05%	85.92%	84.31%
rgg_n_2_23_s0	99.90%	98.99%	99.44%	96.15%
nlpkkt200	99.39%	97.34%	99.47%	97.74%
hugebubbles00010	97.78%	93.93%	95.59%	88.42%
europe	97.86%	96.26%	93.24%	86.42%

Table 4.12: Number of communications initiated by nodes when nodes number is 10

Graph	New Heuristic	Suitor Algorithm
shipsec8	15979	12748
fe_ocean	3418	2202
tx2010	4803	5979
kron_g500-logn21	227561	339858
rgg_n_2_23_s0	6505	4413
nlpkkt200	9034	11408
hugebubbles-00010	3911	2494
europe	3428	2202

Table 4.13: Number of messages exchanged by nodes when nodes number is 10

Graph	New Heuristic	Suitor Algorithm
shipsec8	242470	832345
fe_ocean	4906387	8018432
tx2010	71794	112057
kron_g500-logn21	28498309	45247114
rgg_n_2_23_s0	65085	207329
nlpkkt200	472864539	419890646
hugebubbles-00010	18954198	22483696
europe	4906261	8018432

CHAPTER 5

CONCLUSION

We have designed the first distributed memory parallel $1/2$ approximation algorithm for vertex-weighted matching and prove its time and message complexity, and correctness. Our algorithm is an extension of the shared memory $1/2$ -approximation algorithm. The new approximation algorithm searches for either an augmenting path of length one or an increasing path of length two. If a path is found and the other end of the path is local then update the matching. If the other end of the path is a remote vertex then send a message to the remote node and wait for either an accept message or a reject one. Termination of the algorithm occurs when one iteration fails to find any augmenting or increasing path from all unmatched vertices. The algorithm time complexity is $O(\Delta^2)$ and the message complexity is $O(\Delta^2 n)$ using n processors which is considerable. Given that the approximation algorithm sends a large number of messages, we introduce a more efficient heuristic for the MVM problem. In a distributed memory environment, it is crucial to reduce inter-node communications for faster execution. The new heuristic accomplishes this by ensuring that a request from a vertex u is sent to a remote neighbor v only once. Also, we propose an order of

operations, which is first checking for paths that end with local vertices then looking for paths that end with remote vertices, to decrease the number of messages required to be sent among nodes. The proposed ordering gives advantage over using an MEM approximation algorithm to solve the MVM problem. The heuristic time complexity is $O(\Delta^2)$ and the message complexity is $O(m)$ using n processors.

We have implemented and compared the new heuristic with the Sutor algorithm, using eight graphs with various numbers of vertices, edges, average and max degrees.

With the exception of a few instances, our heuristic demonstrates superior performance compared to the Sutor algorithm across all examined graphs when utilizing 28 to 140 cores on 10 nodes, delivering markedly improved results. This highlights the algorithm's superiority when working in parallel and confirms the advantage of solving the problem directly. In particular, the new heuristic could be 16.4 times faster than Sutor algorithm on one graph. The new heuristic scaled better on large graphs with more than 91 millions edges while on smaller graph it did not scale well.

Next, we outline future work and pathways stemming from our findings and limitations.

- One limitation of our work is the limited number of cores. We would like to see the performance of the new heuristic using higher number of cores.
- Investigate the possibility of designing an efficient distributed memory 2/3-approximation algorithm for MVM problem.
- In this thesis we used straightforward method for partitioning graphs. We will consider more advanced method to minimize the crossing edges and evaluate

the performance.

REFERENCES

- [1] A. Al-Herz and A. Pothén, “A parallel 2/3-approximation algorithm for vertex-weighted matching,” in *Proceedings of the SIAM Workshop on Combinatorial Scientific Computing*. SIAM, 2020, to appear.
- [2] H. W. Kuhn, “The Hungarian method for the assignment problem,” *Naval Research Logistics Quarterly*, vol. 2, no. 1-2, pp. 83–97, 1955.
- [3] A. Mehta, “Online matching and ad allocation,” *Foundations and Trends in Theoretical Computer Science*, vol. 8, no. 4, pp. 265–368, 2012.
- [4] V. Tabatabaee, L. Georgiadis, and L. Tassiulas, “QoS provisioning and tracking fluid policies in input queueing switches,” *IEEE/ACM Trans. Netw.*, vol. 9, no. 5, pp. 605–617, 2001.
- [5] T. F. Coleman and A. Pothén, “The null space problem ii. algorithms,” *SIAM Journal on Algebraic Discrete Methods*, vol. 8, no. 4, pp. 544–563, Oct. 1987.
- [6] A. Pothén and C.-J. Fan, “Computing the block triangular form of a sparse matrix,” *ACM Trans. Math. Softw.*, vol. 16, no. 4, pp. 303–324, 1990.

- [7] A. Pinar, E. Chow, and A. Pothen, “Combinatorial algorithms for computing column space bases that have sparse inverses,” *Electronic Transactions on Numerical Analysis*, vol. 22, pp. 122–145, 2006.
- [8] C. E. Bell, “Weighted matching with vertex weights: An application to scheduling training sessions in NASA space shuttle cockpit simulators,” *European Journal of Operational Research*, vol. 73, no. 3, pp. 443–449, March 1994.
- [9] G. R. Gupta, S. Sanghavi, and N. B. Shroff, “Node weighted scheduling,” in *ACM SIGMETRICS Performance Evaluation Review*, vol. 37. ACM, 2009, pp. 97–108.
- [10] C. E. Bell, “Weighted matching with vertex weights: An application to scheduling training sessions in NASA space shuttle cockpit simulators,” *European Journal of Operational Research*, vol. 73, no. 3, pp. 443–449, March 1994.
- [11] A. Pinar, E. Chow, and A. Pothen, “Combinatorial algorithms for computing column space bases that have sparse inverses,” *Electronic Transactions on Numerical Analysis*, vol. 22, pp. 122–145, 2006.
- [12] T. H. Spencer and E. W. Mayr, “Node weighted matching,” in *Proceedings of the 11th Colloquium on Automata, Languages and Programming*. London, UK: Springer-Verlag, 1984, pp. 454–464.
- [13] A. Al-Herz and A. Pothen, “A $2/3$ -approximation algorithm for vertex-weighted matching,” *Discrete Applied Mathematics*, 2019, Published online Oct. 19, 2019. DOI 10.1016/j.dam.2019.09.013.

- [14] C. Berge, “Two theorems in graph theory,” *Proceedings of the National Academy of Sciences of the United States of America*, vol. 43, no. 9, p. 842, 1957.
- [15] F. Manne and M. Halappanavar, “New effective multithreaded matching algorithms,” in *IEEE 28th International Parallel and Distributed Processing Symposium*. IEEE, 2014, pp. 519–528.
- [16] A. Pothén, S. M. Ferdous, and F. Manne, “Approximation algorithms in combinatorial scientific computing,” *Acta Numerica*, vol. 28, pp. 541–633, 2019.
- [17] H. N. Gabow and R. E. Tarjan, “Faster scaling algorithms for general graph matching problems,” *Journal of the ACM*, vol. 38, no. 4, pp. 815–853, 1991.
- [18] N. J. A. Harvey, “Algebraic algorithms for matching and matroid problems,” *SIAM Journal on Computing*, vol. 39, no. 2, pp. 679–702, 2009.
- [19] M. Mucha and P. Sankowski, “Maximum matchings via gaussian elimination,” in *45th Annual IEEE Symposium on Foundations of Computer Science*. IEEE, 2004, pp. 248–255.
- [20] P. Sankowski, “Maximum weight bipartite matching in matrix multiplication time,” *Theoretical Computer Science*, vol. 410, no. 44, pp. 4480–4488, 2009.
- [21] M. Cygan, H. N. Gabow, and P. Sankowski, “Algorithmic applications of Baur-Strassen’s theorem: Shortest cycles, diameter, and matchings,” *Journal of the ACM*, vol. 62, no. 4, p. 28, 2015.

- [22] J. E. Hopcroft and R. M. Karp, “An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs,” *SIAM Journal on Computing*, vol. 2, no. 4, pp. 225–231, 1973.
- [23] J. Edmonds, “Paths, trees, and flowers,” *Canadian Journal of Mathematics*, vol. 17, pp. 449–467, 1965.
- [24] H. N. Gabow, “An efficient implementation of Edmonds’ algorithm for maximum matching on graphs,” *Journal of the ACM*, vol. 23, no. 2, pp. 221–234, 1976.
- [25] H. N. Gabow and R. E. Tarjan, “A linear-time algorithm for a special case of disjoint set union,” *Journal of Computer and System Sciences*, vol. 30, no. 2, pp. 209–221, 1985.
- [26] S. Micali and V. V. Vazirani, “An $O(\sqrt{|V|}|E|)$ algorithm for finding maximum matching in general graphs,” in *21st Annual Symposium on Foundations of Computer Science*. IEEE, 1980, pp. 17–27.
- [27] V. V. Vazirani, “A proof of the MV matching algorithm,” *Unpublished manuscript*, 2014.
- [28] A. Schrijver, *Combinatorial Optimization: Polyhedra and Efficiency*. Springer, 2003, vol. 24.
- [29] H. N. Gabow, “Implementation of algorithms for maximum matching on non-bipartite graphs.” Ph.D. dissertation, Stanford University, Stanford, CA, USA, 1974.

- [30] M. O. Ball and U. Derigs, “An analysis of alternative strategies for implementing matching algorithms,” *Networks*, vol. 13, no. 4, pp. 517–549, 1983.
- [31] Z. Galil, S. Micali, and H. Gabow, “An $O(EV \log V)$ algorithm for finding a maximal weighted matching in general graphs,” *SIAM Journal on Computing*, vol. 15, no. 1, pp. 120–130, 1986.
- [32] M. L. Fredman and R. E. Tarjan, “Fibonacci heaps and their uses in improved network optimization algorithms,” *Journal of the ACM*, vol. 34, no. 3, pp. 596–615, 1987.
- [33] R. Duan, S. Pettie, and H.-H. Su, “Scaling algorithms for weighted matching in general graphs,” *ACM Transactions on Algorithms*, vol. 14, no. 1, p. 8, 2018.
- [34] J. Edmonds, “Maximum matching and a polyhedron with 0, 1-vertices,” *Journal of Research of the National Bureau of Standards B*, vol. 69, no. 125-130, pp. 55–56, 1965.
- [35] V. Tabatabaee, L. Georgiadis, and L. Tassiulas, “QoS provisioning and tracking fluid policies in input queueing switches,” *IEEE/ACM Transactions on Networking*, vol. 9, no. 5, pp. 605–617, 2001.
- [36] F. Dobrian, M. Halappanavar, A. Pothén, and A. Al-Herz, “A $2/3$ -approximation algorithm for vertex weighted matching in bipartite graphs,” *SIAM Journal on Scientific Computing*, vol. 41, no. 1, pp. A566–A591, 2019.
- [37] M. Halappanavar, “Algorithms for vertex-weighted matching in graphs,” Ph.D. dissertation, Old Dominion University, 2009.

- [38] A. L. Dulmage and N. S. Mendelsohn, “Coverings of bipartite graphs,” *Canadian Journal of Mathematics*, vol. 10, no. 4, pp. 516–534, 1958.
- [39] D. P. Williamson and D. B. Shmoys, *The Design of Approximation Algorithms*. Cambridge University Press, 2011.
- [40] D.-Z. Du, K.-I. Ko, and X. Hu, *Design and Analysis of Approximation Algorithms*. Springer, 2011.
- [41] V. V. Vazirani, *Approximation Algorithms*. Springer, 2013.
- [42] D. S. Hochbaum, *Approximation Algorithms for NP-Hard Problems*. PWS Publishing Co., 1996.
- [43] S. Hougardy, “Linear time approximation algorithms for degree constrained subgraph problems,” in *Research Trends in Combinatorial Optimization*. Springer, 2009, pp. 185–200.
- [44] D. Avis, “A survey of heuristics for the weighted matching problem,” *Networks*, vol. 13, no. 4, pp. 475–493, 1983.
- [45] R. Preis, “Linear time $1/2$ -approximation algorithm for maximum weighted matching in general graphs,” in *STACS 99*. Springer, 1999, pp. 259–269.
- [46] J.-H. Hoepman, “Simple distributed weighted matchings,” 2004.
- [47] F. Manne and R. H. Bisseling, “A parallel approximation algorithm for the weighted maximum matching problem,” in *Parallel Processing and Applied Math-*

- ematics*, R. Wyrzykowski, J. Dongarra, K. Karczewski, and J. Wasniewski, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 708–717.
- [48] D. E. Drake and S. Hougardy, “A simple approximation algorithm for the weighted matching problem,” *Information Processing Letters*, vol. 85, no. 4, pp. 211–213, 2003.
 - [49] J. Maue and P. Sanders, “Engineering algorithms for approximate weighted matching,” in *International Workshop on Experimental and Efficient Algorithms*. Springer Verlag, 2007, pp. 242–255, lecture Notes in Computer Science, Vol. 4525.
 - [50] D. Gale and L. S. Shapley, “College admissions and the stability of marriage,” *The American Mathematical Monthly*, vol. 69, no. 1, pp. 9–15, 1962.
 - [51] S. Pettie and P. Sanders, “A simpler linear time $2/3 - \epsilon$ approximation for maximum weight matching,” *Information Processing Letters*, vol. 91, no. 6, pp. 271–276, 2004.
 - [52] N. H. Hovland, “Maximum weighted matching on a gpu,” Master’s thesis, The University of Bergen, 2020.
 - [53] R. Duan and S. Pettie, “Approximating maximum weight matching in near-linear time,” in *2010 IEEE 51st Annual Symposium on Foundations of Computer Science*. IEEE, 2010, pp. 673–682.
 - [54] S. Hanke and S. Hougardy, “New approximation algorithms for the weighted matching problem.” Research Institute for Discrete Mathematics, University of Bonn, Research Report 101010, 2010, 2010.

- [55] R. Duan and S. Pettie, “Linear time approximation for maximum weight matching,” *J. ACM*, vol. 61, no. 1, 2014, article 1.

VITAE

- Name: Saliyah Shuraym Alotaibi
- Nationality: Saudi
- Date of Birth: 14 Aug 1989
- Email: *alotaibi.saliyah@gmail.com*
- Permanent Address: P.O. Box 13712, Building 8535, Asma Bint Abi Bakr, Ad Diriyah, Saudi Arabia
- Academic Background:
 - 2020-2024: MASTER DEGREE IN COMPUTER SCIENCE, King Fahd University of Petroleum and Minerals.
 - 2018-2019: BUILDING LEADERSHIP, University of Delaware.
 - 2008-2011: BACHELOR OF COMPUTER SCIENCE, Shaqra University.
- Published Research:
 1. "A Distributed Memory $1/2$ Approximation Algorithm for Vertex-Weighted Matching," presented at the 3rd IRC-ISS Student Research Symposium, KFUPM, May 2024.

2. "CSS: An Android Application for Solving CPU Process Scheduling Problems," published and presented at IEEE CSDE , December 2023.
3. "Instructional Coaching," published in the 100 Research of Saudi Teachers to Achieve Saudi Vision in Education encyclopedia, October 2023.
4. "The Leader Teacher: Empowering Educators for Transformational Leadership," training program developed and implemented for the National Center for Professional Development, 2019.

- Conference and Workshop Presentations:

As a recognized expert in artificial intelligence, data science, and emerging technologies, I have been invited to speak at various conferences, workshops, and forums. Below is a list of my speaking engagements:

1. Workshop on Applications of Generative Artificial Intelligence in the Future of Electricity – Saudi Electricity Company, December 2024.
2. Workshop on Analyzing the Performance of Large Language Models using Evaluation Metrics and Analytical Tools – Allam Challenge (SDAIA), November 2024.
3. Seminar on The Role of Artificial Intelligence in Enhancing User Experience on the Internet – Saudi Internet Society, November 2024.
4. Monitor for Generative AI using Llama - Metaverse Academy – July 2024.
5. Webinar on Applications of Generative Artificial Intelligence: Insights and Challenges – Tuwaiq Academy, October 2024.
6. Talk on The Role of Technology Amid the Rapid Wave of Artificial Intelli-

gence – Tuwaiq Talk, Tuwaiq Academy, May 2024.

7. Trainer for Data Science and Artificial Intelligence (T5 Bootcamp) – Tuwaiq Academy, 2023-2024.

8. Research Presentation for Android Application for Solving CPU Process Scheduling Problems (CSS) – Women in Data Science Conference, KFUPM, April 2021.

9. Workshop on Cybersecurity – Women Techmakers, March 2019.