Parallel Approximation Algorithms for Rainbow Matching in Bipartite Graphs with Applications to Machine Learning Systems

Saeed Bakhshan and Maedeh Yahaghi

Department of Computer Science, Wayne State University,
5057 Woodward Ave., Detroit, 48202, Michigan, USA

Abstract. A rainbow matching in an edge-colored graph is defined as a matching in which all edges have distinct colors. The maximum cardinality rainbow matching problem seeks to determine the largest rainbow matching in a graph. It is a fundamental problem in graph theory with numerous applications. In this paper, we present the first parallel approximation algorithm for rainbow matching in edge-colored bipartite graphs. This algorithm achieves a 1/3-approximation ratio, matching the classical guarantee for greedy maximal rainbow matchings (and, more generally, for 3-dimensional matching). Specifically, for any symmetric edge-colored bipartite graph with n vertices, m edges, and q colors, the proposed algorithm computes a maximal rainbow matching of size at least one third of the optimal matching in $O(n)$ time on a PRAM with n^2 processors. Additionally, we present a deterministic sequential version of the algorithm, which computes a 1/3-approximation in $O(m + n \log n)$ time, mirroring the approximation ratio of the parallel algorithm. We provide a practical OpenMP implementation of the proposed parallel algorithm on a multi-core system and perform an extensive performance evaluation. The experimental results show that for large graphs with hundreds of millions of edges, the parallel algorithm achieves a considerable speedup relative to its sequential counterpart of up to 5.908 when using 32 cores. Rainbow matchings naturally arise in data-intensive and machine learning workflows where we must select large sets of pairwise non-conflicting interactions (e.g., user–item, data–worker, or job–machine assignments) under diversity or fairness constraints represented as edge colors. We outline how the proposed sequential and parallel algorithms can serve as scalable combinatorial primitives for such tasks, while leaving a full empirical evaluation of downstream accuracy and fairness benefits to future work.

Keywords: Rainbow matching , approximation algorithms , parallel algorithms , machine learning systems , graph-based learning

1 Introduction

Finding a maximum cardinality matching in a graph is a fundamental problem in graph theory with numerous applications. In a graph, a *matching* is a set of edges such that no two of them share a common vertex. For a bipartite graph with n vertices and m edges, the maximum cardinality matching can be computed in $O(m\sqrt{n})$ time using the Hopcroft-Karp algorithm [1]. In edge-colored graphs, this problem generalizes to the *maximum cardinality rainbow matching* problem. A rainbow matching in an edge-colored graph is defined as a matching in which all edges have distinct colors [2]. This problem, also referred to as *multiple choice matching* [3], asks whether a given edge-colored graph G contains a rainbow matching of size at least k. The optimization variant of this problem seeks to determine the largest rainbow matching in the graph. The additional constraint of requiring distinct edge colors increases the complexity of the problem, making it NP-complete [3]. Additionally, it has been established that the rainbow matching problem remains NP-complete even when restricted to edge-colored bipartite graphs [4].

Formally, the *Maximum Cardinality Rainbow Matching* problem in bipartite graphs is defined as follows:

David C. Wyld et al. (Eds): CSML, AISCA, DNLP, SOEA, NET, BDHI, SIPO – 2026
pp. 01-19, 2026.CS & IT - CSCP 2026 DOI: 10.5121/csit.2026.160201

Given a bipartite graph $G = (U \cup W, E)$, a coloring function $\chi : E \to Q$, where $|U| = |W| = n$, $|E| = m$, and $Q = \{1, \ldots, q\}$ is the set of q colors, find a matching of maximum cardinality such that all the edges in the matching have distinct colors.

The hardness of the problem highlights the need for efficient algorithms capable of finding large, preferably maximal, rainbow matchings in polynomial time, where a *maximal matching* is a matching that is not properly contained in any other matching. *To the best of our knowledge, no parallel algorithm for solving the rainbow matching problem has been proposed to date.*

Rainbow matching is essentially equivalent to the classical *3-dimensional matching* (3DM) problem when we interpret U, W, and the color set Q as the three ground sets, and each edge (u, w) of color c as a hyperedge (u, w, c). Thus, the NP-completeness and approximation hardness results for 3DM (see, e.g., Garey and Johnson [3]) carry over to rainbow matching in this more general setting. At the same time, rainbow matching is naturally defined and studied on general edge-colored graphs, without any bipartiteness restriction; for example, Bennett et al. analyze greedy rainbow matching algorithms in general edge-colored graphs [5]. From this perspective, the bipartite setting we focus on in this paper is not a redundant reformulation of 3DM, but a structured special case of the general rainbow matching problem. Results for bipartite graphs are therefore complementary to those for general graphs: they isolate the algorithmic effect of bipartiteness and enable specialized parallel and approximation techniques that are not directly available in the fully general case.

Rainbow matching has applications in packing problems like Bin Packing, Multiple Knapsack, and Bin Covering. In these problems, items or constraints are represented as colors to ensure distinct groupings that meet capacity or coverage requirements [6]. Another application is in kernelization for graph packing problems [7], where rainbow matching techniques play a critical role in designing polynomial kernels. Beyond these combinatorial applications, rainbow matchings also arise naturally in machine learning and data-intensive systems. In a bipartite user–item or data–worker graph, a rainbow matching corresponds to a set of pairwise non-conflicting assignments in which each edge color models a type, group, or constraint (e.g., item category, data source, or protected attribute). Enforcing that all edges in the matching have distinct colors yields a diverse or fair subset of interactions, which is useful for tasks such as constructing diverse mini-batches or coresets that cover many labels or domains without redundancy, generating fair and diverse recommendation lists that limit over-exposure of any single provider or content group, and allocating training jobs to heterogeneous resources in distributed learning systems while spreading them across resource types or failure domains. In such settings, large rainbow matchings provide a principled way to select a large, diverse, and non-conflicting subset of interactions, motivating the need for scalable approximation algorithms on massive edge-colored bipartite graphs.

In this paper, we *design both sequential and parallel approximation algorithms for solving the rainbow matching problem* in edge-colored bipartite graphs. Our algorithms compute a maximal rainbow matching and thus guarantee a 1/3-approximation: the size of the matching they return is at least one third of the optimal rainbow matching. The sequential algorithm runs in $O(m+n \log n)$ time, while the parallel algorithm runs in $O(n)$ on a CRCW PRAM with n^2 processors for any edge-colored bipartite graph with n vertices, m edges, and q colors.

The remainder of this paper is organized as follows. Section 2 reviews the related works on the rainbow matching problem. Section 3 summarizes our contributions. Section 4 presents the sequential greedy rainbow matching algorithm (S-GRM) and provides an

analysis of its correctness and time complexity. Section 5 introduces the parallel greedy rainbow matching algorithm (P-GRM), discussing its time complexity, and work analysis. Section 6 analyzes the approximation guarantees of both S-GRM and P-GRM, establishing their 1/3-approximation ratio. Section 7 discusses applications of rainbow matching and of our algorithms to machine learning and data-intensive systems. Section 8 discusses experimental results. Finally, Section 9 concludes the paper.

2 Related Works

A motivation for studying rainbow matchings is Ryser's conjecture [8], which states that any Latin square of odd order has at least one Latin transversal. A Latin square is an $n \times n$ grid filled with n distinct symbols, each appearing exactly once in every row and column. A Latin transversal involves selecting n cells—one from each row and column—such that all n distinct symbols are included. Ryser's conjecture can be translated into the setting of graphs where any proper edge coloring of the complete bipartite graph $K_{2n+1,2n+1}$ using $2n+1$ colors guarantees a rainbow matching of size $2n+1$. In this context, the symbols of a Latin square correspond to edge colors, and a Latin transversal relates to selecting edges with distinct colors, where no two selected edges share a vertex. Identifying a Latin transversal corresponds to finding a rainbow matching in a properly edge-colored bipartite graph, where proper coloring ensures that no two edges incident to the same vertex share the same color.

Le and Pfender [2] investigated the hardness and approximation guarantees for the rainbow matching problem. They demonstrated that the maximum rainbow matching can be approximated in polynomial time with an approximation factor of $2/3 - \epsilon$ for any $\epsilon > 0$. This result was obtained by reducing the rainbow matching problem to the *Maximum Independent Set* (MIS) problem on $K_{1,4}$-free graphs.

Gupta et al. [9] presented parametrized algorithms for solving the rainbow matching problem. For paths, they developed a deterministic algorithm with a running time of $O^*\left(\left(\frac{1+\sqrt{5}}{2}\right)^k\right)$, where k is a positive integer representing a matching of size at least k. The algorithm uses the method of bounded search trees combined with a divide-and-conquer strategy. Additionally, they introduced a randomized fixed-parameter tractable (FPT) algorithm for solving the problem in general graphs, which requires $O^*\left(2^k\right)$ time and polynomial space. They also proposed a quadratic kernel for the rainbow matching problem in general graphs [10], which provides additional insights into its parameterized complexity. Their kernelization approach relies on decomposing the graph into smaller components and leveraging an expansion lemma to refine the problem size.

Kelk and Stamoulis [11] investigated the integrality gap of the standard linear programming relaxation of the Bounded Color Matching (BCM) problem. The BCM problem, in a weighted edge-colored graph, concerns finding a maximum weighted matching while ensuring that the number of edges of each color does not exceed the specified limit for that color. They constructed various instance families and derived lower bounds on their integrality gaps. Furthermore, they analyzed the behavior of these instances under the Sherali–Adams "lift-and-project" technique. They also demonstrated that the integrality gap of the natural linear programming formulation improves when specific simple sub-structures are excluded from the input graphs.

Dyer et al. [12] analyzed two greedy algorithms for finding large matchings in *color-free graphs*, demonstrating that these approaches achieve asymptotic matching sizes proportional to the graph size. Their work provided foundational insights into randomized strategies for efficient matching in graphs. Karp and Sipser [13] introduced the KSGreedy algo-

rithm, which was designed for random graphs $G_{n,p}$ with $p = \frac{c}{n}$ where c is a constant. Their algorithm achieves an asymptotically maximal matching size and has become a benchmark for randomized matching algorithms. Later, Aronson et al. [14] refined KSGreedy, obtaining a matching size within $O(n^{1/5} \log^{O(1)} n)$ of the maximum. Bennett et al. [5] extended these greedy strategies to the rainbow matching problem, where the goal is to construct a matching of edges with distinct colors. By adapting the greedy and modified greedy frameworks, their algorithms addressed challenges in randomly colored edge graphs and showed the rainbow matching size obtained by greedy approaches.

None of the previous works provided parallel approximation algorithms for the rainbow matching problem. Instead, they primarily focused on parameterized and randomized greedy sequential algorithms.

3 Our Contribution

In this paper, we propose both sequential and parallel algorithms for the cardinality rainbow matching problem in edge-colored bipartite graphs. Our main contributions are summarized as follows:

- We design P-GRM, a parallel greedy algorithm that achieves a 1/3-approximation on the CRCW PRAM model. For a graph with n vertices, m edges, and q colors, the algorithm runs in $O(n)$ time using n^2 processors. *To the best of our knowledge, this is the first parallel algorithm that provides a constant-factor approximation guarantee for the rainbow matching problem.*
- We introduce S-GRM, a deterministic sequential counterpart that preserves the trivial 1/3-approximation guarantee for maximal rainbow matchings while running in $O(m + n \log n)$ time. Both algorithms share a common greedy core, making their analysis and implementation directly comparable.
- We provide an extensive experimental evaluation (Section 8) that validates the performance of P-GRM in practice. The results show that for large graphs with hundreds of millions of edges, the parallel algorithm achieves a considerable speedup relative to its sequential counterpart of up to 5.908, when using 32 cores.
- We discuss how maximal rainbow matchings and our algorithms can serve as scalable building blocks in machine learning and data-intensive applications, including diverse mini-batch and coreset construction, fair and diverse recommendation, graph-based learning on heterogeneous networks, and resource allocation in distributed training systems (Section 7). These are proposed use cases that illustrate how the combinatorial primitive can be instantiated; we do not empirically evaluate these downstream applications in this work.

4 Sequential Greedy Rainbow Matching Algorithm (S-GRM)

S-GRM obtains a maximal rainbow matching in an edge-colored bipartite graph G, where a *maximal matching* is a matching that is not properly contained in any other matching. The main idea of our design is to maximize the chances of forming a large rainbow matching by prioritizing which vertices and edges to process. Our algorithm processes vertices in non-decreasing order of their degrees, starting with the vertex with the minimum degree. This ensures that vertices with lower degrees are matched first, reducing the chance of missing them later. Additionally, among the vertices of equal degrees, S-GRM prioritizes

Algorithm 1 S-GRM: Sequential Greedy Rainbow Matching

Input: An edge-colored bipartite graph $G = (U \cup W, E)$ with color function $\chi : E \to Q$
Output: Maximal rainbow matching $\mathcal{M}$

Phase 1: Pre-processing and Sorting
1: $\mathcal{M} \leftarrow \emptyset$
2: $\mathcal{C} \leftarrow \emptyset$
3: $\mathcal{F} \leftarrow U$
4: Compute $\deg(w_i), \forall w_i \in W$
5: Compute $\omega(c), \forall c \in Q$ and $\Upsilon(w_i), \forall w_i \in W$
6: $[w_{\pi(1)}, w_{\pi(2)}, \ldots, w_{\pi(n)}] = \mathsf{SORT}(\ [\{w_1, w_2, \ldots, w_n]\ )$
{Sort all $w_i \in W$ in non-decreasing order of $\deg(w_i)$
and if ties, sort w_i in non-decreasing order of $\Upsilon(w_i)$.}
Phase 2: Matching
7: **for** $i = 1 \ldots, n$ **do**
8: $\mathcal{K} \leftarrow \emptyset$
9: **for** each edge $e = (u, w_{\pi(i)})$ **do**
10: **if** $u \in \mathcal{F}$ and $\chi(e) \notin \mathcal{C}$ **then**
11: $\mathcal{K} \leftarrow \mathcal{K} \cup \{e\}$
12: **if** $\mathcal{K} \neq \emptyset$ **then**
13: $e^* \leftarrow \arg\min_{e \in \mathcal{K}} \{\omega(\chi(e))\}$
14: $\mathcal{M} \leftarrow \mathcal{M} \cup \{e^*\}$
15: $\mathcal{C} \leftarrow \mathcal{C} \cup \{\chi(e^*)\}$
16: $\mathcal{F} \leftarrow \mathcal{F} \setminus \{u\}$
17: **return** $\mathcal{M}$

those with a smaller sum of color utilization over all its incident edges. Here, the *color utilization* of color c, denoted by $\omega(c)$, is the number of edges of color c in G, i.e.,

$$\omega(c) = |\{e \,|\, e \in E \wedge \chi(e) = c\}|. \tag{1}$$

This strategy avoids using edges colored with a color having small color utilization in the next iterations of the algorithm, thus increasing the chances of forming a large rainbow matching. When selecting an edge incident to the current vertex, the algorithm chooses the one with the least utilized color. Intuitively, this "degree-and-color-aware" ordering gives priority to vertices that are harder to satisfy (low degree or incident to rare colors), which is in line with classical heuristics for matchings and independent sets. While our approximation analysis in Section 6 only relies on maximality and does not require this specific ordering, our experimental results indicate that it yields large rainbow matchings in practice, and the ordering is natural for the data-intensive scenarios we target.

S-GRM, given in Algorithm 1, consists of two phases: (1) pre-processing and sorting; and (2) matching. In *Phase 1*, S-GRM initializes the matching, $\mathcal{M}$, and the set $\mathcal{C}$ of colors utilized in the matching with the empty set. It also initializes the set $\mathcal{F}$ of vertices in U that are free (not matched) with all the vertices in U (Lines 1-3). It computes $\omega(c)$, the color utilization for each color $c \in Q$, and for each $w_i \in W$, $\Upsilon(w_i)$, the sum of the utilization of the colors of edges incident to a vertex $w_i \in W$ for all such vertices, where $\Upsilon(w_i)$ is given by

$$\Upsilon(w_i) = \sum_{\forall u:(u,w_i) \in E \wedge \chi(u,w_i) = c} \omega(c). \tag{2}$$

This ordering of vertices in W encourages the algorithm to first satisfy vertices whose incident edges either have low degree or rely on rare colors, thereby preserving highly utilized colors and high-degree vertices for later iterations when fewer options remain.

It then calls the SORT procedure which sorts the vertices in the right partition, $w_i \in W$, in non-decreasing order of their degrees. If there are vertices having the same degree, it

sorts them in non-decreasing order of $\Upsilon(w_i)$, the sum of the utilization of the colors of their incident edges (Line 6).

For example, let us assume that graph G has edges colored using three colors c_1, c_2, and c_3, and that the color utilizations for these three colors are $\omega(c_1) = 1$, $\omega(c_2) = 2$, and $\omega(c_3) = 3$. Furthermore, assume that two vertices w_i and w_j have degree 2. The edges incident to w_i are of colors c_1 and c_2, thus $\Upsilon(w_i) = \omega(c_1) + \omega(c_2) = 3$. The edges incident to w_j are of colors c_2 and c_3, thus $\Upsilon(w_j) = \omega(c_2) + \omega(c_3) = 5$. Since $\Upsilon(w_i) < \Upsilon(w_j)$, S-GRM places w_i before w_j in the order obtained in Line 6 of S-GRM.

In *Phase 2*, S-GRM traverses the sorted list of vertices and for every vertex $w_i \in W$, finds the candidate edges that can potentially be matching edges and adds them to the potential matching set, denoted by $\mathcal{K}$ (Lines 8–11). Then, it checks if for a vertex w_i, there are candidate edge(s) (Lines 12–16). If so, it picks the edge (u, w_i) colored with the color having the least utilization in G and adds it to the matching $\mathcal{M}$. It also adds the color of the picked edge to $\mathcal{C}$, and removes vertex u from the set $\mathcal{F}$ of free vertices. If there are no more vertices in the sorted sequence that need to be processed, it returns the matching $\mathcal{M}$ (Line 17).

Theorem 1. *S-GRM obtains a maximal rainbow matching in the edge-colored bipartite graph G, that is, a rainbow matching that is not properly contained in any other rainbow matching.*

Proof. Since S-GRM involves a sequence of iterations, the loop invariant for this algorithm can be stated as follows: at the start of each iteration i, $i = 1, \ldots, n$ of the outer loop (Line 7), matching $\mathcal{M}$ satisfies two properties: (i) it is a rainbow matching (i.e., no two edges in $\mathcal{M}$ share the same vertex and the same color); (ii) it is maximal for the vertices already processed $(w_{\pi(1)}, w_{\pi(2)}, \ldots, w_{\pi(i-1)})$. The algorithm chooses the edges in a greedy manner by selecting the edge with the smallest color utilization $\omega(\chi(e))$ at each step.

Before the first iteration ($i = 1$), matching $\mathcal{M}$ and the set $\mathcal{C}$ of used colors are empty, and the set $\mathcal{F}$ contains vertices in U that are not matched. $\mathcal{M}$ is trivially a maximal rainbow matching. This establishes that the invariant holds initially. As for iteration i, S-GRM processes the vertex $w_{\pi(i)}$ by considering edges incident to it. It filters edges to ensure that their other endpoints, $u \in U$, are in $\mathcal{F}$ and their colors $\chi(e)$ are not in $\mathcal{C}$. Then, it selects an edge e^* to include in the matching whose color has the minimum utilization $\omega(c)$ among the filtered edges, therefore satisfying the greedy property. Once the edge e^* is added to $\mathcal{M}$, both $\mathcal{C}$ and $\mathcal{F}$ are updated, and the matching $\mathcal{M}$ remains maximal since, in every step and for each vertex $w_{\pi(i)}$, it selects an edge which does not share a vertex and a color with previously matched edges. Thus, the invariant is maintained. After the last iteration ($i = n$), all vertices in W have been processed and the matching $\mathcal{M}$ is a maximal rainbow matching for the entire graph. By construction, no additional edges can be added to $\mathcal{M}$ without violating the rainbow matching property. This shows that after every iteration, the matching $\mathcal{M}$ satisfies the desired properties, and when the algorithm terminates, it produces a maximal rainbow matching in the edge-colored bipartite graph. □

Theorem 2. *The running time of S-GRM is $O(m + n \log n)$.*

Proof. To analyze the running time of S-GRM we determine the running time of each of the two phases of the algorithm as follows.

Phase 1: S-GRM initializes three sets, $\mathcal{M}$, $\mathcal{C}$, and $\mathcal{F}$ which takes $O(n)$ due to copying n vertices into $\mathcal{F}$. Computing the degree $\deg(w_i)$ of all vertices $w_i \in W$, the color utilization, and $\Upsilon(w_i)$ involves scanning the set of edges only once, which requires $O(m)$ time (Lines

1–5). Then, S-GRM sorts the n vertices of W in non-decreasing order of their degrees (including tie-breaking based on $\Upsilon(w_i)$) in $O(n \log n)$ time (Line 6). Thus, the running time for Phase 1 is $O(m + n \log n)$.

Phase 2: For each vertex $w_{\pi(i)}$, S-GRM scans all its incident edges. Thus, the total number of edges scanned over all vertices $w_{\pi(i)}$ is m, and therefore this phase requires $O(m)$ time (Lines 8–11). S-GRM selects the edge with the minimum color utilization from $\mathcal{K}$. The total number of edges that are part of $\mathcal{K}$ over all vertices is at most m, and thus, this requires $O(m)$ time (Lines 12–13). Each vertex update requires constant time. Since there are n vertices, this step requires $O(n)$ time (Lines 14–16). Thus, the running time for Phase 2 is $O(m + n)$.

Summing the running times of the two phases leads to a total running time of $O(m + n \log n)$. □

5 Parallel Greedy Rainbow Matching Algorithm (P-GRM)

P-GRM is designed to compute a maximal rainbow matching in an edge-colored bipartite graph G and provide the same approximation guarantees (i.e., a 1/3-approximation ratio) as the sequential algorithm S-GRM. The design of the algorithm assumes the Combining CRCW PRAM model with Minimum as the combining operator and n^2 processors. In the Combining PRAM with Minimum as the operator, if more than one processor write into the same memory cell the value written in the cell is the minimum among the values written by the processors. P-GRM, presented in Algorithm 2, consists of two phases: (1) preprocessing and sorting; and (2) matching.

In *Phase 1*, it computes in parallel the degree of every vertex in the right bipartition, i.e. $\forall w_i \in W$ (Lines 4-5). This involves n processors, each processor i adding the entries of the column corresponding to vertex w_i in the adjacency matrix, for $i = 1, \ldots, n$.

Next, it computes in parallel the color utilization $\omega(c)$, for $c = 1, \ldots, q$, where $q \leq m \leq n^2$ (Lines 6-7). This is done by n^2 processors in parallel and consists of two steps. In the first step, n subsets of n processors determine the color utilization for each of the edges associated with vertex w_i, $i = 1, \ldots, n$. Concretely, we allocate for each vertex w_i an array $\omega_i[1 \ldots q]$ in shared memory, initialized to zero. Processor (i, j), responsible for the j-th potential neighbor of w_i, increments $\omega_i[\chi(e)]$ for each existing edge $e = (u, w_i)$ of color $\chi(e)$. The color utilization for the edges associated with vertex w_i is thus stored in an array of size q; since $q \leq m \leq n^2$, the maximum size of each array is n^2. The second step consists of an element-wise reduction over these arrays to obtain the global color utilization array $\omega[1 \ldots q]$, where

$$\omega[c] \;=\; \sum_{i=1}^{n} \omega_i[c].$$

This reduction can be implemented as a tree of pairwise additions over the n arrays, using n^2 processors and $O(n)$ parallel time. We emphasize that throughout our analysis we assume $q \leq n^2$, which is consistent with our bipartite setting where $m \leq n^2$ and each edge carries exactly one color.

It also computes in parallel the sum of the utilization of the colors of edges incident to a vertex $w \in W$, $\Upsilon(w)$ (Lines 8-9). This is achieved by employing n processors, one per each vertex w_i, which scans the edges adjacent to w_i and sums the color utilizations corresponding to each of these edges. This is where the concurrent read (CR) is needed, since several processors may access the entry corresponding to the same color in the array containing the color utilization.

Algorithm 2 P-GRM: Parallel Greedy Rainbow Matching

Input: Edge-colored bipartite graph $G = (U \cup W, E)$ with color function $\chi : E \to Q$
Output: A maximal rainbow matching $\mathcal{M}$

Phase 1: Pre-processing and Sorting
1: $\mathcal{M} \leftarrow \emptyset$
2: $\mathcal{C} \leftarrow \emptyset$
3: $\mathcal{F} \leftarrow U$
4: **for all** $w_i \in W$ **pardo**
5: compute $\deg(w_i)$
6: **for all** $c \in Q$ **pardo**
7: compute $\omega(c)$
8: **for all** $w_i \in W$ **pardo**
9: compute $\Upsilon(w_i)$
10: $[w_{\pi(1)}, w_{\pi(2)}, \ldots, w_{\pi(n)}]$ = PAR-SORT($[\{w_1, w_2, \ldots, w_n]$)
{Sort all $w_i \in W$ in non-decreasing order of $\deg(w_i)$
and if ties, sort w_i in non-decreasing order of $\Upsilon(w_i)$.}
Phase 2: Matching
11: **for** $i = 1$ **to** n **do**
12: **for all** $u \in \mathcal{F}$ **pardo**
13: **if** $(u, w_{\pi(i)}) \in E$ **and** $\chi(u, w_{\pi(i)}) \notin \mathcal{C}$ **then**
14: $\mathcal{T}_u \leftarrow (u, w_{\pi(i)})$
15: **else**
16: $\mathcal{T}_u \leftarrow \perp$
17: $(u^*, w_{\pi(i)})$ = PAR-MIN($\{\omega(\chi(u, w_{\pi(i)}) \text{ — } \forall \mathcal{T}_u \neq \perp\}$)
18: **if** $(u^*, w_{\pi(i)}) \neq \perp$ **then**
19: $\mathcal{M} \leftarrow \mathcal{M} \cup \{(u^*, w_{\pi(i)})\}$
20: $\mathcal{C} \leftarrow \mathcal{C} \cup \{\chi(u^*, w_{\pi(i)})\}$
21: $\mathcal{F} \leftarrow \mathcal{F} \setminus \{u^*\}$
22: **return** $\mathcal{M}$

Finally, it sorts in parallel all $w_i \in W$ in non-decreasing order of $\deg(w_i)$, and if ties, it sorts the tied w_i's in non-decreasing order of $\Upsilon(w_i)$ (Line 10). This is achieved using the procedure PAR-SORT which implements a CRCW PRAM parallel integer sorting algorithm such as parallel merge sort [15]. This ordering ensures that vertices whose incident edges have scarce colors are processed earlier in the process, thereby reserving frequent colors for potential later matches.

In *Phase 2*, P-GRM considers each vertex in the ordered list $w_{\pi(1)}, \ldots, w_{\pi(n)}$ and tries to find a matching edge to a vertex $u \in U$. When vertex $w_{\pi(i)}$ is processed, all free vertices in the left partition $u \in U$ are checked in parallel by n processors for the existence of edge $(u, w_{\pi(i)})$. If edge $(u, w_{\pi(i)})$ exists and its color is not utilized, then it is considered a candidate edge for matching and is assigned to $\mathcal{T}_u$, the candidate edge variable, otherwise $\mathcal{T}_u$ is set to $\perp$ (i.e., null) (Lines 12–16).

All proposals generated in the same iteration are subjected to a deterministic conflict resolution rule that keeps precisely one edge whenever multiple proposals share a color or a vertex in the left bipartition. The edge selected for the matching is the one with the minimum color utilization $\omega(\chi)$, and any tie is broken by choosing the vertex u that has the smaller index. P-GRM employs the PAR-MIN procedure to determine in parallel on n processors the edge $(u^*, w_{\pi(i)})$ whose color has the minimum color utilization among the candidate edges in $\mathcal{T}_u$, where $u \in \mathcal{F}$ and $\mathcal{T}_u \neq \perp$ (Line 17). PAR-MIN is a procedure for computing in parallel the minimum among a set of at most n integers that returns the edge colored with a color having minimum color utilization. In more detail, each processor responsible for a candidate edge $(u, w_{\pi(i)})$ writes the pair $(\omega(\chi(u, w_{\pi(i)})), u)$ into a shared memory cell using the combining CRCW semantics with the *Minimum* operator

on the first component and vertex index as a tie-breaker. The PRAM retains the pair with smallest first component, and ties are broken by the smallest u. The selected edge for the matching can then be read from this cell in $O(1)$ time. That is, PAR-MIN returns the selected edge for the matching if such an edge is identified, and null ($\perp$) if such an edge does not exist (Line 17). This can be done in constant time on a Combining CRCW PRAM with n processors and Minimum as the combining associative operator. The selected edge is then added to the matching $\mathcal{M}$, its color $\chi(u^*, w_{\pi(i)})$ is added to the set $\mathcal{C}$ of utilized colors, and its left endpoint vertex $u \in U$ is removed from the set of free vertices (Lines 19-21).

Theorem 3. *The running time of P-GRM on a Combining CRCW PRAM with n^2 processors is $O(n)$, and the work is $O(n^3)$.*

Proof. To analyze the running time of P-GRM on the Combining CRCW PRAM model with n^2 processors, we determine the running time of each of the phases of the algorithm.

Phase 1: Pre-processing and Sorting. P-GRM initializes the sets $\mathcal{M}$, $\mathcal{C}$. It also copies all n vertices from U into $\mathcal{F}$, which takes $O(1)$ time in parallel. The degrees $\deg(w_i)$ for all vertices $w_i \in W$ are computed in $O(\log n)$ with n^2 processors.

Computing the color utilization $\omega(c)$, for $c = 1, \ldots, q$, where $q \leq m$, requires $O(n)$ time on n^2 processors and consists of two steps. In the first step, n subsets of n processors determine the color utilization for each of the edges associated with vertex w_i, $i = 1, \ldots, n$. The color utilization for the edges associated with vertex w_i is stored in an array of size q, $q \leq n^2$. The second step consists of element-wise addition of the n arrays of size n^2, which takes $O(n)$ using n^2 processors. Thus, the total running time to compute the color utilization $\omega(c)$ is $O(n)$.

The sum of the utilization of colors $\Upsilon(w_i)$ at each vertex w_i can be computed using n processors, one per vertex w_i, by scanning the edges adjacent to w_i and summing the color utilizations corresponding to each of these edges. This will also require $O(n)$ time.

Sorting the n vertices of W based on their degrees and, in case of ties, based on $\Upsilon(w_i)$, using the PAR-SORT procedure which implements a CRCW PRAM parallel integer sorting algorithm such as parallel merge sort [15] requires $O(\log n)$ parallel running time on n processors. The parallel running time for *Phase 1* is dominated by the procedure which computes the color utilization, and therefore, it is $O(n)$.

Phase 2: Matching. In each iteration i of the loop in Lines 11-21, P-GRM checks for every free vertex $u \in \mathcal{F}$ if the edge $(u, w_{\pi(i)})$ exists and whether its color is not utilized. Since there are potential n edges from vertices $u \in U$ to a specific vertex $w_{\pi(i)}$, each of the n processors checks only one edge in parallel, which takes $O(1)$ time. Next, P-GRM employs the PAR-MIN procedure which computes in parallel the minimum among a set of at most n integers that returns the edge colored with a color having minimum color utilization. This can be done in $O(1)$ time on a Combining CRCW PRAM with n processors and Minimum as the combining associative operator. The edge selected for matching is added to the matching $\mathcal{M}$, the set of utilized colors $\mathcal{C}$ is updated, and the left end vertex u^* of the selected edge is removed from the set of free vertices $\mathcal{F}$. This is done in $O(1)$ time. Thus, the body of the sequential for loop in Lines 11-21 takes $O(1)$ time. Since it executes n iterations, the total time for executing the loop is $O(n)$, which is the running time of Phase 2.

The parallel running time of P-GRM is the sum of the running times of the two phases, that is, $O(n)$. Since P-GRM uses a maximum of $O(n^2)$ processors, the total amount of work performed by the algorithm is $O(n^3)$. □

In the case of dense graphs, P-GRM is not work efficient, that is, the amount of work performed by P-GRM is asymptotically higher than the running time of the sequential algorithm S-GRM, which is $O(n^2)$ when $m = \Theta(n^2)$. More generally, S-GRM performs $O(m + n \log n)$ work, whereas P-GRM performs $O(n^3)$ work independent of m. For sparse instances with $m = O(n)$ this gap is large, and our parallel algorithm is mainly justified in settings where massive parallelism is available and wall-clock time is the primary resource. For very dense graphs with $m = \Theta(n^2)$, the asymptotic work gap between $O(m + n \log n)$ and $O(n^3)$ reduces to a linear factor in n, and our experiments in Section 8 indicate that P-GRM can still exploit many-core parallelism to obtain substantial speedups in practice despite the higher theoretical work. We would like to emphasize that our main goal was to obtain a parallel approximation algorithm with a running time at most linear in the number of vertices that guarantees a solution within a constant factor (here 1/3) of the optimal solution for the rainbow matching in bipartite graphs. Even though the P-GRM algorithm is presented using the most powerful PRAM model, we also provide a practical implementation of the algorithm on a multi-core system and show that it obtains a good speedup in practice. Designing work-efficient parallel algorithms with slightly weaker approximation guarantees is an important direction for future work.

5.1 P-GRM: Illustrative Example

Figure 1 shows the execution of P-GRM on a bipartite graph with four vertices in each bipartition. The input edge-colored bipartite graph G is shown in Figure 1(a). P-GRM sorts the vertices in W in parallel and the sorted order is $\{w_1, w_3, w_4, w_2\}$. Assuming that vertices in W are ready for processing (after all parallel preprocessing steps), the first iteration of the main loop of P-GRM begins with vertex w_1, highlighted in yellow in Figure 1(b), during this first iteration and *matching phase*, two vertices, u_1 and u_3, independently and in parallel, tentatively select the edges (u_1, w_1) and (u_3, w_1), respectively. Both edges are colored blue. Since w_1 cannot be matched with more than one vertex (to satisfy the matching constraint), one of these edges must be eliminated. In this case, the edge (u_3, w_1) is removed.

Figure 1(c) illustrates the second iteration, in which the edge (u_1, w_1), colored blue, is successfully matched in previous iteration. In the beginning of each iteration, the endpoints are shown in gray mean that they are no longer eligible for future selection. The next vertex in the sorted sequence is w_3, highlighted in yellow. Vertex u_2 cannot even tentatively select the edge (u_2, w_3) since it is also colored blue, a color that has already been included in the matching $\mathcal{M}$. Selecting it would therefore violate the rainbow matching constraint. Consequently, in this iteration, only vertex u_3 tentatively selects the edge (u_3, w_3), which is colored red. As this is the only tentative edge in the current round, it is included in the global matching $\mathcal{M}$, signifying that it is permanently selected.

Figure 1(d) shows the subsequent iteration, in which vertex w_4 is processed. Similar to the previous case, vertex u_2 cannot tentatively select the edge (u_2, w_4) because it is colored red, which has already been used. Therefore, only vertex u_4 tentatively selects the edge (u_4, w_4). As this edge is the only tentative match in the current iteration, it is also added to the final matching $\mathcal{M}$.

Figure 1(e) illustrates the processing of the final vertex in the sorted list, namely w_2. At this stage, vertices u_1, u_3, and u_4 have already been matched in earlier iterations. The only remaining edge, (u_2, w_2), is colored black, which has already been used in a previous match. Consequently, vertex u_2 cannot form a valid matching without violating the rainbow matching constraint. As a result, no edge is selected in this iteration, and no additional edge is added to the final matching. Figure 1(f) shows the complete matching $\mathcal{M}$.

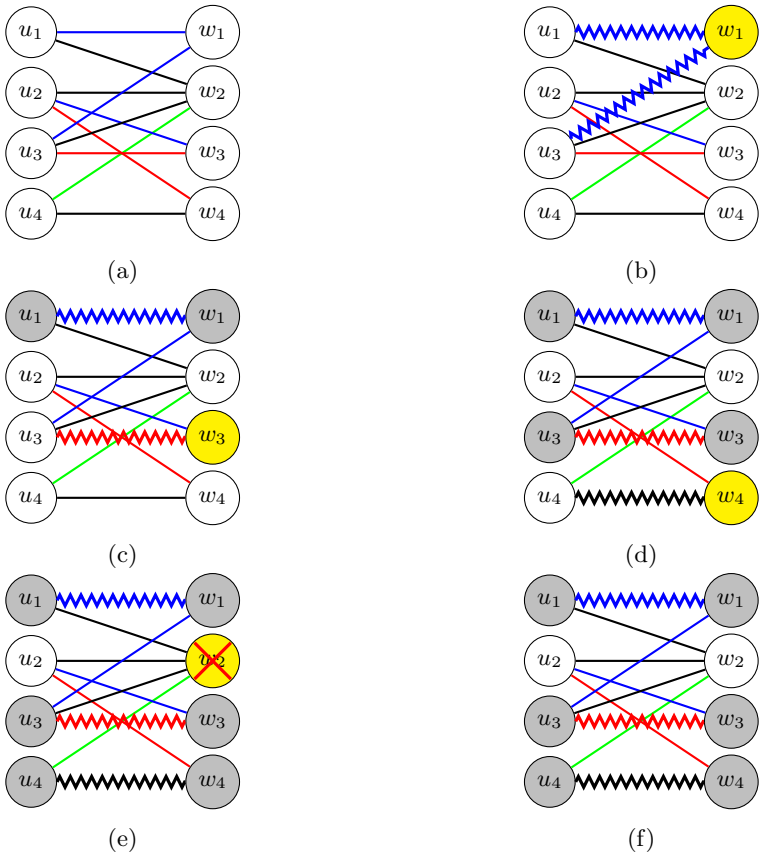

Fig. 1: P-GRM: Illustrative example. (a) the input graph G; (b) - (e) the matching obtained after each of the four iterations of the algorithm; (f) the obtained maximal rainbow matching. (The matching is represented using colored zigzag lines. A vertex marked with yellow is the vertex selected for processing in an iteration. Free vertices are white, while matched vertices are gray.)

6 S-GRM and P-GRM: Approximation Guarantees

In this section we prove the approximation guarantees of S-GRM and P-GRM. Since P-GRM is a parallel version of S-GRM and it produces exactly the same matching as S-GRM, it is enough to analyze the sequential algorithm. We use the standard notion: a polynomial-time approximation algorithm for a maximization problem has *approximation ratio* ρ if for every instance it produces a solution of value S such that $S \geq \rho \cdot S^*$, where S^* is the value of an optimal solution and $\rho < 1$.

Both S-GRM and P-GRM greedily build a *maximal* rainbow matching. We now show that any maximal rainbow matching is a 1/3-approximation of an optimal rainbow matching, which directly implies the desired guarantee for our algorithms. This mirrors the classical analysis of greedy algorithms for 3-dimensional matching.

Theorem 4. *The approximation ratio of S-GRM and P-GRM is* 1/3.

Proof. Let $\mathcal{M}$ be the rainbow matching returned by S-GRM (or P-GRM), and let $\mathcal{M}^*$ be an optimal rainbow matching for the same instance. Recall from Theorem 1 that $\mathcal{M}$ is maximal: no edge can be added to $\mathcal{M}$ without violating either the matching constraint (vertex-disjointness) or the rainbow constraint (distinct colors).

We use a charging argument. Every edge $e = (u, w)$ with color $\chi(e) = c$ has three *resources*:

- its left endpoint u,
- its right endpoint w,
- its color c.

Because $\mathcal{M}$ is maximal, for every edge $e^* = (u^*, w^*) \in \mathcal{M}^* \setminus \mathcal{M}$ at least one of its three resources (left endpoint u^*, right endpoint w^*, or color $\chi(e^*)$) is already used by some edge in $\mathcal{M}$. Otherwise, we could add e^* to $\mathcal{M}$ and obtain a strictly larger rainbow matching, contradicting maximality.

We now define a mapping (charging rule) from edges of $\mathcal{M}^*$ to edges of $\mathcal{M}$.

Charging rule. We conceptually replay the execution of S-GRM: edges are added to $\mathcal{M}$ one by one in a certain deterministic order. Initially, no edge from $\mathcal{M}^*$ is charged. When the algorithm adds an edge $f \in \mathcal{M}$, we do the following:

- Consider all edges $e^* \in \mathcal{M}^*$ that are still uncharged and that *first become invalid* at this point because of f, i.e., e^* shares at least one resource (an endpoint or the color) with f, and before adding f it was still feasible to add e^* to the current matching.
- Among those, assign (charge) each such edge e^* to the newly added edge f.

Edges $e^* \in \mathcal{M}^* \cap \mathcal{M}$ are charged to themselves when they are inserted into $\mathcal{M}$. Every edge in $\mathcal{M}^*$ is either in $\mathcal{M}$ or is blocked at some step by the first edge of $\mathcal{M}$ that uses one of its resources, so every edge of $\mathcal{M}^*$ is eventually charged to exactly one edge of $\mathcal{M}$.

We now bound how many edges from $\mathcal{M}^*$ can be charged to a single edge $f = (u, w)$ of color c in $\mathcal{M}$. Because $\mathcal{M}^*$ is itself a rainbow matching, the following hold:

- At most one edge in $\mathcal{M}^*$ uses the left endpoint u.
- At most one edge in $\mathcal{M}^*$ uses the right endpoint w.
- At most one edge in $\mathcal{M}^*$ uses the color c.

Therefore, in the entire process, at most three distinct edges from $\mathcal{M}^*$ can ever become invalid *for the first time* because of f: one via u, one via w, and one via c. Those are precisely the edges that get charged to f by our charging rule. Hence, each edge $f \in \mathcal{M}$ receives at most three charges.

Since every edge of $\mathcal{M}^*$ is charged to exactly one edge in $\mathcal{M}$, and each edge of $\mathcal{M}$ receives at most three charges, we obtain

$$|\mathcal{M}^*| \;\le\; 3\,|\mathcal{M}|.$$

Rearranging yields

$$|\mathcal{M}| \;\ge\; \frac{1}{3}\,|\mathcal{M}^*|,$$

so $\mathcal{M}$ is a 1/3-approximation of the optimal rainbow matching. This bound holds for any maximal rainbow matching and therefore for both S-GRM and P-GRM, in close analogy with the classical analysis for 3-dimensional matching. □

We remark that this analysis only relies on the fact that $\mathcal{M}$ is a maximal rainbow matching; it does not exploit the particular degree– and color–aware ordering used by S-GRM. Thus the 1/3 bound applies to any greedy procedure that builds a maximal rainbow matching.

To complement the analysis, it is useful to observe that greedy strategies may still produce solutions significantly smaller than the optimum.

7 Applications to Machine Learning and Data-Intensive Systems

Rainbow matchings naturally model diversity- and constraint-aware selection problems in machine learning and large-scale data processing. In this section we outline several scenarios in which the maximum cardinality rainbow matching problem on edge-colored bipartite graphs captures the structure of a machine learning or systems task, and in which S-GRM and P-GRM can be used as scalable combinatorial primitives. These are proposed use cases and conceptual mappings: we do not perform end-to-end machine learning experiments or claim empirical improvements in downstream metrics such as accuracy or fairness in this work.

7.1 Diverse mini-batch and coreset selection

Consider a bipartite graph $G = (U \cup W, E)$ where U represents data points (or data clusters) and W represents mini-batch "slots" or workers. Each edge $(u, w) \in E$ indicates that data point u is eligible to fill slot w, and the edge color $\chi(u, w)$ models a label, domain, or sensitive group (e.g., class, dataset source, or demographic attribute).

A rainbow matching in this graph corresponds to a set of pairwise non-conflicting assignments such that:

- each data point and each slot appears in at most one assignment, and
- each label/domain/group (color) is used at most once.

Thus, a large rainbow matching induces a mini-batch (or coreset) that is both non-redundant at the instance level and diverse across colors. S-GRM and P-GRM provide a way to construct such batches in time $O(m + n \log n)$ and $O(n)$, respectively, on very large graphs. In data-intensive regimes where batch construction itself is non-trivial, the parallel algorithm P-GRM is particularly suitable as a batch or coreset selection subroutine.

7.2 Fair and diverse recommendation

In a recommender-system setting, we can interpret U as users, W as items (or ads), and the color set Q as item categories, content providers, or protected groups. An edge (u, w) exists if item w is a viable recommendation for user u, and its color encodes the provider or group to which the item belongs.

A rainbow matching in this user–item graph then represents a set of recommendations in which:

- no user or item participates in more than one recommendation, and
- no provider or group (color) is over-represented.

Such a matching can be used as a post-processing step: starting from a large pool of candidate recommendations produced by a machine learning model, a maximal rainbow matching selects a large, conflict-free, and color-diverse subset. The 1/3-approximation guarantee implies that any maximal rainbow matching, including the one generated by S-GRM or P-GRM, captures at least one third of the maximum number of simultaneously enforceable diversity-constrained recommendations.

7.3 Graph-based and heterogeneous network learning

Many modern machine learning models operate on heterogeneous or relation-typed graphs, including knowledge graphs, user–item–context networks, and multi-relational interaction

graphs. In these graphs, edge types such as “friend-of”, “purchased”, or “clicked” can naturally be interpreted as colors.

When training graph-based models, we often need to subsample edges to form training batches, or to construct sparse views of a dense interaction network. A rainbow matching in an edge-colored bipartite subgraph $G = (U \cup W, E)$, where U and W are two node types and colors represent relation types, provides a subset of edges that is:

- conflict-free (no shared endpoints), which simplifies parallelism and avoids over-emphasizing a small set of nodes, and
- type-diverse (no repeated relation type), which enforces balanced coverage over edge types.

Using S-GRM or P-GRM as edge-sampling mechanisms yields maximal rainbow matchings with a guaranteed 1/3-approximation ratio, which is uncommon for heuristic sampling strategies used in practice.

7.4 Resource allocation in distributed training systems

At the systems level, distributed training and large-scale inference pipelines must assign jobs to heterogeneous compute resources under various diversification and locality constraints. We can model such a scenario with a bipartite graph where U is the set of training jobs or model components, W is the set of machines or accelerators, and the color of an edge (u, w) encodes a resource type, failure domain, or time slot.

A rainbow matching selects a set of job–machine assignments such that each job and each machine appears at most once and each resource type or domain is used at most once. This captures, for example, the requirement that within a scheduling round or batch, jobs are spread across:

- different hardware types (e.g., GPUs from different generations),
- different failure domains (e.g., racks or availability zones), or
- different time intervals or quotas.

In this context, P-GRM can be viewed as a parallel scheduling primitive that computes, in $O(n)$ time on a CRCW PRAM with n^2 processors, a large, diverse set of assignments that respect both matching and color constraints. The OpenMP implementation evaluated in Section 8 demonstrates that such a primitive is practical on multi-core machines and scales to graphs with hundreds of millions of edges.

8 EXPERIMENTAL ANALYSIS

8.1 Experimental setup

System setup. All experiments were performed on a dedicated server with `CentOS Linux 7` and kernel `5.15.80-200.el7.x86_64`. The server houses an AMD EPYC 74F3 “Milan” CPU, 24 physical Zen 3 cores with simultaneous multithreading, giving 48 logical cores at 3.19 GHz. The server carries 256 GB of DDR4-3200 RAM. Each core owns 32 KB L1 data and 32 KB L1 instruction caches, has a 512 KB private L2, and all cores share a 256 MB L3 cache. The code[1] was written in C++17 with OpenMP and compiled with `GCC 7.3.0` using `-O3 -march=native -fopenmp`, which enables OpenMP 4.5.

[1] The code implementing the algorithms will be made available on github upon publication of the paper, together with data-generation scripts and configuration files to facilitate reproducibility of the experiments.

Table 1: Parameters of graph instances

Parameter	Values
Number of vertices, n	$\{0.5, 1, 1.5\} \times 10^4$
Number of edges, m	$[0.5 \times 10^6, \ldots, 180 \times 10^6]$
Graph density, δ	0.2, 0.45, 0.8
Number of colors, q	$n/4$, $3n/4$

Data sets. Since we are not aware of any available data sets of instances of colored bipartite graphs, we generated several large edge-colored bipartite graphs with up to 180 million edges using the Erdős–Rényi $G(n, m)$ model [16]. Vertices are split evenly into U and W; edges join the sets and each edge gets a random color. For every size–density–color combination we drew edges uniformly, and colored them independently. We store each generated graph into an `mtx` type file. The first line lists sizes of the partitions, number of edges, and number of colors. Every other line stores one edge as (u, w, c) with $u \in U$, $w \in W$, and c the color. Table 1 shows the combination of parameters for the generated edge-colored bipartite graph instances used in the experiments.

8.2 Experimental Results

To analyze the performance of our parallel greedy rainbow matching algorithm (P-GRM) we conducted several experiments. We investigate the performance of the algorithm in terms of *execution time* and *speedup* with respect to parameters such as the number of vertices in each bipartition (assuming symmetric bipartite graphs), the density of the graph, and the number of distinct colors for edges. Using different number of cores, we ran P-GRM along with the sequential greedy rainbow matching algorithm (S-GRM) five times on each instance. We consider that the *speedup* is defined as the ratio of the running time of S-GRM and the running time of P-GRM. The plots in this section show the averages of execution times and speedups obtained over five runs.

We analyze the execution time of both S-GRM and P-GRM and the speedup obtained by P-GRM on systems with the number of cores ranging from 2 to 32 for the 18 groups of instances (different combination of number of vertices, density, and number of colors). In the rest of this section we investigate the effect of the aforementioned parameters on the performance of P-GRM.

The impact of number of vertices. Figures 2 and 3 show the speedup and execution time of P-GRM versus the number of cores for graphs with $n = \{0.5, 1, 1.5\} \times 10^4$ vertices per partition, fixed number of colors $q = 3n/4$, and densities $\delta = 0.2, 0.45, 0.8$.

For smaller graphs ($n = 0.5 \times 10^4$), the speedup plateaus at higher number of cores due to parallelization overhead. For graphs with $\delta = 0.2$, the speedup peaks at about 4.43 for 16 cores (Fig. 2 (a)), but drops to 4.17 for 32 cores due to thread contention. The speedup scales better for larger graphs: with $n = 1 \times 10^4$ and $\delta = 0.45$, except for 8 and 16 cores, the speedup increases almost linearly from 1.85 (for 2 cores) to 4.83 (for 32 cores) (Fig. 2(b)). The largest graphs ($n = 1.5 \times 10^4$) achieve the highest speedup: for $\delta = 0.8$, the speedup reaches 5.908 for 32 cores (Fig. 2(c)) showing the impact of larger graph instances. For $n = 1.5 \times 10^4$ and sparse graphs ($\delta = 0.2$) P-GRM achieves a speedup of 5.78 for 32 cores (Fig. 2(a)), while for dense graphs ($\delta = 0.8$) it achieves a speedup of 5.90. This highlights that larger, denser graphs provide more parallelism allowing P-GRM to obtain good speedup.

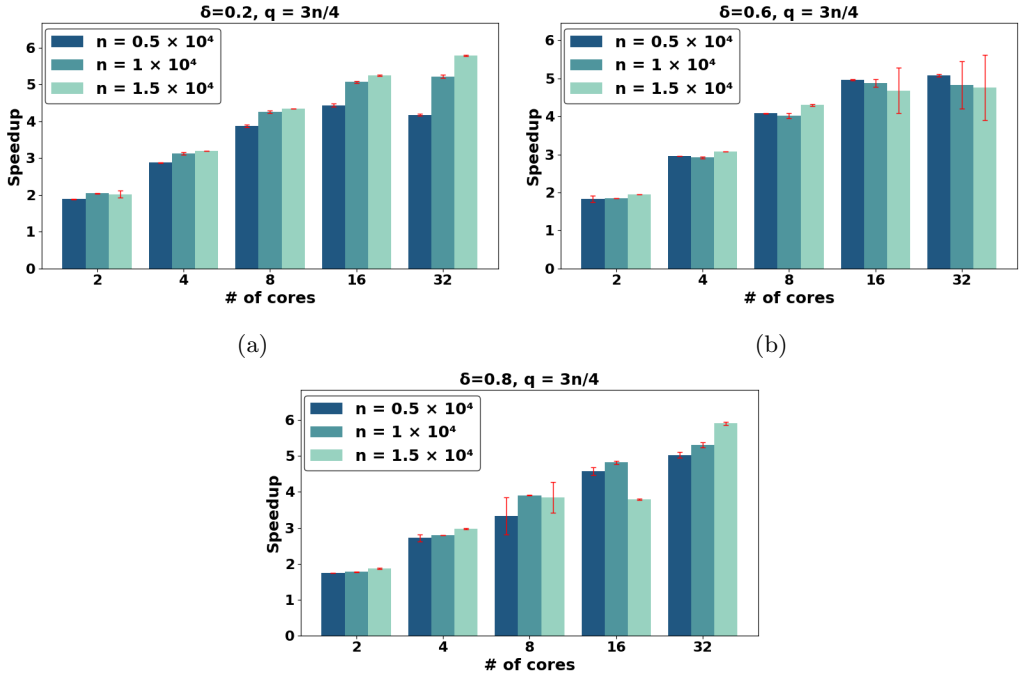

Fig. 2: P-GRM speedup vs. number of cores: graphs with $n = \{0.5, 1, 1.5\} \times 10^4$ vertices, $q = 3n/4$ colors, and number of edges ranging from 0.5 to 180 million. The speedup reaches up to 5.908 for large graphs and 32 cores.

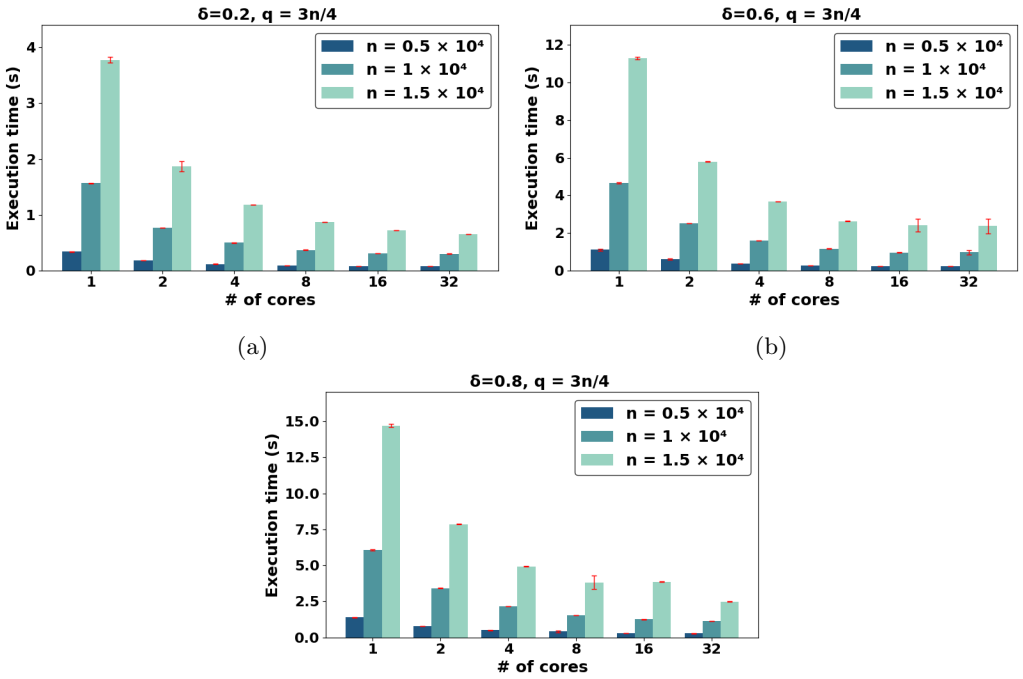

Fig. 3: P-GRM execution time vs. number of cores: graphs with $n = \{0.5, 1, 1.5\} \times 10^4$ vertices, $q = 3n/4$ colors, and number of edges ranging from 0.5 to 180 million. The bars corresponding to 1 core represent the execution time of S-GRM. As the number of cores increases, the execution time decreases significantly, especially for larger graphs.

The impact of number of colors. Figure 4 shows how varying the number of colors affects the speedup of P-GRM, considering graphs with density 0.8. We compare the performance in the case of small color palettes ($n/4$ colors) and rich color palettes ($3n/4$ colors). Increasing the number of colors generally leads to higher speedups, but there are some cases in which we observe something different. For instance, Figure 4 (c) shows that in the case of 8 and 16 cores (for the instance of size 1.5×10^4) we do not observe as much speedup as we do for 16 and 32 cores. Figure 4 (a), shows the impact of the number of colors on graphs of size $n = 0.5 \times 10^4$. In general, we observe an increase in the speedup as we increase the number of cores, but this increase saturates gradually.

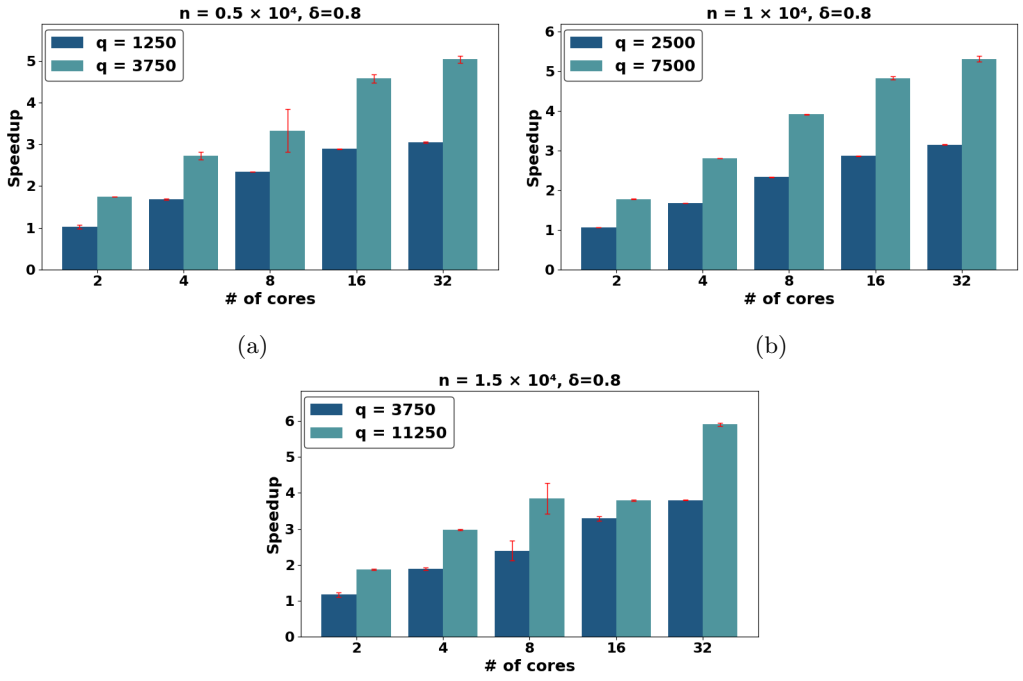

Fig. 4: P-GRM speedup vs. number of cores: graphs with $n = \{0.5, 1, 1.5\} \times 10^4$ vertices, density $\delta = 0.8$, under different number of colors q. The number of edges ranges from 0.5 to 180 million. A richer color palette leads to better speedup.

The impact of density. Figure 5 shows how the graph density affects the speedup of P-GRM for fixed number of colors $q = 3n/4$. In the case of the smallest graph ($n = 0.5 \times 10^4$, Figure 5 (a)), the speedup for 32 cores rises from 4.17 at $\delta = 0.2$ to 5.08 at $\delta = 0.45$ and decreases to 5.03 at $\delta = 0.8$. For graphs with $n = 1 \times 10^4$ (Figure 5 (b)), the speedup decreases from 5.21 to 4.83 and then increases again to 5.30. Lastly, for graphs with $n = 1.5 \times 10^4$ (Figure 5 (c)) and with the same consideration, the speedup decreases from 5.78 to 4.76 and then increases again to 5.90. P-GRM obtains better speedup in the case of larger graphs with lower densities.

Overall, the experiments show that P-GRM scales reasonably well with the number of cores and benefits from dense graphs and high color diversity. These trends provide practical guidance for applying the algorithm in large-scale graph processing scenarios where performance and runtime are critical.

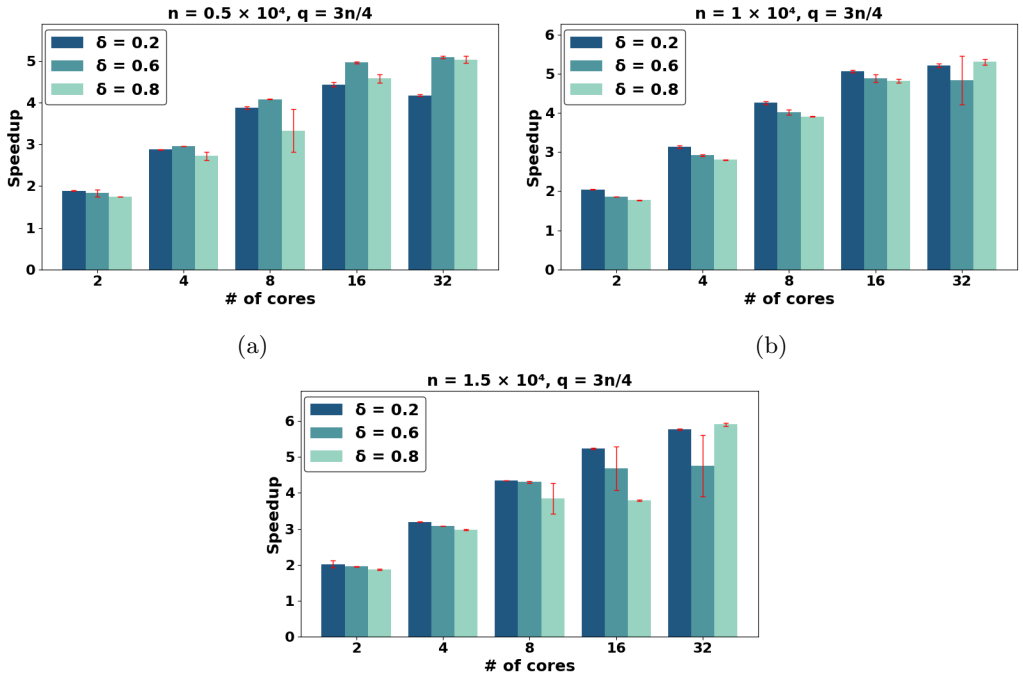

Fig. 5: P-GRM speedup vs. number of cores: graphs with $n = \{0.5, 1, 1.5\} \times 10^4$ vertices and $q = 3n/4$ colors, under different graph densities δ. The number of edges ranges from 0.5 to 180 million. Larger graphs with lower densities lead to better speedup overall.

9 Conclusion

In this paper, we proposed sequential (S-GRM) and parallel (P-GRM) 1/3-approximation algorithms for maximum cardinality rainbow matching in edge-colored bipartite graphs. P-GRM has a running time of $O(n)$ on a CRCW-PRAM with n^2 processors, while its sequential counterpart, S-GRM has a running time of $O(m + n \log n)$. We implemented both algorithms and performed an extensive experimental evaluation of the performance of P-GRM on a multi-core system. The results show that for large graphs with hundreds of millions of edges, the parallel algorithm achieves a good speedup relative to its sequential counterpart of up to 5.908, when using 32 cores.

One limitation of P-GRM is that it is not work efficient. Because our design goal was to guarantee the classical 1/3-approximation factor for a maximal rainbow matching and to obtain a fast parallel implementation, we did not attempt to minimize total work. In our future work, we plan to design parallel algorithms for rainbow matching that provide more relaxed approximation guarantees but are work efficient. This will allow the parallel algorithms to obtain better performance in practice. We also plan to design parallel randomized algorithms for the maximum cardinality rainbow matching in both bipartite and general graphs.

Beyond purely combinatorial and systems considerations, an interesting direction for future work is to integrate our algorithms as building blocks in concrete machine learning pipelines. As discussed earlier, maximal rainbow matchings can model diverse mini-batch or coreset construction, fair and diverse recommendation, graph-based learning on heterogeneous networks, and resource allocation in distributed training systems. Evaluating S-GRM and P-GRM on real user–item and job–machine data sets, and studying how the

size and diversity of the resulting rainbow matchings affect downstream accuracy and fairness, are natural next steps.

References

1. J. E. Hopcroft, R. M. Karp, An $n^{5/2}$ algorithm for maximum matching in bipartite graphs, SIAM J. Comput. 2 (1971) 225–231.
2. V. B. Le, F. Pfender, Complexity results for rainbow matchings, Theor. Comput. Sci. 524 (2014) 27–33.
3. M. R. Garey, D. S. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness, W. H. Freeman & Co., New York, NY, USA, 1979.
4. A. Itai, M. Rodeh, S. L. Tanimoto, Some matching problems for bipartite graphs, J. ACM 25 (4) (1978) 517–525.
5. P. Bennett, C. Cooper, A. Frieze, Rainbow greedy matching algorithms, in: A. Nikeghbali, P. M. Pardalos, M. T. Rassias (Eds.), Optimization, Discrete Mathematics, and Applications to Data Sciences, Springer Nature Switzerland, Cham, 2025, pp. 33–50.
6. M. Bannach, S. Berndt, M. Maack, M. Mnich, A. Lassota, M. Rau, M. Skambath, Solving packing problems with few small items using rainbow matchings, in: J. Esparza, D. Král' (Eds.), Proc. 45th Int. Symp. Math. Found. Comput. Sci. (MFCS), Vol. 170 of LIPIcs, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Dagstuhl, Germany, 2020, pp. 11:1–11:14.
7. S. Bessy, M. Bougeret, D. M. Thilikos, S. Wiederrecht, Kernelization for graph packing problems via rainbow matching, in: Proc. 2023 ACM-SIAM Symp. Discrete Algorithms (SODA), SIAM, Seattle, WA, 2023, pp. 3654–3663.
8. H. J. Ryser, Neuere probleme der kombinatorik, Vorträge über Kombinatorik, Oberwolfach 69 (91) (1967) 35.
9. S. Gupta, S. Roy, S. Saurabh, M. Zehavi, Parameterized algorithms and kernels for rainbow matching, Algorithmica 81 (2019) 1684–1698.
10. S. Gupta, S. Roy, S. Saurabh, M. Zehavi, Quadratic vertex kernel for rainbow matching, Algorithmica 82 (2020) 881–897.
11. S. Kelk, G. Stamoulis, Integrality gaps for colorful matchings, Discrete Optim. 32 (2019) 73–92.
12. M. Dyer, A. M. Frieze, B. G. Pittel, The average performance of the greedy matching algorithm, Ann. Appl. Probab. 3 (2) (1993) 526–552.
13. R. M. Karp, M. Sipser, Maximum matchings in sparse random graphs, in: Proc. 22nd IEEE Symp. Found. Comput. Sci. (FOCS), IEEE, Los Alamitos, CA, 1981, pp. 364–375.
14. J. Aronson, A. M. Frieze, B. G. Pittel, Maximum matchings in sparse random graphs: Karp-sipser revisited, Random Struct. Algorithms 12 (1998) 111–177.
15. R. Cole, Parallel merge sort, SIAM J. Comput. 17 (4) (1988) 770–785.
16. P. Erdős, A. Rényi, On random graphs i, Publicationes Mathematicae Debrecen 6 (1959) 290–297.