Two approximation algorithms for bipartite matching on multicore architectures

Fanny Dufossé[a], Kamer Kaya[b,*], Bora Uçar[c]

[a]Inria Lille, Nord Europe, 59650, Villeneuve d'Ascq, France
[b]Sabancı University, Faculty of Engineering and Natural Sciences, Istanbul, Turkey
[c]LIP, UMR5668 (CNRS - ENS Lyon - UCBL - Université de Lyon - INRIA), Lyon, France

Abstract

We propose two heuristics for the bipartite matching problem that are amenable to shared-memory parallelization. The first heuristic is very intriguing from a parallelization perspective. It has no significant algorithmic synchronization overhead and no conflict resolution is needed across threads. We show that this heuristic has an approximation ratio of around 0.632 under some common conditions. The second heuristic is designed to obtain a larger matching by employing the well-known Karp-Sipser heuristic on a judiciously chosen subgraph of the original graph. We show that the Karp-Sipser heuristic always finds a maximum cardinality matching in the chosen subgraph. Although the Karp-Sipser heuristic is hard to parallelize for general graphs, we exploit the structure of the selected subgraphs to propose a specialized implementation which demonstrates very good scalability. We prove that this second heuristic has an approximation guarantee of around 0.866 under the same conditions as in the first algorithm. We discuss parallel implementations of the proposed heuristics on a multicore architecture. Experimental results, for demonstrating speed-ups and verifying the theoretical results in practice, are provided.

Keywords: shared memory parallelism, matching, bipartite graphs, approximation algorithm
2010 MSC: 05C70, 68W10, 68W25

1. Introduction

We consider the maximum cardinality bipartite matching problem. A matching in a graph is a set of edges no two of which share a common vertex. The maximum cardinality matching problem asks for a matching of maximum size.
5 There are a number of polynomial time algorithms to solve this problem exactly.

*Corresponding author
Email addresses: fanny.dufosse@inria.fr (Fanny Dufossé), kaya@sabanciuniv.edu (Kamer Kaya), bora.ucar@ens-lyon.fr (Bora Uçar)

The lowest worst-case time complexity of the known algorithms is $\mathcal{O}(\sqrt{n}\tau)$ for a bipartite graph with n vertices and τ edges—the first of such algorithms is described by Hopcroft and Karp [1]. There is considerable interest in simpler and faster algorithms that have some approximation guarantee [2]. Such cheap algorithms are used as a jump-start routine by the current state of the art matching algorithms [2, 3, 4]. Furthermore, there are applications [5] where approximate cardinality matchings are used.

Most of the existing heuristics obtain good results in practice, but their worst-case guarantee is only around 1/2. Among those, the Karp-Sipser (KS) heuristic [6] is very well known. It finds maximum cardinality matchings in highly sparse (random) graphs but does not have a constant ratio approximation for denser ones (this algorithm will be reviewed later in Section 2). KS obtains very good results in practice. Currently, it is the suggested one to be used as a jump-start routine [2, 4] for exact algorithms, especially for augmenting-path based ones [7]. Algorithms that achieve an approximation ratio of $1-1/e$, where e is the base of the natural logarithm are designed for the online case [8]. Many of these algorithms are sequential in nature in that a sequence of greedy decisions are made in the light of the previously made decisions. Another heuristic is obtained by truncating the Hopcroft and Karp (HK) algorithm. HK, starting from a given matching, augments along a maximal set of shortest disjoint paths, until there are no augmenting paths. If one lets HK run until the shortest augmenting paths are of length k, then a $1 - 2/k$ approximate matching is obtained for $k \geq 3$. The run time of this heuristic is $\mathcal{O}(\tau k)$, for a bipartite graph with τ edges.

We propose two matching heuristics (Section 3) for bipartite graphs. Both heuristics construct a subgraph of the input graph by randomly choosing some edges. They then obtain a maximum matching in the selected subgraph and return it as an approximate matching for the input graph. The probability density function for choosing a given edge in both heuristics is obtained with a sparse matrix scaling method. The first heuristic is shown to deliver a constant approximation guarantee of 0.632 of the maximum cardinality matching under the condition that the scaling method has successfully scaled the input matrix. The second one builds on top of the first one and improves the approximation ratio to 0.866, under the same condition as in the first one. Both of the heuristics are designed to be amenable to parallelization in modern multicore systems. The first heuristic does not require a conflict resolution scheme. Furthermore, it does not have any synchronization requirements. The second heuristic employs KS to find a matching on the selected subgraph. We show that KS becomes an exact algorithm on those subgraphs. Further analysis of the properties of those subgraphs is carried out to design a specialized implementation of KS for efficient parallelization. The approximation guarantees of the two proposed heuristics do not deteriorate with the increased degree of parallelization, thanks to their design, which is usually not the case for parallel matching heuristics [9].

The organization of the paper is as follows. In Section 2, we give some background on matching and matching heuristics. This section also contains a brief summary of a well-known doubly stochastic scaling method, and a mathemati-

cal identity that we will need later. In Section 3, we propose the two matching heuristics, discuss their efficient parallelization, and analyze their approximation guarantee. During the analysis, we assume that bipartite graphs have two properties: (i) the same number of vertices in both vertex classes; (ii) each edge appears in a matching that contains all vertices. Under these criteria, the scaling algorithm summarized in Section 2 works successfully. Later on (Section 3.3), we discuss the bipartite graphs without these properties. We then give experiments in Section 4, where we observe the theoretical findings in practice (even with general bipartite graphs), and present parallelization results.

2. Notation and background

Let $G = (V_R \cup V_C, E)$ be a bipartite graph, where V_R and V_C are two vertex classes and E is the edge set. G can be represented as a sparse matrix $\mathbf{A}$. Each row (column) of $\mathbf{A}$ corresponds to a unique vertex in V_R (in V_C) so that $a_{ij} = 1$ if and only if $(v_i, v_j) \in E$. Using this correspondence, we refer to the vertices in the two classes as the row and column vertices. The number of edges incident on a vertex is called its degree. A *path* in a graph is a sequence of vertices such that each consecutive vertex pair share an edge. A vertex *is reachable from* another one, if there is a path between them. The *connected components* of a graph are the equivalence classes of vertices under the "is reachable from" relation. A *cycle* in a graph is a path whose start and end vertices are the same. A *simple cycle* is a cycle with no vertex repetitions. A *tree* is a connected graph with no cycles. A *spanning tree* of a connected graph G is a tree containing all vertices of G.

A directed graph $G_D = (V, E)$ with vertex set V and edge set E can be associated with an $n \times n$ sparse matrix $\mathbf{A}$. Here, $|V| = n$, and for each $a_{ij} \neq 0$ where $i \neq j$, we have a directed edge from v_i to v_j. A directed graph is strongly connected, if every vertex is reachable from every other vertex by following the directed edges.

2.1. Matching

A matching $\mathcal{M}$ in a bipartite graph $G = (V_R \cup V_C, E)$ is a subset of edges E where a vertex in $V_R \cup V_C$ is in at most one edge in $\mathcal{M}$. Given a matching $\mathcal{M}$, a vertex v is said to be *matched* by $\mathcal{M}$ if v is in an edge of $\mathcal{M}$, otherwise v is called *unmatched*. If all the vertices are matched by $\mathcal{M}$, then $\mathcal{M}$ is said to be a *perfect matching*. The *cardinality* of a matching $\mathcal{M}$, denoted by $|\mathcal{M}|$, is the number of edges in $\mathcal{M}$. The maximum cardinality matching problem asks for a matching of maximum size. There are a number of well-known, exact, and polynomial-time algorithms for this problem on bipartite graphs. A recent paper [7] gives a classification of those algorithms.

Parallel (exact) matching algorithms on modern architectures have been recently investigated. Azad et al. [9] study the implementations of a set of known bipartite graph matching algorithms on shared memory systems. Deveci et al. [10, 11] investigate the implementation of some known matching algorithms

or their variants on GPU. There are quite good speedups reported in these implementations, yet there are non-trivial instances where parallelism does not help (for any of the algorithms).

Our focus is on matching heuristics that have linear run time complexity and good quality guarantees on the size of the matching. Recent surveys of matching heuristics are given by Kaya et al. [4, Section 4] and Langguth et al. [2]. Two heuristics, called the cheap matching and Karp-Sipser heuristic, stand out and are suggested as initialization steps in the best two exact matching algorithms [7]. These two heuristics also attracted theoretical interest.

The *cheap matching* heuristic has two variants in the literature. The first variant randomly visits the edges and matches the two endpoints of an edge if they are both available. The theoretical performance guarantee of this heuristic is 1/2, i.e., the heuristic delivers matchings of size at least half of the maximum matching cardinality. This is analyzed theoretically [12] and shown to obtain results that are near the worst-case on certain classes of graphs. The second variant of the cheap matching heuristic repeatedly selects a random vertex and matches it with a random neighbor. The matched vertices, along with the ones which become isolated, are removed from the graph and the process continues until the whole graph is consumed. This variant also has a 1/2 worst-case approximation guarantee (see for example a proof by Pothen and Fan [13]), and it is somewhat better ($0.5 + \epsilon$ for $\epsilon \geq 0.0000025$ [14] which has been recently improved to $\epsilon \geq 1/256$ [15]).

We make use of the Karp-Sipser (KS) heuristic to design one of the proposed heuristics. We summarize KS here and refer the reader to the original paper [6]. The theoretical foundation of KS is that if there is a vertex v with exactly one neighbor (v is called *degree-one*), then there is a maximum cardinality matching in which v is matched with its neighbor. That is, matching v with its neighbor is an *optimal* decision. Using this observation, the KS heuristic runs as follows. Check whether there is a degree-one vertex; if so then match the vertex with its unique neighbor and delete both vertices (and the edges incident on them) from the graph. Continue this way until the graph has no edges (in which case we are done) or all remaining vertices have degree larger than one. In the latter case, pick a random edge, match the two endpoints of this edge, and delete those vertices and the edges incident on them. Then repeat the whole process on the remaining graph. The phase before the first random choice of edges made by the KS algorithm is called Phase 1, and the rest is called Phase 2 (where new degree-one vertices may arise). The run time of this heuristic is linear. This heuristic matches all but $\tilde{O}(n^{1/5})$ vertices of a random undirected graph [16]. One disadvantage of KS is that because of the degree dependencies of the vertices to the already matched vertices, an efficient parallelism is hard to achieve (a list of degree-one vertices needs to be maintained). That is probably why some inflicted forms (successful but without any known quality guarantee) of this heuristic were used in recent studies [9].

Recent studies focusing on approximate matching algorithms on parallel systems include heuristics for graph matching problem [17, 18, 19] and also heuristics used for initializing bipartite matching algorithms [9]. Lotker et al. [20]

140 present a distributed $1-1/k$ approximate matching algorithm for bipartite graphs. This nice theoretical algorithm has $O(k^3 \log \Delta + k^2 \log n)$ time steps with a message length of $O(\log \Delta)$ where Δ is the maximum degree of a vertex and n is the number vertices in the graph. Blelloch et al. [21] propose an algorithm to compute maximal matchings (1/2 approximate) with $O(\tau)$ *work*
145 and $O(\log^3 \tau)$ *depth* with high probability on a bipartite graph with τ edges. This is an elaboration of the cheap matching heuristic for parallel systems. Although the performance metrics *work* and *depth* are quite impressive, the approximation guarantee stays as in the serial variant. A striking property of this heuristic is that it trades parallelism and reduced work while always finding the
150 same matching (including those found in the sequential version). Birn et al. [22] discuss maximal (1/2 approximate) matchings in $O(\log^2 n)$ time and $O(\tau)$ work in CREW PRAM model. Practical implementations on distributed memory and GPU systems are discussed—a shared memory implementation is left as future work in the cited paper.

155 *2.2. Scaling matrices to doubly stochastic form*

An $n \times n$ matrix $\mathbf{A} \neq 0$ is said to have *support* if there is a perfect matching in the associated bipartite graph. An $n \times n$ matrix $\mathbf{A}$ is said to have *total support* if each edge in its bipartite graph can be put into a perfect matching. A square sparse matrix is called irreducible if its directed graph is strongly
160 connected. A square sparse matrix $\mathbf{A}$ is called fully indecomposable if for a permutation matrix $\mathbf{Q}$, the matrix $\mathbf{B} = \mathbf{AQ}$ has a zero free diagonal and the directed graph associated with $\mathbf{B}$ is irreducible. Fully indecomposable matrices have total support; but a matrix having total support could be a block diagonal matrix, where each block is fully indecomposable. For more formal definitions
165 of support, total support, and the fully indecomposability, see for example the book by Brualdi and Ryser [23, Ch. 3 and Ch. 4]. Any nonnegative matrix $\mathbf{A}$ with total support can be scaled with two (unique) positive diagonal matrices $\mathbf{D}_R$ and $\mathbf{D}_C$ such that $\mathbf{D}_R\mathbf{A}\mathbf{D}_C$ is doubly stochastic (that is, the sum of entries in any row and in any column of $\mathbf{D}_R\mathbf{A}\mathbf{D}_C$ is equal to one). If $\mathbf{A}$ has support but
170 not total support, then $\mathbf{A}$ can be scaled to a doubly stochastic matrix but not with two positive diagonal matrices [24]—this fact is also seen in more recent treatments [25, 26, 27]).

The Sinkhorn-Knopp algorithm [24] is a well-known method for scaling matrices to doubly stochastic form. This algorithm generates a sequence of matrices
175 (whose limit is doubly stochastic) by normalizing the columns and the rows of the sequence of matrices alternately. That is, the initial matrix is normalized such that each column has sum one. Then, the resulting matrix is normalized so that each row has sum one and so on so forth.

We use a parallel implementation of the Sinkhorn-Knopp scaling method,
180 shown in Algorithm 1, but other doubly stochastic scaling methods [25, 26, 27, 28] can also be used. In Algorithm 1, $\mathbf{A}_{i*}$ and $\mathbf{A}_{*j}$ are the sets of column and row indices of the nonzeros at the ith row and jth column of $\mathbf{A}$, respectively. Instead of the diagonal scaling matrices $\mathbf{D}_r$ and $\mathbf{D}_c$, we use two arrays $\mathbf{d}_r$ and $\mathbf{d}_c$ to store the (diagonal) entries of the scaling matrices. As is seen, given an error

---

Algorithm 1 ScaleSK: Parallel Sinkhorn-Knopp scaling

Input $\mathbf{A}$: an $n \times n$ matrix with total support, ε: the error threshold
Output $\mathbf{d}_r$, $\mathbf{d}_c$: row/column scaling arrays

1: **for** $i = 1$ to n **in parallel do**
2: $\quad \mathbf{d}_r[i] \leftarrow 1$
3: $\quad \mathbf{d}_c[i] \leftarrow 1$
4: **for** $j = 1$ to n **in parallel do**
5: $\quad csum[j] \leftarrow \sum_{i \in \mathbf{A}_{*j}} a_{ij}$
6: **repeat**
7: $\quad$ **for** $j = 1$ to n **in parallel do**
8: $\quad\quad \mathbf{d}_c[j] \leftarrow 1/csum[j]$
9: $\quad$ **for** $i = 1$ to n **in parallel do**
10: $\quad\quad rsum \leftarrow \sum_{j \in \mathbf{A}_{i*}} a_{ij} \times \mathbf{d}_c[j]$
11: $\quad\quad \mathbf{d}_r[i] \leftarrow 1/rsum$
12: $\quad$ **for** $j = 1$ to n **in parallel do**
13: $\quad\quad csum[j] \leftarrow \sum_{i \in \mathbf{A}_{*j}} \mathbf{d}_r[i] \times a_{ij}$
14: **until** $|\max\{1 - csum[j] : 1 \leq j \leq n\}| \leq \varepsilon$

---

185 threshold, the method runs until convergence, where we want to stop when both the row sums and column sums are in the ε closure of one. At each iteration, we first balance the columns and then the rows, at which point the row sums are one (modulo round-off errors), but the column sums are not. The stopping criteria for convergence is therefore to have the maximum difference between
190 the column sums and one as small as possible. At the end, $\mathbf{d}_r[i] \times a_{ij} \times \mathbf{d}_c[j]$ gives the scaled entry. There are techniques to improve the parallel performance of Algorithm 1. For example, in case of skewness in degree distributions, one can assign multiple threads to a single row with many nonzeros. However, we do not focus on this issues here.

195 *2.3. A mathematical fact*

We will make use of the following identity, whose proof is given for the sake of completeness.

Lemma 1. *Let $(u_i)_{i \in S}$ and $(v_i)_{i \in S}$ be two sequences and $S = \{1, 2, \ldots, |S|\}$ be their index set. Then we have*

$$\prod_{i \in S}(u_i + v_i) = \sum_{S' \subseteq S} \left(\prod_{i \in S'} u_i\right) \cdot \left(\prod_{i \in S \setminus S'} v_i\right).$$

Proof. We prove this formula by induction on the cardinality of S. The equality is evident for $|S| = 1$. Assume that it holds also for all sequences with $|S| < k$. Now let $|S|$ be k and S'' be $S \setminus \{k\}$. Then by the inductive hypothesis, the

product $\prod_{i \in S}(u_i + v_i)$ is equal to

$$
\begin{aligned}
& (u_k + v_k) \cdot \sum_{S' \subseteq S''} \left(\prod_{i \in S'} u_i \right) \cdot \left(\prod_{i \in S'' \setminus S'} v_i \right) \\
= \quad & u_k \sum_{S' \subseteq S''} \left(\prod_{i \in S'} u_i \right) \cdot \left(\prod_{i \in S'' \setminus S'} v_i \right) + v_k \sum_{S' \subseteq S''} \left(\prod_{i \in S'} u_i \right) \cdot \left(\prod_{i \in S'' \setminus S'} v_i \right) \\
= \quad & \sum_{S' \subseteq S, k \in S'} \left(\prod_{i \in S'} u_i \right) \cdot \left(\prod_{i \in S \setminus S'} v_i \right) + \sum_{S' \subseteq S, k \notin S'} \left(\prod_{i \in S'} u_i \right) \cdot \left(\prod_{i \in S \setminus S'} v_i \right) \\
= \quad & \sum_{S' \subseteq S} \left(\prod_{i \in S'} u_i \right) \cdot \left(\prod_{i \in S \setminus S'} v_i \right) .
\end{aligned}
$$

□

3. Two matching heuristics

200 We propose two simple matching heuristics for the maximum cardinality bipartite matching problem that are efficiently parallelizable and have guaranteed approximation ratios. The first heuristic does not require synchronization nor conflict resolution assuming that the write operations to the memory are atomic (which is discussed in Section 4). This heuristic and its approximation 205 guarantee of around 0.632 are described in the following subsection. The second heuristic is designed to obtain larger matchings compared to those obtained by the first one. This heuristic employs the Karp-Sipser heuristic on a judiciously chosen subgraph of the input graph. We show that for this subgraph, the KS heuristic is an exact algorithm, and a specialized, efficient implementation of KS 210 is possible to obtain matchings of size around 0.866 of the maximum cardinality.

3.1. One-sided matching

The first matching heuristic we propose, OneSidedMatch, scales the given adjacency matrix $\mathbf{A}$ (each a_{ij} is originally either 0 or 1) and uses the scaled entries to randomly choose a column as a match for each row. The pseudocode 215 of the heuristic is shown in Algorithm 2.

OneSidedMatch first obtains the scaling vectors $\mathbf{d}_r$ and $\mathbf{d}_c$ corresponding to a doubly stochastic matrix $\mathbf{S}$ (line 1). After initializing the `cmatch` array, for each row i of $\mathbf{A}$, the heuristic randomly chooses a column $j \in \mathbf{A}_{i*}$ based on the probabilities computed by using corresponding scaled entries of row i. It then 220 matches i and j. Clearly multiple rows can choose the same column and write to the same entry in `cmatch`. We assume that in the parallel, shared-memory setting, one of the write operation survives, and the `cmatch` array defines a valid matching, i.e., $\{\{\texttt{cmatch}[j], j\} : cmatch[j] \neq NIL\}$. We now analyze its approximation guarantee in terms of the matching cardinality.

Algorithm 2 ONESIDEDMATCH

Input $\mathbf{A}$: an $n \times n$, (0,1)-matrix with total support
Output `cmatch`$[\cdot]$: the rows matched to columns
1: $(\mathbf{d}_r, \mathbf{d}_c) \leftarrow$ SCALESK($\mathbf{A}$)
2: **for** $j = 1$ **to** n **in parallel do**
3: `cmatch`$[j] \leftarrow NIL$
4: **for** $i = 1$ **to** n **in parallel do**
5: Pick a random column $j \in \mathbf{A}_{i*}$ by using the probability density function

$$\frac{s_{ik}}{\Sigma_{\ell \in \mathbf{A}_{i*}} s_{i\ell}}, \text{for all } k \in \mathbf{A}_{i*}$$

where $s_{ik} = \mathbf{d}_r[i] \times \mathbf{d}_c[k]$ is the corresponding entry in the scaled matrix $\mathbf{S} = \mathbf{D}_R \mathbf{A} \mathbf{D}_C$.
6: `cmatch`$[j] \leftarrow i$

225 **Theorem 1.** *Let* $\mathbf{A}$ *be an* $n \times n$*, (0,1)-matrix with total support. Then,* ONESIDEDMATCH *obtains a matching of size at least* $n(1-1/e) \approx 0.632n$ *in expectation as* $n \to \infty$*.*

Proof. To compute the matching cardinality, we will count the columns that are not picked by any row and subtract it from n. Since $\Sigma_{k \in \mathbf{A}_{i*}} s_{ik} = 1$ for each row i of $\mathbf{S}$, the probability that a column j is not picked by any of the rows in $\mathbf{A}_{*j}$ is equal to $\prod_{i \in \mathbf{A}_{*j}} (1 - s_{ij})$. By applying the arithmetic-geometric mean inequality, we obtain

$$\sqrt[d_j]{\prod_{i \in \mathbf{A}_{*j}} (1 - s_{ij})} \leq \frac{d_j - \sum_{i \in \mathbf{A}_{*j}} s_{ij}}{d_j} \ ,$$

where $d_j = |\mathbf{A}_{*j}|$ is the degree of column vertex j. Therefore,

$$\prod_{i \in \mathbf{A}_{*j}} (1 - s_{ij}) \leq \left(1 - \frac{\sum_{i \in \mathbf{A}_{*j}} s_{ij}}{d_j}\right)^{d_j} \ .$$

Since $\mathbf{S}$ is doubly stochastic, we have $\sum_{i \in \mathbf{A}_{*j}} s_{ij} = 1$ and

$$\prod_{i \in \mathbf{A}_{*j}} (1 - s_{ij}) \leq \left(1 - \frac{1}{d_j}\right)^{d_j} \ .$$

The function on the right hand side above is an increasing one, and has the limit

$$\lim_{d_j \to \infty} \left(1 - \frac{1}{d_j}\right)^{d_j} = \frac{1}{e} \ ,$$

where e is the base of the natural logarithm. By the linearity of expectation, the expected number of unmatched columns is no larger than $\frac{n}{e}$. Hence, the
230 cardinality of the matching is no smaller than $n\,(1 - 1/e)$. □

In Algorithm 2, we split the rows among the threads with a **parallel for** construct. For each row i, the corresponding thread chooses a random number r from a uniform distribution with range $(0, \sum_{k \in \mathbf{A}_{i*}} s_{ik}]$. Then, the last nonzero column index j for which $\sum_{1 \le k \le j} s_{ik} \le r$ is found and `cmatch`$[j]$ is set to i.
235 Since no synchronization or conflict detection is required, the heuristic promises significant speedups.

3.2. Two-sided matching

OneSidedMatch's approximation guarantee and suitable structure for parallel architectures make it a good cheap matching heuristic. The natural ques-
240 tion that follows is whether a heuristic with a better guarantee exits. Of course, the sought heuristic should also be simple and easy to parallelize. We asked: "what happens if we repeat the process for the other (column) side of the bipartite graph"? The question led us to the following algorithm. Let each row select a column, and let each column select a row. Take all these $2n$ choices to
245 construct a bipartite graph G (a subgraph of the input) with $2n$ vertices and at most $2n$ edges (if i chooses j and j chooses i, we have one edge), and seek a maximum cardinality matching in G. Since the number of edges is at most $2n$, any exact matching algorithm on this graph would be fast—in particular the worst case run time would be $\mathcal{O}(n^{1.5})$ [1]. Yet, we can do better and ob-
250 tain a maximum cardinality matching in linear time by running the Karp-Sipser heuristic on G, as we display in Algorithm 3.

The most interesting component of TwoSidedMatch is the incorporation of the Karp-Sipser heuristic for two reasons. First, although it is only a heuristic, KS computes a maximum cardinality matching on the bipartite graph G
255 constructed in Algorithm 3. Second, although KS has a sequential nature, we can obtain good speedups with a specialized parallel implementation. In general, it is hard to parallelize (non-trivial) graph algorithms, and it is even harder when the overall cost is $\mathcal{O}(n)$, which is the case for KS on G. We give a series of lemmas below which enables us to use KS as an exact algorithm with a good
260 shared-memory parallel performance.

The first lemma describes the structure of G constructed at line 8 of TwoSidedMatch.

Lemma 2. *Each connected component of G constructed in Algorithm 3 contains at most one simple cycle.*

265 *Proof.* A connected component $M \subseteq G$ with n' vertices can have at most n' edges. Let T be a spanning tree of M. Since T contains $n' - 1$ edges, the remaining edge in M can create at most one cycle when added to T. □

Lemma 2 explains why KS is an exact algorithm on G. If a component does not contain a cycle, KS consumes all its vertices in Phase 1. Therefore, all of
270 the matching decisions given by KS are optimal for this component. Assume a component contains a simple cycle. After Phase 1, the component is either consumed, or due to Lemma 2, it is reduced to a simple cycle. In the former case, the matching is a maximum cardinality one. In the latter case, an arbitrary edge

Algorithm 3 TwoSidedMatch

Input $\mathbf{A}$: an $n \times n$, (0,1)-matrix with total support
Output `match`$[\cdot]$: the mate of each vertex or *NIL*

1: $(\mathbf{d}_r, \mathbf{d}_c) \leftarrow$ ScaleSK($\mathbf{A}$)
2: **for** $i = 1$ **to** n **in parallel do**
3: Pick a random column $j \in \mathbf{A}_{i*}$ by using the probability density function

$$\frac{s_{ik}}{\Sigma_{\ell \in \mathbf{A}_{i*}} s_{i\ell}}, \text{ for all } k \in \mathbf{A}_{i*}$$

where $s_{ik} = \mathbf{d}_r[i] \times \mathbf{d}_c[k]$ is the corresponding entry in the scaled matrix $\mathbf{S} = \mathbf{D}_R \mathbf{A} \mathbf{D}_C$.
4: `rchoice`$[i] \leftarrow j$
5: **for** $j = 1$ **to** n **in parallel do**
6: Pick a random row $i \in \mathbf{A}_{*j}$ by using the probability density function

$$\frac{s_{kj}}{\Sigma_{\ell \in \mathbf{A}_{*j}} s_{\ell j}}, \text{ for all } k \in \mathbf{A}_{*j}.$$

7: `cchoice`$[j] \leftarrow i$
8: Construct a bipartite graph $G = (V_R \cup V_C, E)$ where

$$E = \{\{i, \texttt{rchoice}[i]\} : i \in \{1, \ldots, n\}\} \cup \{\{\texttt{cchoice}[j], j\} : j \in \{1, \ldots, n\}\}.$$

9: `match` $\leftarrow$KarpSipser(G)

of the cycle can be used to match a pair of vertices. This decision necessarily 275 leads to a unique perfect matching in the remaining simple path. These two arguments can be repeated for all the connected components to see that the KS heuristic finds a maximum cardinality matching in G.

Algorithm 4 describes our parallel KS implementation KarpSipserMT. The graph is represented using a single array `choice`, where `choice`$[u]$ is the ver- 280 tex randomly chosen by $u \in V_R \cup V_C$. The `choice` array is a concatenation of the arrays `rchoice` and `cchoice` set in TwoSidedMatch. Hence, an explicit graph construction for G (line 8 of Algorithm 3) is not required, and a transformation of the selected edges to a graph storage scheme is avoided. KarpSipserMT uses three atomic operations for synchronization. The first 285 operation _Add(*memory, value*) atomically adds a *value* to a *memory* location. It is used to compute the vertex degrees in the initial graph (line 9). The second operation _CompAndSwap(*memory, value, replace*) first checks whether the *memory* location has the *value*. If so, its content is *replace*d. The final content is returned. The third operation _AddAndFetch(*memory, value*) atomically 290 adds a given *value* to a *memory* location and the final content is returned. We will describe the use of these two operations later.

KarpSipserMT has two phases which correspond to the two phases of KS. The first phase of KarpSipserMT is similar to that of KS in that optimal matching decisions are made about some degree-one vertices. The second phase

Algorithm 4 KarpSipserMT

```
Input G = {V, choice[·]: the chosen vertex for each u ∈ V}
Output match[·]: the match array for u ∈ V
 1: for all u ∈ V in parallel do
 2:   mark[u] ← 1
 3:   deg[u] ← 1
 4:   match[u] ← NIL
 5: for all u ∈ V in parallel do
 6:   v ← choice[u]
 7:   mark[v] ← 0
 8:   if choice[v] ≠ u then
 9:     _Add(deg[v], 1)
10: for each vertex u in parallel do        ▶ Phase 1: out-one vertices
11:   if mark[u] = 1 then
12:     curr ← u
13:     while curr ≠ NIL do
14:       nbr ← choice[curr]
15:       if _CompAndSwap(match[nbr], NIL, curr) = curr then
16:         match[curr] ← nbr
17:         curr ← NIL
18:         next ← choice[nbr]
19:         if match[next] = NIL then
20:           if _AddAndFetch(deg[next], −1) = 1 then
21:             curr ← next
22:           else
23:             curr ← NIL
24: for each column vertex u in parallel do        ▶ Phase 2: the rest
25:   v ← choice[u]
26:   if match[u] = NIL and match[v] = NIL then
27:     match[u] ← v
28:     match[v] ← u
```

295 of KarpSipserMT handles remaining vertices very efficiently, without bothering with their degrees. The following definitions are used to clarify the difference between an original KS implementation and KarpSipserMT.

Definition 1. Given a matching and the array `choice`, let u be an unmatched vertex and $v = \texttt{choice}[u]$. Then u is called:

300
- *out-one*, if v is unmatched, and no unmatched vertex w with $\texttt{choice}[w] = u$ exists.
- *in-one*, if v is matched, and only a single unmatched vertex w with $\texttt{choice}[w] = u$ exists.

The first phase of KarpSipserMT (**for** loop of line 10) does not track and
305 match all degree-one vertices. Instead, only the out-one vertices are taken into account. For each such vertex u that is already out-one before Phase 1, we have $\texttt{mark}[u] = 1$. KarpSipserMT visits these vertices (lines 10-11). Newly arising out-one vertices are consumed right away without maintaining a list. The second phase of KarpSipserMT (**for** loop of line 24) is much simpler
310 than that of KS as the degrees of the vertices are not tracked/updated. We now discuss how these simplifications are possible while ensuring a maximum cardinality matching in G.

Observation 1. *An out-one or an in-one vertex is a degree-one in KS.*

Observation 2. *A degree-one vertex in KS is either an out-one or an in-*
315 *one vertex, or it is one of the two vertices u and v in a 2-clique such that $v = \texttt{choice}[u]$ and $u = \texttt{choice}[v]$.*

Lemma 3. *If there exists an in-one vertex in G at any time during the execution of* KarpSipserMT*, an out-one vertex also exists.*

Proof. Let u be an in-one vertex and let v be the unmatched vertex such that
320 $\texttt{choice}[v] = u$. Let $\mathcal{P}$ be the longest vertex sequence $w_1, w_2, \ldots, w_k, u$ such that all w_is are unmatched, $\texttt{choice}[w_i] = w_{i+1}$ for $1 \leq i < k$, and $v = w_k$. If $\mathcal{P}$ has a finite length, then w_1 is an out-one vertex and we are done. On the other hand, if $\mathcal{P}$ has an infinite length it must contain a cycle. Furthermore, u must be in this cycle, since each w_i's next vertex, which also needs to be in the
325 cycle, is uniquely defined by `choice`. But u is an in-one vertex and $\texttt{choice}[u]$ is already matched. Thus $\mathcal{P}$ has a finite length, and an out-one vertex (w_1) always exists. □

According to Observation 1, all the matching decisions given by KarpSipserMT in Phase 1 are optimal, since an out-one vertex is a degree-one
330 vertex. Observation 2 implies that among all the degree-one vertices, KarpSipserMT ignores only the in-ones and 2-cliques. According to Lemma 3, an in-one vertex cannot exist without an out-one vertex, therefore they are handled in the same phase. The 2-cliques that survive Phase 1 are handled in KarpSipserMT's Phase 2, since they can be considered as cycles.

335 To analyze the second phase of KarpSipserMT, we will use the following lemma.

Lemma 4. *Let $G' = (V'_R \cup V'_C, E')$ be the graph induced by the remaining vertices after the first phase of* KarpSipserMT*. Then, the set*

$$\{(u, \texttt{choice}[u]) : u \in V'_R, \texttt{choice}[u] \in V'_C\}$$

is a maximum cardinality matching in G'.

Proof. Apart from 2-cliques, no out-one or in-one (that is no degree-one) vertex remains after Phase 1. A component of G' can be a trivial (a singleton vertex),
340 a 2-clique, or a simple cycle, according to Lemma 2. Let P be a non-trivial component. Since the original graph is bipartite, if P is a cycle it has the edges $(u, \texttt{choice}[u])$ and $(\texttt{choice}[v], v)$ for $u \in V'_R \cap P$ and $v \in V'_C \cap P$. The edge set $\{(u, \texttt{choice}[u]) : u \in V'_R \cap P, \texttt{choice}[u] \in V'_C\}$ defines a maximum cardinality matching for P. The union of these edge sets matches all the vertices except
345 those in the trivial components, hence it is a maximum matching in G'. □

In the light of Observations 1 and 2 and Lemmas 2–4, KarpSipserMT is an exact algorithm on the graphs created in Algorithm 3. The worst case (sequential) run time of our implementation of KS is linear.

KarpSipserMT tracks and consumes only the out-one vertices. This brings
350 high flexibility while executing KarpSipserMT in multi-threaded environments. Consider the example in Figure 1. Here, after matching a pair of vertices and removing them from the graph, multiple degree-one vertices can be generated. The standard KS uses a list to store these new degree-one vertices. Such a list is necessary to obtain larger matching, but the associated synchronizations
355 while updating it in parallel will be an obstacle for efficiency. The synchronization can be avoided up to some level if one sacrifices the approximation quality by not making all optimal decisions (as in some existing work [9]). We continue with the following lemma to take advantage of the special structure of the graphs in TwoSidedMatch for parallel efficiency in Phase 1.

360 **Lemma 5.** *Consuming an out-one vertex creates at most one new out-one vertex.*

Proof. Let u be the out-one vertex that is selected by KarpSipserMT, and let v be $\texttt{choice}[u]$. Since u is a degree-one vertex, its removal will only affect v. On the other hand, although a number of in-one vertices may appear, v's removal
365 can only make the vertex $w = \texttt{choice}[v]$ out-one. This happens iff w is still unmatched, and there is no other unmatched vertex y with $\texttt{choice}[y] = w$. □

According to Lemma 5, KarpSipserMT does not need a list to store the new out-one vertices, since the process can continue with the new out-one vertex. In a shared-memory setting, there are two concerns for the first phase from
370 the synchronization point of view. First, multiple threads that are consuming different out-one vertices can try to match them with the same unmatched vertex. To handle such cases, KarpSipserMT uses the atomic _CompAndSwap operation (line 15 of Algorithm 4) and ensures that only one of these matchings will be processed. In this case, other threads, whose matching decisions are

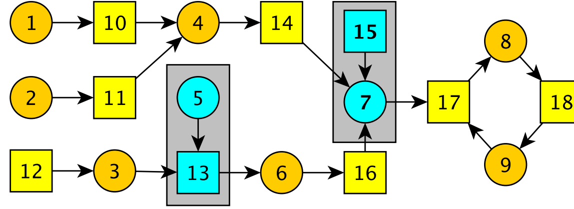


Figure 1: A toy bipartite graph with 9 row (circles) and 9 column (squares) vertices. The edges are oriented from a vertex u to the vertex choice$[u]$. Assuming all the vertices are currently unmatched, matching 15-7 (or 5-13) creates two degree-one vertices. But no out-one vertex arises after matching (15-7) and only one, vertex 6, arises after matching (5-13).

375 not performed, continue with the next vertex in the **for** loop at line 10. The second concern is that while consuming out-one vertices, several threads may create the same out-one vertex (and want to continue with it). For example, in Figure 1, when two threads consume the out-one vertices 1 and 2 at the same time, they both will try to continue with vertex 4. To handle such cases, an
380 atomic _AddAndFetch operation (line 20 of Algorithm 4) is used to synchronize the degree reduction operations on the potential out-one vertices. This approach explicitly orders the vertex consumptions and guarantees that only the thread who performs the last consumption before a new out-one vertex u appears continues with u. The other threads which wanted to continue with the
385 same path stop and skip to the next unconsumed out-one vertex in the main **for** loop. One last concern that can be important on the parallel performance is the maximum length of such paths since a very long path can yield a significant imbalance on the work distribution to the threads. the load imbalance that may be created by the length of such paths. We investigated the length of such paths
390 experimentally (see Section 4.2) and observed them to be too short to hurt the parallel performance.

The second phase of KarpSipserMT is efficiently parallelized by using the idea in Lemma 4. That is, a maximum cardinality matching for the graph remaining after the first phase of KarpSipserMT can be obtained via a simple
395 parallel **for** construct (see line 24 of Algorithm 4).

Quality of approximation

If the initial matrix $\mathbf{A}$ is the $n \times n$ matrix of 1s; that is $a_{ij} = 1$ for all $1 \leq i, j \leq n$, then the doubly stochastic matrix $\mathbf{S}$ is such that $s_{ij} = \frac{1}{n}$ for all $1 \leq i, j \leq n$. In this case, the graph G created by Algorithm 3 is a random 1-
400 out bipartite graph [29]. Referring to a study by Meir and Moon [30], Karoński and Pittel [31] argue that the maximum cardinality of a matching in a random 1-out bipartite graph is $2(1 - \Omega)n \approx 0.866n$ in expectation where $\Omega \approx 0.567$, also called Lambert's $W(1)$, is the unique solution of the equation $\Omega e^{\Omega} = 1$. In the remaining of this subsection, we will show that the same result holds for
405 any square matrix $\mathbf{A}$ with total support. We state this as a theorem.

Theorem 2. *Let* $\mathbf{A}$ *be an* $n \times n$ *matrix with total support. Then,* TwoSidedMatch *obtains a matching of size* $2(1-\Omega)n \approx 0.866n$ *in expectation as* $n \to \infty$*, where* $\Omega \approx 0.567$ *is the unique solution of the equation* $\Omega e^{\Omega} = 1$*.*

Theorem 2 contributes to the known results about the Karp-Sipser heuristic (recall that it is known to leave out $\tilde{O}(n^{1/5})$ vertices) by showing a constant approximation ratio with some preprocessing. The existence of total support does not seem to be necessary for the Theorem 2 to hold (see the next subsection).

We now set the scene for the proof of Theorem 2.

Let $G = (A, B, E')$ be the graph constructed at line 8 of Algorithm 3. In the following, let $f(a) \in B$ be the vertex randomly selected by an $a \in A$ in the algorithm. Similarly, let $g(b)$ denotes the vertex in A selected by $b \in B$. We will extend the f and g functions for sets of vertices as follows: for a set $S \subseteq A$, $f(S) = \bigcup_{a \in S}\{f(a)\}$; for a set $S' \subseteq B$, $g(S') = \bigcup_{b \in S'}\{g(b)\}$. We will denote the vertex set selecting a specific vertex $a \in A$ by $g^{-1}(a) = \{b \in B : g(b) = a\}$, and the vertex set selecting a specific vertex $b \in B$ by $f^{-1}(b) = \{a \in A : f(a) = b\}$.

Our objective is to have an upper bound on the A vertices that will not be matched by a Karp-Sipser (KS) execution on G. Consider the vertex set $A_1 \subseteq A$ that are not picked by any node $b \in B$. Since an $a \in A_1$ is a degree-1 vertex of G, if a is a matched vertex it is matched with its only neighbor $f(a) \in B$. Let $B_1 \subseteq B$ be $f(A_1)$. If $A_1 = \emptyset$ all the vertices of A belong to a cycle, and hence, to a perfect matching. Otherwise, $|A_1| - |B_1|$ vertices in A_1 cannot be matched by KS and matching the B_1 vertices with A_1 vertices will probably create new degree-1 vertices in $G' = (A \setminus A_1, B \setminus B_1, E' \cap (A \setminus A_1) \times (B \setminus B_1))$.

Starting with A_1, let B_k recursively be the set of B vertices picked by at least one vertex of A_k, that is $B_k = f(A_k)$, and let A_k be the set of vertices not picked by any $B \setminus B_{k-1}$ vertex, that is $A_k = A \setminus g(B \setminus B_{k-1})$. Note that $A_{k-1} \subseteq A_k$ for all $k > 1$. Similarly, $B_{k-1} \subseteq B_k$ and B_k contains all B vertices that are perfectly matchable to the vertices in A_k. Here, $|A_k|$, $|B_k|$ and $|A_k| - |B_k|$ are increasing sequences of k. The analysis simulates a batched KS variant where at each step k, all $B_k \setminus B_{k-1}$ vertices are matched to the $A_k \setminus A_{k-1}$ vertices at once leaving some of $A_k \setminus A_{k-1}$ vertices forever unmatched, i.e., isolated, and creating new degree-1 A vertices (which appear in A_{k+1}). A_n does not contain all the A vertices and $A \setminus A_n$ is perfectly matchable to $B \setminus B_n$. Therefore, the number of A vertices that will remain unmatched is $|A_n| - |B_n|$ and the cardinality of the matching found by KS will be $|A| - |A_n| + |B_n|$.

For each batched KS execution, there is at least one non-batched KS execution which obtains the matchings in the batches in some specific order and reaches the maximum cardinality. Hence, as all the algorithms that prioritize degree-1 vertices, the batched KS algorithm analyzed here obtains the maximum cardinality matching.

Let $\alpha_i^{(k)}$ denote the probability $P(a_i \in A_k)$ that the vertex a_i belongs to A_k for $i, k \in \{1, 2, \ldots, n\}$. Similarly, let $\beta_j^{(k)}$ denote the probability $P(b_j \in B_k)$ that b_j belongs to B_k for $j, k \in \{1, 2, \ldots, n\}$. Note that $\sum_i \alpha_i^{(n)} - \sum_j \beta_j^{(n)} = E(|A_n| - |B_n|)$.

450 Theorem 2 will be proved by bounding the deficiency of the matching by upper bounding $|A_n| - |B_n|$.

Lemma 6. *When the recursive process above is executed based on the values in the scaled matrix* $\mathbf{S}$*, the vertex* $b_j \in B$ *appears in* B_k *with probability*

$$\beta_j^{(k)} \geq 1 - e^{-\left(\sum_{i=1}^{n} s_{ij}\cdot\alpha_i^{(k)}\right)}.$$

Proof. We write $\beta_j^{(k)} = P(b_j \in B_k)$ as

$$1 - \sum_{A' \subseteq A} P(f^{-1}(b_j) = A') \cdot P\left(\bigcap_{a_i \in A'} a_i \notin A_k\right) \tag{1}$$

where the first term of the summation is

$$P(f^{-1}(b_j) = A') = \left(\prod_{a_i \in A'} s_{ij}\right) \cdot \left(\prod_{a_i \notin A'} (1 - s_{ij})\right) . \tag{2}$$

For the second term of the summation, let E_i be the event that "$a_i \notin A_k$", that is a_i is selected by a vertex in $B \setminus B_{k-1}$. We first prove by induction on $|A'|$ that $P(\bigcap_{a_i \in A'} E_i) \leq \prod_{a_i \in A'} P(E_i)$. For $|A'| = 1$, the (in)equality is evident; assume that it is correct also for all A' with $|A'| < \ell$. Let A' be a subset of A of size ℓ having the vertex a_u. Let $A'' = A'/a_u$. By Bayes law and the induction assumption, we have:

$$P\left(\bigcap_{a_i \in A'} E_i\right) = P\left(\bigcap_{a_i \in A''} E_i\right) \cdot P\left(E_u | \bigcap_{a_i \in A''} E_i\right) \leq \prod_{a_i \in A''} P(E_i) \cdot P\left(E_u | \bigcap_{a_i \in A''} E_i\right).$$

We now prove that $P\left(E_u | \bigcap_{a_i \in A''} E_i\right) \leq P(E_u)$. Intuitively, the A vertices will be selected by the previously decided $B \setminus B_{k-1}$ vertices and a condition that restricts the event E_u to the selection of some other A vertices simply reduces the probability of E_u. That is, if all the vertices in A'' are already selected by the vertices in $B \setminus B_{k-1}$, then fewer selection options and less chances remain for a_u to be selected. More formally, $P\left(E_u | \bigcap_{a_i \in A''} E_i\right)$ can be written as

$$\begin{aligned}
& \sum_{B' \subseteq B/B_{k-1}} P(E_u | (\bigcap_{a_i \in A''} E_i) \cap (B' = g^{-1}(A''))) \cdot P(B' = g^{-1}(A'')) \\
= & \sum_{B' \subseteq B/B_{k-1}} P(a_u \in g(B \setminus (B' \cup B_{k-1}))) \cdot P(B' = g^{-1}(A'')) \\
\leq & \sum_{B' \subseteq B/B_{k-1}} P(a_u \in g(B \setminus B_{k-1})) \cdot P(B' = g^{-1}(A'')) \\
\leq & \;\; P(E_u).
\end{aligned}$$

This implies that for all $A' \subseteq A$, we have $P(\bigcap_{a_i \in A'} E_i) \leq \prod_{a_i \in A'} P(E_i)$ and therefore,

$$P\left(\bigcap_{a_i \in A'} a_i \notin A_k\right) \leq \prod_{a_i \in A'} P(a_i \notin A_k) \ . \tag{3}$$

Based on these observations, $\beta_j^{(k)} = P(b_j \in B_k)$ can be bounded as follows:

$$\beta_j^{(k)} = 1 - \sum_{A' \subseteq A} P(f^{-1}(b_j) = A') \cdot P\left(\bigcap_{a_i \in A'} a_i \notin A_k\right)$$

by (2) and (3), we have

$$\geq 1 - \sum_{A' \subseteq A} \left(\prod_{a_i \in A'} s_{ij}\right) \cdot \left(\prod_{a_i \notin A'} (1 - s_{ij})\right) \cdot \left(\prod_{a_i \in A'} P(a_i \notin A_k)\right)$$

then by Lemma 1

$$= 1 - \prod_{a_i \in A} (1 - s_{ij} + s_{ij} \cdot P(a_i \notin A_k))$$

$$= 1 - \prod_{a_i \in A} (1 - s_{ij} \cdot P(a_i \in A_k))$$

$$= 1 - \prod_{a_i \in A} \left(1 - s_{ij} \cdot \alpha_i^{(k)}\right)$$

and by the arithmetic geometric inequality,

$$\geq 1 - \left(1 - \frac{\left(\sum_{i \in A} s_{ij} \cdot \alpha_i^{(k)}\right)}{n}\right)^n$$

$$\geq 1 - e^{-\left(\sum_{i \in A} s_{ij} \cdot \alpha_i^{(k)}\right)} \ .$$

□

Lemma 7. *When the recursive process above is executed based on the values in the scaled matrix* $\mathbf{S}$*, the vertex* $a_i \in A$ *appears in* A_k *with probability*

$$\alpha_i^{(k)} \leq e^{-\left(1 - \sum_{j=1}^{n} s_{ij} \cdot \beta_j^{(k-1)}\right)}$$

Proof. The proof will follow the previous one. We write $\alpha_i^{(k)}$, the probability that a_i belongs to A_k, as

$$\sum_{B' \subseteq B} P(g^{-1}(a_i) = B') \cdot P\left(\bigcap_{b_j \in B'} b_j \in B_{k-1}\right)$$

where the first term of the summation is

$$P(g^{-1}(a_i) = B') = \left(\prod_{b_j \in B'} s_{ij} \right) \cdot \left(\prod_{b_j \notin B'} 1 - s_{ij} \right) . \tag{4}$$

We will obtain an upper bound on the second term of the summation to upper bound $\alpha_i^{(k)}$. Let F_j be the event "$b_j \in B_{k-1}$", that is, b_j is selected by a vertex in A_{k-1}. We first prove by induction on $|B'|$ that $P(\bigcap_{b_j \in B'} F_j) \leq \prod_{b_j \in B'} P(F_j)$. For $|B'| = 1$, the (in)equality is evident; assume that it holds also for all B' with $|B'| < \ell$. Let B' be a subset of B of size ℓ having the vertex b_u. Let $B'' = B'/b_u$. By Bayes law and the induction assumption, we have:

$$P\left(\bigcap_{b_j \in B'} F_j \right) = P\left(\bigcap_{b_j \in B''} F_j \right) \cdot P\left(F_u | \bigcap_{j \in B''} F_j \right) \leq \prod_{b_j \in B''} P(F_j) \cdot P\left(F_u | \bigcap_{j \in B''} F_j \right) .$$

We now prove that $P(F_u | \bigcap_{j \in B''} F_j) \leq P(F_u)$. Intuitively, the B_k vertices will be selected by the previously decided A_k vertices and a condition that restricts the event F_u to the selection of some other B_k vertices simply reduces the probability of F_u. More formally, $P(F_u | \bigcap_{j \in B''} F_j)$ can be written as

$$\begin{aligned}
& \sum_{A' \subseteq A_k} P(F_u | (\bigcap_{b_j \in B''} F_j) \cap (A' = f^{-1}(B'') \cap A_k)) \cdot P(A' = f^{-1}(B'') \cap A_k) \\
\leq & \sum_{A' \subseteq A_k} P(b_u \in f(A_k/A')) \cdot P(A' = f^{-1}(B'') \cap A_k) \\
\leq & \sum_{A' \subseteq A_k} P(b_u \in f(A_k)) \cdot P(A' = f^{-1}(B'') \cap A_k) \\
\leq & \ P(F_u) .
\end{aligned}$$

This implies that for all $B' \subseteq B$, we have $P(\bigcap_{b_j \in B'} F_j) \leq \prod_{b_j \in B'} P(F_j)$. Therefore,

$$P\left(\bigcap_{b_j \in B'} b_j \in B_{k-1} \right) \leq \prod_{b_j \in B'} P(b_j \in B_{k-1}). \tag{5}$$

Based on these observations, $\alpha_i^{(k)} = P(a_i \in A_k)$ can be upper bounded as

follows:

$$\alpha_i^{(k)} = \sum_{B' \subseteq B} P(g^{-1}(a_i) = B') \cdot P(B' \subseteq B_{k-1})$$

by (4) and (5), we have

$$\leq \sum_{B' \subseteq B} \left(\prod_{b_j \in B'} s_{ij} \right) \cdot \left(\prod_{b_j \notin B'} (1 - s_{ij}) \right) \cdot \left(\prod_{b_j \in B'} P(b_j \in B_{k-1}) \right)$$

then by Lemma 1

$$= \prod_{j \in B} (1 - s_{ij} + s_{ij} \cdot P(b_j \in B_{k-1})) \ ,$$

and by the arithmetic geometric inequality

$$\leq \left(1 - \frac{\left(\sum_{b_j \in B} s_{ij} \cdot (1 - \beta_j^{(k)}) \right)}{n} \right)^n$$

$$\leq e^{-\left(1 - \sum_{b_j \in B} s_{ij} \cdot \beta_j^{(k)}\right)} \ .$$

□

Now consider the sequences $\alpha_i^{(k)}$ and $\beta_j^{(k)}$ for fixed i and j; since these sequences are increasing with k and the terms are always smaller than 1, they converge to some value. Let us denote these limits by α_i and β_j, respectively. Let α and β be the vectors of α_is and β_js, i.e.,

$$\begin{aligned} \alpha = & \quad (\alpha_1, \alpha_2, \dots, \alpha_n) \quad \in [0,1]^n \\ \beta = & \quad (\beta_1, \beta_2, \dots, \beta_n) \quad \in [0,1]^n \end{aligned}$$

where by Theorems 6 and 7

$$\begin{aligned} \alpha_i \leq & \quad e^{-\left(1 - \sum_{1 \leq j \leq n} s_{ij} \cdot \beta_j\right)} && \text{for all } 1 \leq i \leq n \ , \\ \beta_j \geq & \quad 1 - e^{-\sum_{1 \leq i \leq n} s_{ij} \cdot \alpha_i} && \text{for all } 1 \leq j \leq n \ . \end{aligned}$$

We now prove that for $2n$ values respecting these properties, $\sum_i \alpha_i - \sum_j \beta_j \leq n \cdot (2\Omega - 1)$, where Ω equals to $W(1)$ of Lambert's W function ($\Omega = e^{-\Omega}$). Note that $2\Omega - 1 \approx 0.134$, and therefore the expected value of the matching is about $0.866 \cdot n$.

Lemma 8. *Let $X = (x_i)_{1 \leq i \leq n} \in [0,1]^n$, $Y = (y_j)_{1 \leq j \leq n} \in [0,1]^n$ such that:*

$$x_i \leq e^{-\left(1 - \sum_{1 \leq j \leq n} s_{ij} \cdot y_j\right)}. \tag{6}$$

$$y_j \geq 1 - e^{-\sum_{1 \leq i \leq n} s_{ij} \cdot x_i} \tag{7}$$

The maximum value of $\sum_i x_i - \sum_j y_j$ is $n \cdot (2\Omega - 1)$ which is attained when all $x_i = \Omega$ and all $y_j = 1 - \Omega$, where Ω is the unique solution of $\Omega e^{\Omega} = 1$.

465 *Proof.* Let (X_Ω, Y_Ω) denote the solution where $x_i = \Omega$ and $y_j = 1 - \Omega$ for all i and j. The solution (X_Ω, Y_Ω) satisfies (6) and (7) and attains the value $n \cdot (2\Omega - 1)$. We now show that there cannot be a solution with a value larger than $n \cdot (2\Omega - 1)$.

We are going to take a solution $(X, Y) \neq (X_\Omega, Y_\Omega)$ and investigate three
470 cases: (i) $0 < x_i, y_j < 1$ for all i and j, and $\sum x_i - \sum y_j > n \cdot (2\Omega - 1)$; (ii) there is an $x_i = 0$ or $y_j = 0$; (iii) there is an $x_i = 1$ or $y_j = 1$. In all three cases, we are going to show that (X, Y) cannot be optimal. In the remaining case, where $0 < x_i, y_j < 1$ for all i and j, and $\sum x_i - \sum y_j \leq n \cdot (2\Omega - 1)$, the solution is no better than (X_Ω, Y_Ω).

475 **Case (i):** Here $0 < x_i, y_j < 1$ for all i and j, and $\sum x_i - \sum y_j > n \cdot (2\Omega - 1)$. In this case, we show that (X, Y) cannot be optimal, thus achieving a contradiction.

Let (X', Y') be defined as follows: $x'_i = \frac{x_i - (1-p) \cdot \Omega}{p}$ and $y'_j = \frac{y_j - (1-p) \cdot (1-\Omega)}{p}$ for any $p \in \mathbb{R}$ with $0 < p < 1$. In other words,

$$x_i = p \cdot x'_i + (1 - p) \cdot \Omega \tag{8}$$

$$y_j = p \cdot y'_j + (1 - p) \cdot (1 - \Omega)\ . \tag{9}$$

Then from (6),

$$\begin{aligned}
x_i &\leq e^{-\left(1 - \sum_{1 \leq j \leq n} s_{ij} \cdot y_j\right)} \\
p \cdot x'_i + (1 - p) \cdot \Omega &\leq e^{-\left(1 - \sum_{1 \leq j \leq n} s_{ij} \cdot \left(p \cdot y'_j + (1-p) \cdot (1 - \Omega)\right)\right)} \\
&\quad \text{and by the convexity of the function } x \to e^{-(1-x)}, \\
&\leq p \cdot e^{-\left(1 - \sum_{1 \leq j \leq n} s_{ij} \cdot y'_j\right)} + (1 - p) \cdot \Omega \\
x'_i &\leq e^{-\left(1 - \sum_{1 \leq j \leq n} s_{ij} \cdot y'_j\right)}\ .
\end{aligned}$$

The same way, from (7),

$$\begin{aligned}
y_j &\geq 1 - e^{-\sum_{1 \leq i \leq n} s_{ij} \cdot x_i} \\
p \cdot y'_j + (1 - p) \cdot (1 - \Omega) &\geq 1 - e^{-\sum_{1 \leq i \leq n} s_{ij} \cdot \left(p \cdot x'_i + (1-p) \cdot \Omega\right)} \\
&\quad \text{and by the concavity of the function } x \to 1 - e^{-x}, \\
&\geq p \cdot \left(1 - e^{-\sum_{1 \leq i \leq n} s_{ij} \cdot x'_i}\right) + (1 - p) \cdot (1 - \Omega) \\
y'_j &\geq 1 - e^{-\sum_{1 \leq i \leq n} s_{ij} \cdot x'_i}\ .
\end{aligned}$$

Thus, (X', Y') satisfies (6) and (7).

We now define a p with the property that $0 < p < 1$ which leads to
480 $0 \leq x'_i, y'_j \leq 1$. Let $m_x = \min_i\{x_i\}$, $m_y = \min_j\{y_j\}$, $M_x = \max_i\{x_i\}$ and $M_y = \max_j\{y_j\}$. Let $p = \max\{\frac{M_x - \Omega}{1 - \Omega}, \frac{\Omega - m_x}{\Omega}, \frac{M_y - 1 + \Omega}{\Omega}, \frac{1 - \Omega - m_y}{1 - \Omega}\}$. Since, $0 < x_i, y_j < 1$, we have $0 < p < 1$. We note that by definition $p \cdot (x'_i - \Omega) = x_i - \Omega$ and $p \cdot (y'_j - 1 + \Omega) = y_j - 1 + \Omega$. We now analyze x'_i. If $x'_i \geq \Omega$,

$$\begin{aligned} p \cdot (x_i' - \Omega) &= x_i - \Omega \\ (x_i' - \Omega) &= \tfrac{x_i - \Omega}{p} \\ x_i' - \Omega &\le \tfrac{1-\Omega}{M_x - \Omega} \cdot (x_i - \Omega) \\ &\le 1 - \Omega \\ x_i' &\le 1 \end{aligned}$$

and the same way, if $x_i' \le \Omega$,

$$\begin{aligned} p \cdot (x_i' - \Omega) &= x_i - \Omega \\ x_i' - \Omega &\ge \tfrac{\Omega}{\Omega - m_x} \cdot (x_i - \Omega) \\ &\ge -\Omega \\ x_i' &\ge 0 \ . \end{aligned}$$

485 Hence, $0 \le x_i' \le 1$. The same can be shown for y_j' by using the other two inequalities for p. Thus, (X', Y') is a feasible solution.

We now show that $\sum_i x_i' - \sum_j y_j' > \sum_i x_i - \sum_j y_j$, contradicting the optimality of (X, Y). By Equations (8) and (9), we have

$$\sum (p \cdot x_i' + (1-p) \cdot \Omega) = \sum x_i \text{ and} \sum \left(y_j' + (1-p) \cdot (1-\Omega) \right) = \sum y_j \ .$$

Therefore,

$$p \cdot \left(\sum x_i' - \sum y_j' - n \cdot (2\Omega - 1) \right) = \sum x_i - \sum y_j - n \cdot (2\Omega - 1) > 0 \ . \quad (10)$$

Hence,

$$\begin{aligned} \textstyle\sum_i x_i' - \sum_j y_j' &= \textstyle\sum x_i' - \sum y_j' - n \cdot (2\Omega - 1) + n \cdot (2\Omega - 1) \\ &> p \cdot \left(\textstyle\sum x_i' - \sum y_j' - n \cdot (2\Omega - 1) \right) + n \cdot (2\Omega - 1) \\ &\quad \text{by using Equation (10) and its analogue for } y_j' \\ &= \textstyle\sum x_i - \sum y_j \end{aligned}$$

Case (ii): Here $x_i = 0$ or $y_j = 0$ for some i and j. We will show that in this case, unless all x_i and y_j are zero, we can obtain (X', Y') such that $\sum x_i' - \sum y_j' > \sum x_i - \sum y_j$, and hence (X, Y) cannot be optimal. But if
490 all $x_i = 0$ and $y_j = 0$, then we have $\sum x_i - \sum y_j = 0 < n \cdot (2\Omega - 1)$, and hence (X, Y) is not optimal.

Suppose that $y_j = 0$. Then, from Equation (7), we have $x_j = 0$ for all $s_{ij} > 0$.

Suppose that $x_i = 0$. Let $x_i'' = e^{-1}$, and $x_k'' = x_k$ for all $k \ne i$ and let $y_j'' = 1 - (1 - y_j) \cdot (1 - s_{ij} \cdot e^{-1})$ for all j. Note that $x_i'' = e^{-1} \le e^{-(1 - \sum_i s_{ij} \cdot y_j'')}$ and $y_j'' \ge y_j$. Then,

$$\begin{aligned} y_j'' &= 1 - (1 - y_j) \cdot (1 - s_{ij} \cdot e^{-1}) \\ &\ge 1 - e^{-\sum_i s_{ij} \cdot x_i} (1 - s_{ij} \cdot e^{-1}) \\ &\ge 1 - e^{-\sum_i s_{ij} \cdot x_i} e^{-s_{ij} \cdot e^{-1}} \\ &\ge 1 - e^{-\sum_i s_{ij} \cdot x_i''} \end{aligned}$$

and for $k \neq i$,

$$\begin{aligned} x''_k &= x_k \\ &\leq e^{-(1-\sum_j s_{kj}\cdot y_j)} \\ &\leq e^{-(1-\sum_j s_{kj}\cdot y''_j)} . \end{aligned}$$

Then, (X'', Y'') is a solution, and

$$\sum_i x''_i - y''_i = \sum_i x_i - y_i + e^{-1}\left(1 - \sum_j s_{ij}\cdot(1-y_j)\right).$$

495 If (X, Y) were optimum, then $\sum_j s_{ij} \cdot y_j = 0$, and for all j with $s_{ij} > 0$, we have $y_j = 0$.

Since the graph of $\mathbf{A}$ has the strong Hall property, for any proper subset $V \subsetneq A$ of vertices, we have $|V| < |adj(V)|$, and the same condition holds for the proper subsets of B vertices. Therefore, the two properties just
500 shown imply that all $x_i = 0$ and all $y_i = 0$.

Case (iii): Here $x_i = 1$ or $y_j = 1$ for some i and j. Let (X', Y') be such that $x'_i = 1 - y_i$ and $y'_j = 1 - x_j$. Then (X', Y') is feasible and have the same value. This is because, $0 \leq x'_i, y'_j \leq 1$ for all i and j, and (6) and (7) are exchanged, if we replace x_i by $1 - y_i$ and y_i by $1 - x_i$. Since X or
505 Y contains a one, X' or Y' contains a zero. Therefore, we can use Case (ii) above for (X', Y') and show that if (X', Y') were optimal, then $x_i = 1$ and $y_j = 1$ for all i and j. Therefore, $\sum x_i - \sum y_j = 0 < n \cdot (2\Omega - 1)$.

By the cases above, (X_Ω, Y_Ω) is optimal and hence $n \cdot (2\Omega - 1)$ is the maximum value. □

510 We are now ready to give the proof of Theorem 2.

Proof of Theorem 2. If the original matrix $\mathbf{A}$ is fully indecomposable, then we have the desired result by Lemma 8. If the matrix has total support, but decomposable (that is $\mathbf{A}$ is block diagonal with square diagonal blocks being fully indecomposable), then the analysis in Lemma 8 shows that at each diagonal
515 block $2(1-\Omega)$ approximation is achieved, thus yielding the same approximation ratio for $\mathbf{A}$. □

3.3. Further discussions

The Sinkhorn-Knopp scaling algorithm converges linearly (when $\mathbf{A}$ has total support) where the convergence rate is equivalent to the square of the second
520 largest singular value of the resulting, doubly stochastic matrix [25]. If the matrix does not have (total) support, less is known about the Sinkhorn-Knopp scaling algorithm, in which case, we are not able to bound the run time of the scaling step. However, the scaling algorithms should be run only a few iterations

(see also below) in practice, in which case the practical run time of our heuristics would be linear (in edges and vertices).

We have discussed the proposed matching heuristics while assuming that $\mathbf{A}$ has total support. This can be relaxed in two ways to render the overall approach practical in any bipartite graph. The first relaxation is that we do not need to run the scaling algorithms until convergence (as in some other uses of similar algorithms [26]). If $\sum_{i\in\mathbf{A}_{*j}} s_{ij} \geq \alpha$ instead of $\sum_{i\in\mathbf{A}_{*j}} s_{ij} = 1$ for all $j \in \{1,\ldots,n\}$ then, $\lim_{d\to\infty}\left(1-\frac{\alpha}{d}\right)^d = \frac{1}{e^{\alpha}}$. In other words, if we apply the scaling algorithms a few iterations, or until some relatively large error tolerance, we can still derive similar results. For example, if $\alpha = 0.92$, we will have a matching of size $n\left(1-\frac{1}{e^{\alpha}}\right) \approx 0.601n$ (larger column sums give improved ratios; but there are columns whose sum is less than one, when the convergence is not achieved) for OneSidedMatch. With the same α, TwoSidedMatch will obtain an approximation of 0.840 (obtained by plugging in $\sum s_{ij} = \alpha$ in Equation (7); again larger column sums give improved ratios). In our experiments, the number of iterations were always a few, where the proven approximation guarantees were always observed. The second relaxation is that we do not need total support; we do not even need support nor equal number of vertices in the two vertex classes. We note that most theoretical studies on randomized matching heuristics concentrate on graphs with perfect matching, as this is enough to present approximation guarantees [15, Section 2]. Since little is known about the scaling methods on such matrices, we do not dwell into the subject (scaling algorithms are not our focus), but we mention some facts and observations, and later on, present some experiments to demonstrate the practicality of the proposed OneSidedMatch and TwoSidedMatch heuristics.

A sparse matrix (not necessarily square) can be permuted into a block upper triangular form using the canonical Dulmage-Mendelsohn (DM) decomposition [32]

$$\mathbf{A} = \begin{pmatrix} H & * & * \\ O & S & * \\ O & O & V \end{pmatrix}, \quad S = \begin{pmatrix} S_1 & * \\ O & S_2 \end{pmatrix}$$

where, H (horizontal) has more columns than rows and has a matching covering all rows; S is square and has a perfect matching; and V (vertical) has more rows than columns and a matching covering all columns. The following facts about the DM decomposition are well known [33, 13]. Any of these three blocks can be void. If H is not connected, then it is block diagonal with horizontal blocks. If V is not connected, then it is block diagonal with vertical blocks. If S does not have total support, then it is in block upper triangular form, shown on the right, where S_1 and S_2 have the same structure recursively, until each block S_i is has total support. The entries in the blocks shown by "*" cannot be put into a maximum cardinality matching. When the presented scaling methods are applied to a matrix, the entries in "*" blocks will tend to zero (the case of S is well documented [24]). Furthermore, the row sums of the blocks of H will be a multiple of the column sums in the same block; a similar statement holds for V; finally S will be doubly stochastic. That is, the scaling algorithms

applied to bipartite graphs without perfect matchings will zero out the entries in the irrelevant parts and identify the entries that can be put into a maximum cardinality matching.

4. Experiments

We conduct experiments for two purposes. The first purpose is to observe the theoretical findings in practice. To do so, we first run the proposed heuristics on a set of matrices arising in various applications having support and measure the quality of the results. We then run the heuristics on sparse random matrices to compare the quality of TwoSidedMatch with the original Karp-Sipser. This will show if the approximation ratio of TwoSidedMatch is any better than simply running KS. We also conduct experiments with matrices, both square and rectangular, that do not have total support to highlight the cases that are not fully supported by our theoretical analysis. The second purpose of experiments is to investigate the efficiency of our parallel implementations of the proposed heuristics.

The experiments were carried out on a machine equipped with two Intel Sandybridge-EP CPUs clocked at 2.00Ghz and 256GB of memory split across the two NUMA domains. Each CPU has eight-cores (16 cores in total) and HyperThreading is enabled. Each core has its own 32kB L1 cache and 256kB L2 cache. The 8 cores on a CPU share a 20MB L3 cache. The machine runs 64-bit Debian with Linux 2.6.39-bpo.2-amd64. All the codes are compiled with `gcc 4.4.5` with the `-O2` optimization flag. All algorithms are implemented using C and OpenMP parallelism. The (`dynamic,512`) OpenMP scheduling policy is employed while running all the algorithms except KarpSipserMT for which we used (`guided`). The `dynamic` policy is chosen to avoid a possible load imbalance due to a non-uniform degree distribution in the bipartite graph. On the other hand, the `guided` policy is a better fit for KarpSipserMT since the number of degree-one/out-one vertices tends to decrease throughout the execution. These claims are also verified by a set of preliminary experiments. Parallel runs are performed with 2, 4, 8, 16 threads.

In our first heuristic OneSidedMatch, the match array is constructed by assuming a "last-store wins" policy which guarantees that the last write operation to the same memory location survives. In our implementation, we used: (i) 32-bit unsigned integers for vertex IDs which are sufficient to handle graphs with 4 billion vertices in practice; and (ii) the `malloc` function for memory allocation which returns a suitably aligned pointer for any built-in type. Our architectural setting guarantees that when the memory is aligned, all the 32-bit store and load operations are atomic. Such a setting is very common in practice. However, for architectures which do not have such a guarantee, implementations and techniques to force atomic stores and loads might be required.

For atomic operations required by KarpSipserMT, `gcc 4.5.5`'s built-in functions, which support any integral scalar up to 8 bytes in length, are used. Specifically, we used `__sync_fetch_and_add`, `__sync_bool_compare_and_swap`, and `__sync_sub_and_fetch` operations. These operations are currently being

supported efficiently by many multicore architectures from Intel and AMD. Hence, the algorithms are expected to perform reasonably well on modern multicore machines as we will show in this section.

610 We use a fixed number of scaling iterations. This way, the scaling algorithm is simplified, as no convergence check is required. Furthermore, its overhead is bounded as well. In most of the experiments, we use $0, 1, 5$, or 10 scaling iterations, where the case of 0 corresponds to applying the matching heuristics with uniform edge selection probabilities.

615 *4.1. Experimental verification of theoretical results*

4.1.1. Matching quality

We investigate the matching quality of the proposed heuristics on all square matrices with support from the UFL Sparse Matrix Collection [34] having at least 1000 non-empty rows/columns and at most 20000000 nonzeros (some of 620 the matrices are also in the 10th DIMACS challenge [35]). There were 742 matrices satisfying these properties at the time of experimentation. With the OneSidedMatch heuristic, the quality guarantee of 0.632 was surpassed with 10 iterations of the scaling method in 729 matrices. With the TwoSidedMatch heuristic and the same number of iterations of the scaling methods, the quality 625 guarantee of 0.866 was surpassed in 705 matrices. Making 10 more scaling iterations smoothed out the remaining instances. Note that the cheap matching heuristic (the second variant discussed in Section 2.1) obtained, on average (both the geometric and arithmetic means), matchings of size 0.93 of the maximum cardinality. In the worst case, 0.82 was observed.

630 *4.1.2. Comparison of* TwoSidedMatch *with* KarpSipser

Next, we analyze the performance of the proposed heuristics with respect to KS on a matrix class which we designed as a bad case for KS. Let $\mathbf{A}$ be an $n \times n$ matrix, R_1 be the set of $\mathbf{A}$'s first $n/2$ rows, and R_2 be the set of $\mathbf{A}$'s last $n/2$ rows; similarly let C_1 and C_2 be the set of first $n/2$ and the set of last 635 $n/2$ columns. As Figure 2 shows, $\mathbf{A}$ has a full $R_1 \times C_1$ block and an empty $R_2 \times C_2$ block. The last $k \ll n$ rows and columns of R_1 and C_1, respectively, are full. Each of the blocks $R_1 \times C_2$ and $R_2 \times C_1$ has a nonzero diagonal. Those diagonals form a perfect matching when combined. In the sequel, a matrix whose corresponding bipartite graph has a perfect matching will be called *full-sprank*, 640 and *sprank-deficient* otherwise.

When $k \leq 1$, the KS heuristic consumes the whole graph during Phase 1 and finds a maximum cardinality matching. When $k > 1$, Phase 1 immediately ends, since there is no degree-one vertex. In Phase 2, the first edge (nonzero) consumed by KS is selected from a uniform distribution over the nonzeros whose 645 corresponding rows and columns are still unmatched. Since the block $R_1 \times C_1$ is full, it is more likely that the nonzero will be chosen from this block. Thus, a row in R_1 will be matched with a column in C_1, which is a bad decision since the block $R_2 \times C_2$ is completely empty. Hence, we expect a decrease on the performance of KS as k increases. On the other hand the probability that

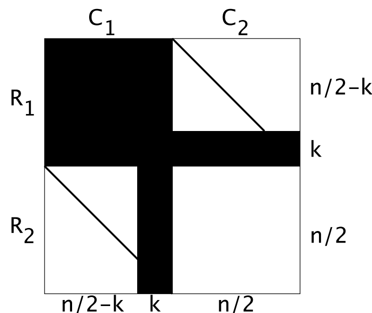


Figure 2: A generic full-sprank matrix structure that is bad for KS.

k	KARP SIPSER	0	1		5		10	
		Qual.	Err.	Qual.	Err.	Qual.	Err.	Qual.
2	0.782	0.522	13.853	0.557	3.463	0.989	0.578	0.999
4	0.704	0.489	11.257	0.516	3.856	0.980	0.604	0.997
8	0.707	0.466	8.653	0.487	4.345	0.946	0.648	0.996
16	0.685	0.448	6.373	0.458	4.683	0.885	0.725	0.990
32	0.670	0.447	4.555	0.453	4.428	0.748	0.867	0.980

(Columns 0–10: Results with different number of scaling iterations)

Table 1: Quality comparison (minimum of 10 executions for each instance) of the KS heuristic and TWOSIDEDMATCH on matrices described in Fig. 2 with $n = 3,200$ and $k \in \{2, 4, 8, 16, 32\}$.

650 TWOSIDEDMATCH chooses an edge from that block goes to zero, as those entries cannot be in a perfect matching.

The results of the experiments are in Table 1. The first column shows the k value. Then the *matching quality* obtained by KS, and by TWOSIDEDMATCH with different number of scaling iterations (0, 1, 5, 10), as well as the scaling 655 error are given. The scaling error is the maximum difference between 1 and each row/column sum of the scaled matrix (for 0 iterations it is equal to $n-1$ for all cases). The quality of a matching is computed by dividing its cardinality to the maximum one, which is $n = 3200$ for these experiments. To obtain the values in each cell of the table, we run the programs 10 times and give the minimum 660 quality (as we are investigating the worst-case behavior). The highest variance for KS and TWOSIDEDMATCH were (up to four significant digits) 0.0041 and 0.0001, respectively. As expected, when k increases, the KS heuristic performs worse, and the matching quality drops to 0.67 for $k = 32$. TWOSIDEDMATCH's performance increases with the number of scaling iterations. As the experiment 665 shows, only 5 scaling iterations are sufficient to make the proposed two-sided matching heuristic significantly better than KS. However, this number is not enough to reach 0.866 for the matrix with $k = 32$. On this matrix, with 10 iterations, only 2% of the rows/columns remain unmatched.

d	iter	sprank	OneSidedMatch	TwoSidedMatch	d	iter	sprank	OneSidedMatch	TwoSidedMatch
Square 100000×100000									
2	0	78,225	0.770	0.912	4	0	97,787	0.644	0.838
2	1	78,225	0.797	0.917	4	1	97,787	0.673	0.848
2	5	78,225	0.850	0.939	4	5	97,787	0.719	0.873
2	10	782,25	0.879	0.954	4	10	97,787	0.740	0.886
3	0	92,786	0.673	0.851	5	0	99,223	0.635	0.840
3	1	92,786	0.703	0.857	5	1	99,223	0.662	0.851
3	5	92,786	0.756	0.884	5	5	99,223	0.701	0.873
3	10	92,786	0.784	0.902	5	10	99,223	0.716	0.882
Rectangular 100000×120000									
2	0	87,373	0.793	0.912	4	0	99,115	0.729	0.899
2	1	87,373	0.815	0.918	4	1	99,115	0.754	0.910
2	5	87,373	0.861	0.939	4	5	99,115	0.792	0.933
2	10	87,373	0.886	0.955	4	10	99,115	0.811	0.946
3	0	96,564	0.739	0.896	5	0	99,761	0.725	0.905
3	1	96,564	0.769	0.904	5	1	99,761	0.749	0.917
3	5	96,564	0.813	0.930	5	5	99,761	0.781	0.936
3	10	96,564	0.836	0.945	5	10	99,761	0.792	0.943

Table 2: Matching qualities of the proposed heuristics on random sparse matrices with uniform nonzero distribution. Square matrices in the top half, rectangular matrices in the bottom half. d: average number of nonzeros per row.

4.1.3. Matching quality on bipartite graphs without perfect matchings

670 We analyze the proposed heuristics on a class of random sprank-deficient square ($n = 100000$) and rectangular ($m = 100000$ and $n = 120000$) matrices with a uniform nonzero distribution (two more sprank-deficient matrices are used in the scalability tests as well). These matrices are generated by Matlab's `sprand` command (generating Erdös-Rényi random matrices [36]). The total 675 nonzeros is set to be around $d \times 100000$ for $d \in \{2, 3, 4, 5\}$. The top half of Table 2 presents the results of this experiment with square matrices, and the bottom half presents the results with the rectangular ones.

As in the previous experiments, the matching qualities in the table is the minimum of 10 executions for the corresponding instances. As Table 2 shows, 680 when the deficiency is high (correlated with small d), it is easier for our algorithms to approximate the maximum cardinality. However, when d gets larger, the algorithms require more scaling iterations. Even in this case, 5 iterations are sufficient to achieve the guaranteed qualities. In the square case, the minimum quality achieved by OneSidedMatch and TwoSidedMatch were 0.701 685 and 0.873 In the rectangular case, the minimum quality achieved by OneSidedMatch and TwoSidedMatch were 0.781 and 0.930, respectively, with 5 scaling iterations. In all cases, increased scaling iterations results in higher quality matchings, in accordance with the previous results on square matrices with perfect matchings.

Name	n	# of edges	Avg. deg.	std_A	std_B	$\frac{sprank}{n}$
atmosmodl	1,489,752	10,319,760	6.9	0.3	0.3	1.00
audikw_1	943,695	77,651,847	82.2	42.5	42.5	1.00
cage15	5,154,859	99,199,551	19.2	5.7	5.7	1.00
channel	4,802,000	85,362,744	17.8	1.0	1.0	1.00
europe_osm	50,912,018	108,109,320	2.1	0.5	0.5	0.99
Hamrle3	1,447,360	5,514,242	3.8	1.6	2.1	1.00
hugebubbles	21,198,119	63,580,358	3.0	0.03	0.03	1.00
kkt_power	2,063,494	12,771,361	6.2	7.45	7.45	1.00
nlpkkt240	27,993,600	760,648,352	26.7	2.22	2.22	1.00
road_usa	23,947,347	57,708,624	2.4	0.93	0.93	0.95
torso1	116,158	8,516,500	73.3	419.59	245.44	1.00
venturiLevel3	4,026,819	16,108,474	4.0	0.12	0.12	1.00

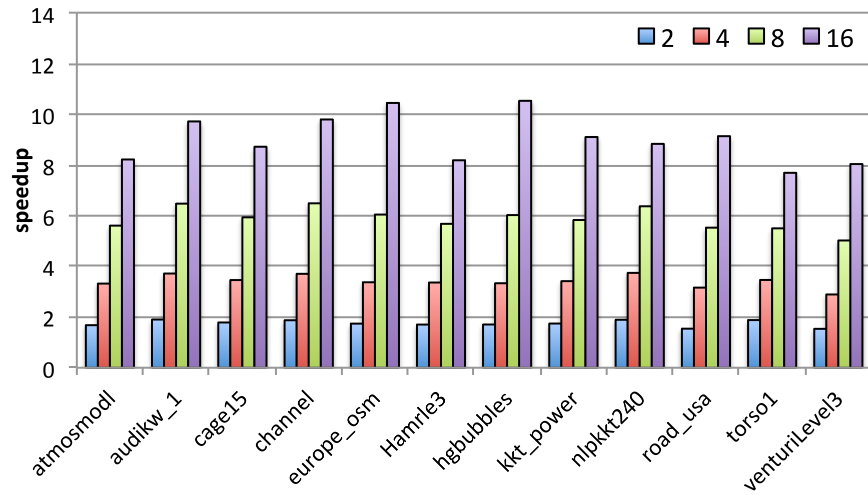
Table 3: Properties of the matrices used in the experiments. In the table, sprank is the maximum matching cardinality, Avg. deg. is the average vertex degree, std_A and std_B are the standard deviations of A and B vertex (row and column) degrees, respectively.

690 *4.2.* OneSidedMatch *and* TwoSidedMatch *in parallel*

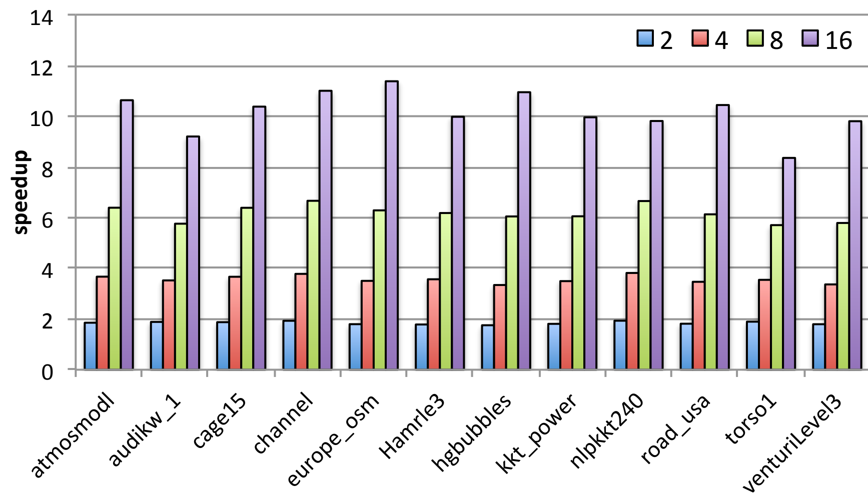
To analyze the scalability of the proposed heuristics in practice, we used 12 large bipartite graphs corresponding to real-life matrices from UFL collection arising in different application domains. The names hugebubbles and channel refer to the matrices hugebubbles-00020 and channel-500x100x100-b050, respec-
695 tively. The properties of the bipartite graphs are given in Table 3 and the sequential run times (the run time of OneSidedMatch includes that of ScaleSK, and TwoSidedMatch includes those of ScaleSK and KarpSipserMT) along with the scaling errors for 1, 5, and 10 iterations are given in Table 4. All the executions in this experiment are repeated 20 times and the first five are ig-
700 nored. The times are computed by using the geometric mean of the remaining 15 executions for each instance. No significant variances were observed among the individual run times. The speedup values are computed with respect to the execution with a single-thread.

Figures 3(a) and 3(b) show the individual speedup values for ScaleSK
705 and the proposed matching heuristic OneSidedMatch, respectively. When executed with 16 threads, ScaleSK obtains a speedup value around 8 or more for all matrices. The maximum speedup of 10.6 is obtained for hugebubbles. The scalability of OneSidedMatch is better. For 10 matrices, a speedup value around 10 or more is obtained. The maximum speedup of 11.4 is obtained for
710 the matrix europe_osm with 16 threads.

The structure of a matrix can affect the scalability. Both for ScaleSK and OneSidedMatch, the minimum speedups (7.7 and 8.4, respectively) are obtained on torso1. As Table 3 shows, torso1 and audikw_1 are the two smallest matrices with less than 10^6 rows and columns and a high variance on
715 their degree distributions. As Figure 3(b) shows, OneSidedMatch obtains its worst speedups on these matrices. This is not a coincidence; when the *coefficient*



(a) ScaleSK



(b) OneSidedMatch

Figure 3: Speedups for ScaleSK and OneSidedMatch with a single scaling iteration.

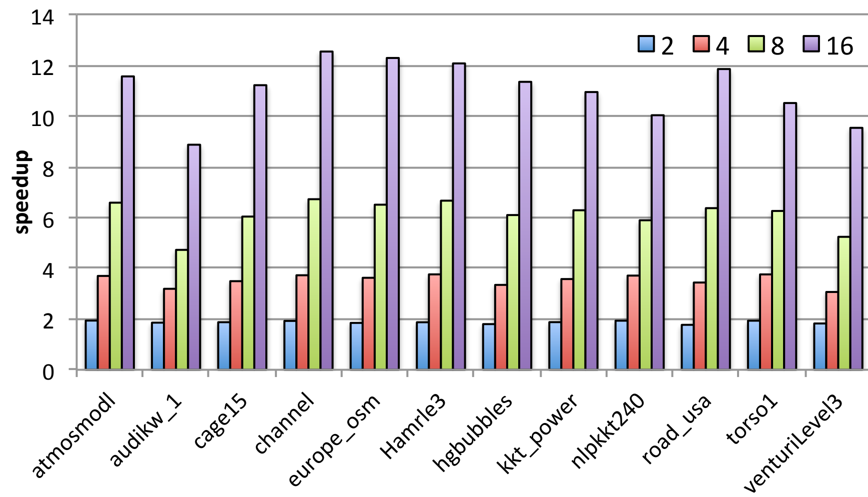
Name	Scaling error (number of iters.) 1	5	10	Exec. times with single thread (secs) ScaleSK	OneSidedMatch	KarpSipserMT	TwoSidedMatch
atmosmodl	0.06	0.01	0.00	0.037	0.095	0.236	0.273
audikw_1	0.17	0.02	0.01	0.188	0.364	0.452	0.640
cage15	0.18	0.03	0.02	0.306	0.627	1.373	1.679
channel	0.10	0.01	0.00	0.274	0.537	0.937	1.211
europe_osm	8.43	8.00	8.00	1.625	3.599	9.643	11.270
Hamrle3	0.99	0.37	0.15	0.028	0.067	0.196	0.223
hugebubbles	0.33	0.17	0.11	1.303	2.840	7.942	9.251
kkt_power	13.83	1.27	1.00	0.063	0.132	0.339	0.401
nlpkkt240	2.23	0.99	0.71	1.864	3.704	6.642	8.481
road_usa	6.08	6.00	6.00	0.712	1.581	4.237	4.949
torso1	0.13	0.02	0.01	0.021	0.040	0.045	0.066
venturiLevel3	0.23	0.05	0.03	0.094	0.239	0.672	0.766

Table 4: Scaling error: the maximum difference between one and column sums; Sequential execution times of ScaleSK (per itaration), OneSidedMatch, KarpSipserMT, and TwoSidedMatch.

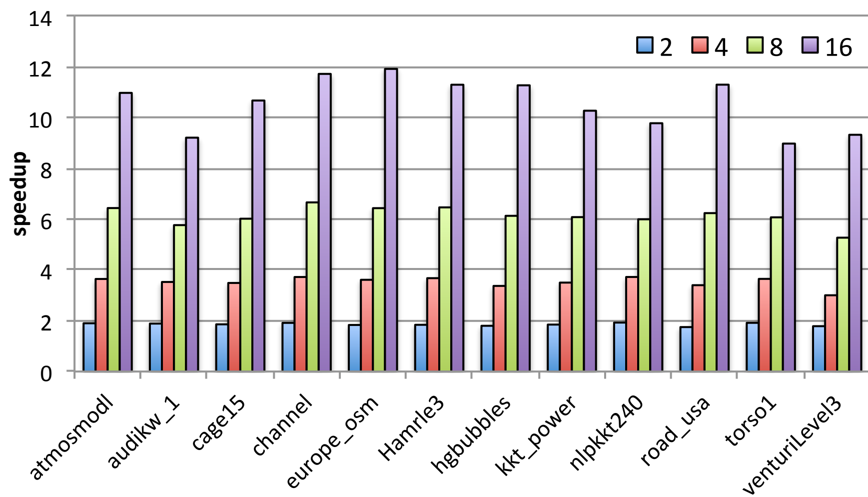
of variation (the ratio of the standard deviation to the mean) on the number of nonzeros per row is high, the effect of the load imbalance can be significant. Although we conducted a set of preliminary experiments on OpenMP scheduling
720 policies, we did not fine tune it to have the best one. We believe that the OpenMP `dynamic` policy should perform well especially for these matrices with a high variance on the number of nonzeros per row. However, one can still obtain a better performance with a different chunk size (other than 512) fine tuned for each matrix.

725 We repeated the scalability experiment for KarpSipserMT and TwoSidedMatch on the same matrix set. The results for the first set can be seen in Figures 4(a) and 4(b), respectively. On average (geometric mean), KarpSipserMT obtains a speedup of 11.1 with 16 threads. The maximum speedup of 12.6 is obtained on the matrix `channel`. These results show that the pro-
730 posed KarpSipserMT is highly scalable on the graphs generated in TwoSidedMatch without any quality loss with the increasing thread counts (see [9] for an efficient but inexact parallel KS implementation).

The parallel performance of the KarpSipserMT algorithm depends on the distribution of the out-one vertex paths to the threads. Throughout the execu-
735 tion, starting from an initially out-one vertex, the threads follow these paths that intersect with each other. In our implementation, at the points of intersection, one of the threads continue, and the others are assigned to the remaining paths, if any. To analyze how balanced these paths are distributed to the threads, we first investigated the maximum path length traversed by a thread and did
740 not observe a path length larger than 20 in any of the executions. We then counted the edges processed by each thread for the *edge selection* (lines 2-7 of



(a) KarpSipserMT



(b) TwoSidedMatch

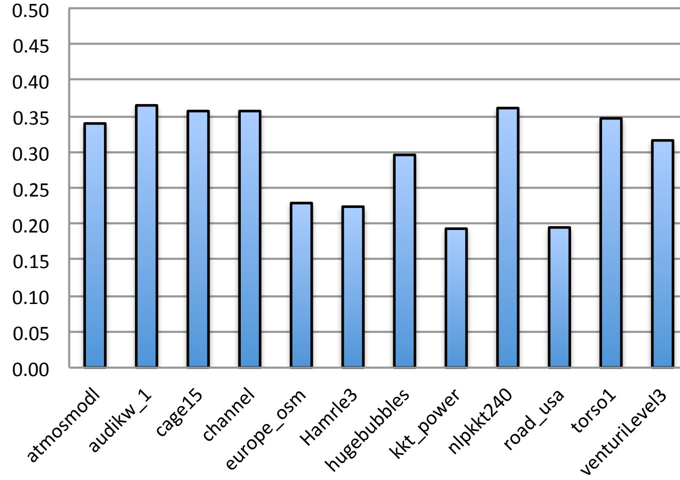
Figure 4: Speedups for KarpSipserMT and TwoSidedMatch with a single scaling iteration.

Algorithm 3 where the **rchoice** and **cchoice** arrays are constructed) and KARP-SIPSERMT phases of TWOSIDEDMATCH and measured the imbalance. Here we give the details of these experiments.

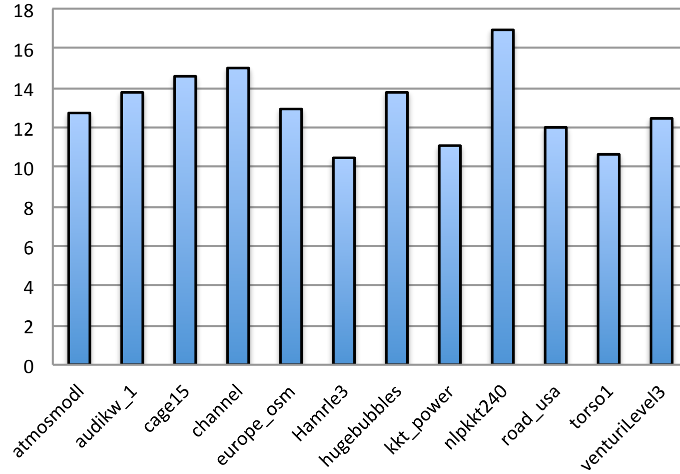
745 We counted the number of out-one paths and the maximum path length traversed by a thread in KARPSIPSERMT after 10 scaling iterations and using 16 threads. We repeated the experiment 20 times and report the averages. Figure 5(a) shows the number of such paths normalized with respect to the number of vertices, $2n$, for the 12 matrices in Table 3. As expected, the average number 750 of initial out-one vertices is always smaller than 0.37 which is the difference between one and the quality guarantee of ONESIDEDMATCH. Figure 5(b) shows the average maximum path length for the same set of matrices. As the numbers show, one thread does not follow a very long path before going for the next one. Hence, we do not expect that the imbalance on the workload will have significant 755 impact on the performance of KARPSIPSERMT. We repeated the same experiment on an R-MAT [37] graph with $n = 10^7$ vertices and $m = 10^9$ edges generated with probabilities $(0.45, 0.15, 0.15, 0.25)$ using GTgraph [38]. The R-MAT graphs are well known and this experiment gives additional insight on the length of the paths traversed by different threads. For the R-MAT matrix we 760 used, the standard deviations std_A and std_B are computed as 183.5 and 99.5. In this experiment with 20 repetitions, the average ratio of the out-one paths to $2n$ is 0.31 and the average maximum path length is 16.

We computed the imbalance on the workload by counting the number of edges processed by each thread during the edge selection and KARPSIPSERMT 765 phases of TWOSIDEDMATCH. We performed the experiment with 8 and 16 threads where we executed the heuristic 20 times for each case. Figures 6(a) and 6(b) show the average imbalance where for each execution, the imbalance is computed as the ratio of the maximum number of edges per thread to the average minus 1. As expected, compared to 8-thread setting, the imbalance values are 770 generally higher for 16 threads. For KARPSIPSERMT, the imbalance values are quite nice and less than one except for `Hamrle3` with 16 threads. This explains the good scalability of KARPSIPSERMT in Figure 4(a). As Figure 6(a) shows, the edge selection part of TWOSIDEDMATCH suffers more from the workload imbalance. However, for 8 out of 12 matrices, the imbalance values are still less 775 than or close to one. But for `audikw_1`, `Hamrle3`, `kkt_power`, and `torso1` the values can go over two with 16 threads. To analyze this behavior, in Figure 7, we sorted the matrices with respect to their coefficients of variation (CV) and plot the values along with the corresponding imbalances for 16 threads. We chose to use CV instead of standard deviation since the deviation alone (without mean) 780 is not very useful to analyze the dispersion. In contrast, both CV and imbalance are dimensionless numbers. As the figure shows, although the correlation is not exact, the imbalance values tend to increase and for higher CVs, we have higher imbalance values.

We repeated the scalability experiment with all the matrices in UFL having 785 more than billion nonzeros (at the time of writing there were three of them) to test the proposed heuristics with large memory requirements. We measured the speedup values with 16 threads. Originally, these matrices had empty rows



(a) Number of out-one paths normalized to $2n$ for an $n \times n$ matrix

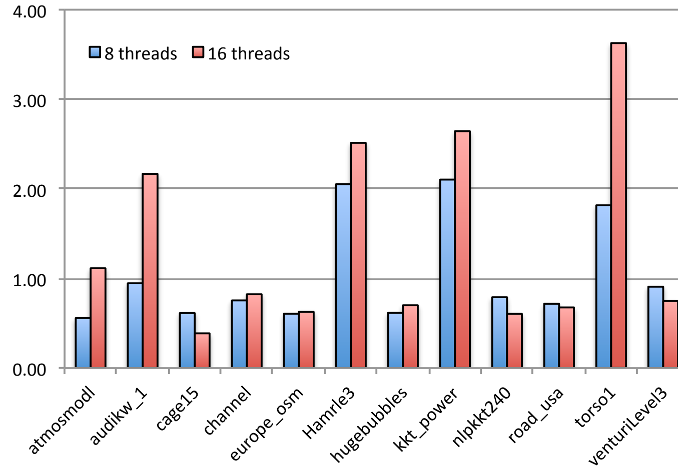


(b) Maximum out-one path length

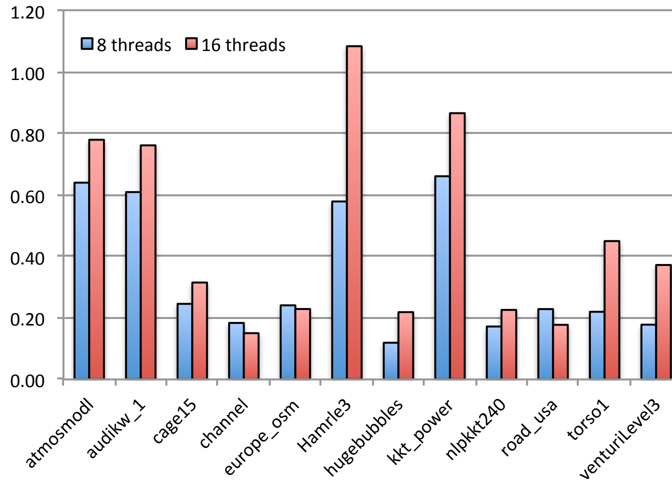
Figure 5: Normalized number of out-one vertex paths and the maximum path length followed by the threads. For this experiment, 10 scaling iterations and 16 threads have been used. The numbers are the averages of 20 independent executions.

			Degree			Speedup		
Name	n	# of edges	Min	Max	Avg	SK	1SD	2SD
`it-2004`	41,291,594	1,177,010,503	1	9,965	28.5	9.9	12.6	12.1
`webbase-2001`	118,142,155	1,110,987,046	1	3,842	9.4	10.3	12.8	12.6
`sk-2005`	50,636,154	1,980,929,102	1	12,870	39.1	9.4	12.4	11.0

Table 5: The speedup values for ScaleSK (SK), OneSidedMatch (1SD), and TwoSidedMatch (2SD) with 16 threads for the billion-scale graphs from UFL.



(a) The imbalance on the number of edges processed by each thread during the edge selection.



(b) The imbalance on the the number of edges processed by each thread during KarpSipserMT.

Figure 6: The imbalances on the number of processed edges for (a) the edge selection and (b) KarpSipserMT within TwoSidedMatch for 8 and 16 threads. The values are the averages of 20 executions.

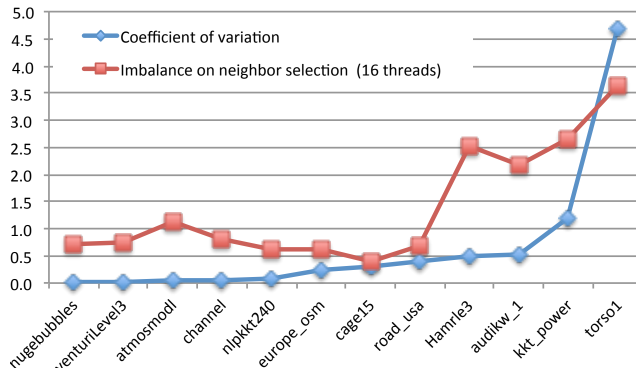


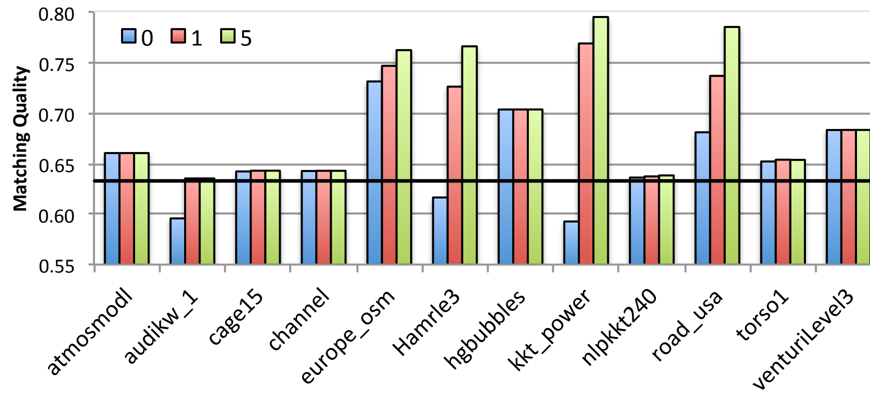
Figure 7: Imbalance vs. coefficient of the variation: the matrices are sorted along wide the x-axis with respect to their coefficients of variation that is the ratio of the overall standard deviation of their degree distribution to the average degree. The imbalance values are the averages of 20 executions with 16 threads.

and columns: to avoid complications, we added one additional nonzero to the diagonal of each empty row/column. The properties of the matrices and the speedup values for ScaleSK, OneSidedMatch, and TwoSidedMatch are in Table 5. As seen in this table, the speed up values obtained for these large matrices are slightly better than the averages reported before.

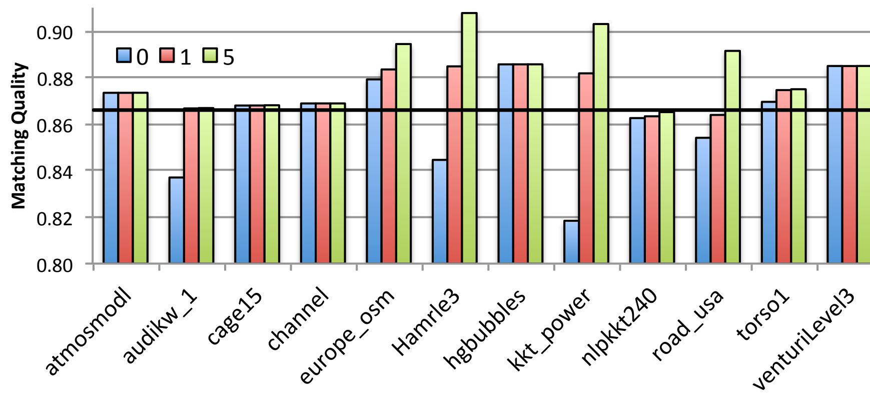
With 16 threads, TwoSidedMatch obtains a speedup value of 10.6 on the average. Compared to the average of OneSidedMatch (10.1), it is slightly better. However, in a sequential setting, TwoSidedMatch is 2.6 times slower. To further compare these two heuristics, we present Figures 8(a) and 8(b) that show the qualities on our test matrices. In the figures, the first columns represent the case that the neighbors are picked from a uniform distribution over the adjacency lists, i.e., the case with no scaling, hence no guarantee. The quality guarantees are achieved with only 5 iterations for almost all the cases except TwoSidedMatch on `nlpkkt240` which required 15 scaling iterations. Even with a single iteration, the quality of TwoSidedMatch is more than 0.86 for all matrices. Only for two among them, the quality is between 0.863 and 0.866. The results are similar for OneSidedMatch. However, even with 10 scaling iterations, OneSidedMatch cannot achieve a quality of 0.80 on any of the matrices. We conclude that OneSidedMatch is faster, TwoSidedMatch has a better quality guarantee, and both demonstrate good speedups.

4.3. Further results with existing codes

We run Azad et al.'s [9] variant of Karp-Sipser (`ksV`), the two greedy matching heuristics of Blelloch et al. [21] (`inc` and `nd` which are referred to as "incremental" and "non-deterministic reservation" in the original paper), and the proposed OneSidedMatch and TwoSidedMatch heuristics with one and 16 threads on the matrices given in Table 3. We also add one more matrix `wc_50K_64` from the family shown in Fig. 2 with $n = 50000$ and $k = 64$. The



(a) OneSidedMatch



(b) TwoSidedMatch

Figure 8: Matching qualities of OneSidedMatch and TwoSidedMatch. The horizontal lines are at 0.866 and 0.632, respectively, which are the approximation guarantees for the heuristics (conjectured for TwoSidedMatch). Legend contains iteration numbers.

	Run time in seconds									
	One thread					16 threads				
Name	ksV	inc	nd	1SD	2SD	ksV	inc	nd	1SD	2SD
atmosmodl	0.20	14.80	0.07	0.12	0.30	0.03	1.63	0.01	0.01	0.03
audikw_1	0.98	206.00	0.22	0.55	0.82	0.10	16.90	0.02	0.06	0.09
cage15	1.54	9.96	0.39	0.92	1.95	0.29	1.10	0.04	0.09	0.19
channel	1.29	105.00	0.30	0.82	1.46	0.14	9.07	0.03	0.08	0.13
europe_osm	12.03	51.80	2.06	4.89	11.97	1.20	7.12	0.21	0.46	1.05
Hamrle3	0.15	0.12	0.04	0.09	0.25	0.02	0.02	0.01	0.01	0.02
hugebubbles	8.36	4.65	1.28	3.92	10.04	0.95	0.53	0.18	0.37	0.94
kkt_power	0.55	0.68	0.09	0.19	0.45	0.08	0.16	0.01	0.02	0.05
nlpkkt240	10.88	1900.00	2.56	5.56	10.11	1.15	237.00	0.23	0.58	1.06
road_usa	6.23	3.57	0.96	2.16	5.42	0.70	0.38	0.09	0.22	0.50
torso1	0.11	7.28	0.02	0.06	0.09	0.02	1.20	0.00	0.01	0.01
venturiLevel3	0.45	2.72	0.13	0.32	0.84	0.08	0.29	0.01	0.04	0.09
wc_50K_64	7.21	3200.00	1.46	4.39	5.80	1.17	402.00	0.12	0.55	0.59
	Quality of the obtained matching									
Name	ksV	inc	nd	1SD	2SD	ksV	inc	nd	1SD	2SD
atmosmodl	1.00	1.00	1.00	0.66	0.87	0.98	1.00	0.97	0.66	0.87
audikw_1	1.00	1.00	1.00	0.64	0.87	0.95	1.00	0.97	0.64	0.87
cage15	1.00	1.00	1.00	0.64	0.87	0.95	1.00	0.91	0.64	0.87
channel	1.00	1.00	1.00	0.64	0.87	0.99	1.00	0.99	0.64	0.87
europe_osm	0.98	0.95	0.95	0.75	0.89	0.98	0.95	0.94	0.75	0.89
Hamrle3	1.00	0.84	0.84	0.75	0.90	0.97	0.84	0.86	0.75	0.90
hugebubbles	0.99	0.96	0.96	0.70	0.89	0.96	0.96	0.90	0.70	0.89
kkt_power	0.85	0.79	0.79	0.78	0.89	0.87	0.79	0.86	0.78	0.89
nlpkkt240	0.99	0.99	0.99	0.64	0.86	0.99	0.99	0.99	0.64	0.86
road_usa	0.94	0.81	0.81	0.76	0.88	0.94	0.81	0.81	0.76	0.88
torso1	1.00	1.00	1.00	0.65	0.88	0.98	1.00	0.98	0.65	0.87
venturiLevel3	1.00	1.00	1.00	0.68	0.88	0.97	1.00	0.96	0.68	0.88
wc_50K_64	0.50	0.50	0.50	0.81	0.98	0.70	0.50	0.51	0.81	0.98

Table 6: Run time and quality of different heuristics. OneSidedMatch and TwoSidedMatch are labelled with 1SD and 2SD, respectively. The heuristic `ksV` is from Azad et al. [9], and the heuristics `inc` and `nd` are from Blelloch et al. [21].

815 OneSidedMatch and TwoSidedMatch heuristics are run with two scaling iterations, and the reported run time includes all components of the heuristics. The `inc` and `nd` heuristics are compiled with `g++ 4.4.5` and `ksV` is compiled with `gcc 4.4.5`. For all the heuristics, we used `-O2` optimization and the appropriate OpenMP flag for compilation. The run time of all heuristics (in seconds)
820 and their matching qualities are given in Table 6.

As seen in Table 6, `nd` is the fastest of the tested heuristics with both one thread and 16 threads on this data set. The second fastest heuristic is OneSidedMatch. The heuristic `ksV` is faster than TwoSidedMatch on six instances with one thread. With 16 threads, TwoSidedMatch is almost always faster
825 than `ksV`—the absolute difference is small as both heuristics are quite efficient and have a maximum run time of 1.17 seconds. The heuristic `inc` was observed to have large run time on some matrices with relatively high nonzeros per row. All the existing heuristics are always very efficient, except `inc` which had difficulties on some matrices in the data set. In the original reference [21], `inc` is quite

830 efficient in some other matrices (where the codes are compiled with Cilk). We do not see run time with nd elsewhere, but in our data set it was always better than inc. The proposed OneSidedMatch heuristic's quality is almost always close to its theoretical limit. TwoSidedMatch also obtains quality results close to its theoretical limit. Looking at the quality of the matching with one thread,
835 we see that ksV obtains (almost always) the best score, except on kkt_power and wc_50K_64, where TwoSidedMatch obtains the best score. The heuristics inc and nd obtain the same score with one thread, but with 16 threads inc obtains better quality than nd almost always. OneSidedMatch never obtains the best score (TwoSidedMatch is always better), yet it is better than ksV,
840 inc and nd on the synthetic wc_50K_64 matrix. The TwoSidedMatch heuristic obtains better results than inc and nd on Hamrle3, kkt_power, road_usa, and wc_50K_64 with both one thread and 16 threads. Also on hugebubbles with 16 threads, the difference between TwoSidedMatch and nd is 1% (in favor of nd).

845 **5. Conclusion**

We proposed two heuristics for the bipartite maximum cardinality matching problem. The first one, OneSidedMatch, is shown to have an approximation guarantee of $1-1/e \approx 0.632$. The second heuristic, TwoSidedMatch, is shown to have an approximation guarantee of 0.866. Both algorithms use well-known
850 methods to scale the sparse matrix associated with the given bipartite graph to a doubly stochastic form whose entries are used as the probability density functions to randomly select a subset of the edges of the input graph. OneSidedMatch selects exactly n edges to construct a subgraph in which a matching of the guaranteed cardinality is identified with virtually no overhead, both in
855 sequential and parallel execution. TwoSidedMatch selects around $2n$ edges and then runs the Karp-Sipser (KS) heuristic as an exact algorithm on the selected subgraph to obtain a matching of conjectured cardinality. The subgraphs are analyzed to develop a specialized KS algorithm for efficient parallelization. All theoretical investigations are first performed assuming bipartite graphs with
860 perfect matchings, and the scaling algorithms have converged. Then, theoretical arguments and experimental evidence are provided to extend the results to cover other cases and validate the applicability and practicality of the proposed heuristics in general settings. Parallel performance is analyzed on a shared memory parallel computer with up to 16 threads and speedups beyond 10 fold
865 are demonstrated.

To avoid race conditions with multiple threads, the second heuristic TwoSidedMatch depends on atomic operations which are efficiently supported by many modern multicore architectures. As for manycore devices such as GPU's, recent architectures also have support for atomic operations. For example,
870 NVIDIA Kepler GK110 has a support for 32-bit and 64-bit atomicCAS (compare and swap), atomicAdd, and atomicExch (exchange) operations which can be used to implement various constructs. However, their performance is reported to vary and not as good as the atomicity performance of CPUs. It would

be an interesting experiment to implement the proposed heuristics on such ar-
875 chitectures allergic to atomicity or on the ones without a support by using other techniques for locking and synchronization.

Acknowledgements

Bora Uçar was supported by ANR project SOLHAR (ANR-13-MONU-0007). We thank Mahantesh Halappanavar for providing us with the codes from his
880 earlier work [9].

References

[1] J. E. Hopcroft, R. M. Karp, An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs, SIAM J. Comput. 2 (4) (1973) 225–231.

[2] J. Langguth, F. Manne, P. Sanders, Heuristic initialization for bipartite
885 matching problems, J. Exp. Algorithmics 15 (2010) 1.1–1.22.

[3] J. Magun, Greedy matching algorithms, an experimental study, J. Exp. Algorithmics 3 (1998) 6.

[4] I. S. Duff, K. Kaya, B. Uçar, Design, implementation, and analysis of maximum transversal algorithms, ACM T. Math. Software 38 (2) (2011) 13:1–
890 13:31.

[5] N. McKeown, The iSLIP scheduling algorithm for input-queued switches, IEEE/ACM Trans. Netw. 7 (2) (1999) 188–201.

[6] R. M. Karp, M. Sipser, Maximum matching in sparse random graphs, in: 22nd Annual IEEE Symposium on Foundations of Computer Science
895 (FOCS), Nashville, TN, USA, 1981, pp. 364–375.

[7] K. Kaya, J. Langguth, F. Manne, B. Uçar, Push-relabel based algorithms for the maximum transversal problem, Comput. Oper. Res. 40 (5) (2013) 1266–1275.

[8] R. M. Karp, U. V. Vazirani, V. V. Vazirani, An optimal algorithm for on-
900 line bipartite matching, in: 22nd annual ACM symposium on Theory of computing (STOC), Baltimore, MD, USA, 1990, pp. 352–358.

[9] A. Azad, M. Halappanavar, S. Rajamanickam, E. G. Boman, A. Khan, A. Pothen, Multithreaded algorithms for maximum matching in bipartite graphs, in: IEEE 26th International Parallel Distributed Processing Sym-
905 posium (IPDPS), Shanghai, China, 2012, pp. 860–872.

[10] M. Deveci, K. Kaya, B. Uçar, Ü. V. Çatalyürek, GPU accelerated maximum cardinality matching algorithms for bipartite graphs, in: F. Wolf, B. Mohr, D. an Mey (Eds.), Euro-Par 2013 Parallel Processing, Vol. 8097 of LNCS, Springer Berlin Heidelberg, 2013, pp. 850–861.

[11] M. Deveci, K. Kaya, Ü. V. Çatalyürek, B. Uçar, A push-relabel-based maximum cardinality matching algorithm on GPUs, in: 42nd International Conference on Parallel Processing, Lyon, France, 2013, pp. 21–29.

[12] M. E. Dyer, A. M. Frieze, Randomized greedy matching, Random Struct. Algor. 2 (1) (1991) 29–45.

[13] A. Pothen, C.-J. Fan, Computing the block triangular form of a sparse matrix, ACM T. Math. Software 16 (4) (1990) 303–324.

[14] J. Aronson, M. Dyer, A. Frieze, S. Suen, Randomized greedy matching II, Random Struct. Algor. 6 (1) (1995) 55–73.

[15] M. Poloczek, M. Szegedy, Randomized greedy algorithms for the maximum matching problem with new analysis, in: IEEE 53rd Annual Sym. on Foundations of Computer Science (FOCS), New Brunswick, NJ, USA, 2012, pp. 708–717.

[16] J. Aronson, A. Frieze, B. G. Pittel, Maximum matchings in sparse random graphs: Karp-Sipser revisited, Random Struct. Algor. 12 (2) (1998) 111–177.

[17] Ü. V. Çatalyürek, M. Deveci, K. Kaya, B. Uçar, Multithreaded clustering for multi-level hypergraph partitioning, in: 26th IEEE International Parallel and Distributed Processing Symposium (IPDPS), Shanghai, China, 2012, pp. 848–859.

[18] B. Fagginger Auer, R. Bisseling, A GPU algorithm for greedy graph matching, in: Facing the Multicore Challenge II, Vol. 7174 of LNCS, Springer-Verlag Berlin, 2012, pp. 108–119.

[19] M. Halappanavar, J. Feo, O. Villa, A. Tumeo, A. Pothen, Approximate weighted matching on emerging manycore and multithreaded architectures, Int. J. High Perform. C. 26 (4) (2012) 413–430.

[20] Z. Lotker, B. Patt-Shamir, S. Pettie, Improved distributed approximate matching, in: Proceedings of the Twentieth Annual Symposium on Parallelism in Algorithms and Architectures (SPAA), ACM, Munich, Germany, 2008, pp. 129–136.

[21] G. E. Blelloch, J. T. Fineman, J. Shun, Greedy sequential maximal independent set and matching are parallel on average, in: Proceedings of the Twenty-fourth Annual ACM Symposium on Parallelism in Algorithms and Architectures (SPAA), ACM, 2012, pp. 308–317.

[22] M. Birn, V. Osipov, P. Sanders, C. Schulz, N. Sitchinava, Efficient parallel and external matching, in: F. Wolf, B. Mohr, D. an Mey (Eds.), Euro-Par 2013 Parallel Processing, Vol. 8097 of LNCS, Springer Berlin Heidelberg, 2013, pp. 659–670.

[23] R. A. Brualdi, H. J. Ryser, Combinatorial Matrix Theory, Vol. 39 of Encyclopedia of Mathematics and its Applications, Cambridge University Press, Cambridge, UK; New York, USA; Melbourne, Australia, 1991.

[24] R. Sinkhorn, P. Knopp, Concerning nonnegative matrices and doubly stochastic matrices, Pacific J. Math. 21 (1967) 343–348.

[25] P. A. Knight, The Sinkhorn–Knopp algorithm: Convergence and applications, SIAM J. Matrix Anal. A. 30 (1) (2008) 261–275.

[26] P. A. Knight, D. Ruiz, B. Uçar, A symmetry preserving algorithm for matrix scaling, SIAM J. Matrix Anal. A. 35 (3) (2014) 931–955.

[27] D. Ruiz, A scaling algorithm to equilibrate both row and column norms in matrices, Tech. Rep. TR-2001-034, RAL (2001).

[28] P. A. Knight, D. Ruiz, A fast algorithm for matrix balancing, IMA Journal of Numerical Analysis 33 (3) (2013) 1029–1047.

[29] D. W. Walkup, Matchings in random regular bipartite digraphs, Discrete Math. 31 (1) (1980) 59–64.

[30] A. Meir, J. W. Moon, The expected node-independence number of random trees, Indagationes Mathematicae 76 (1973) 335–341.

[31] M. Karoński, B. Pittel, Existence of a perfect matching in a random $(1 + e^{-1})$–out bipartite graph, J. Comb. Theory Ser. B 88 (2003) 1–16.

[32] A. L. Dulmage, N. S. Mendelsohn, Coverings of bipartite graphs, Can. J. Math. 10 (1958) 517–534.

[33] A. Pothen, Sparse null bases and marriage theorems, Ph.D. thesis, Dept. Computer Science, Cornell Univ., Ithaca, New York (1984).

[34] T. A. Davis, Y. Hu, The University of Florida sparse matrix collection, ACM T. Math. Software 38 (1) (2011) 1:1–1:25.

[35] D. A. Bader, H. Meyerhenke, P. Sanders, D. Wagner (Eds.), Graph Partitioning and Graph Clustering - 10th DIMACS Implementation Challenge Workshop, Georgia Institute of Technology, Atlanta, GA, USA, February 13-14, 2012. Proceedings, Vol. 588 of Contemporary Mathematics, American Mathematical Society, 2013.

[36] P. Erdös, A. Rényi, On random matrices, Publications of the Mathematical Institute of the Hungarian Academy of Sciences 8 (3) (1964) 455–461.

[37] D. Chakrabarti, Y. Zhan, C. Faloutsos, R-MAT: A recursive model for graph mining, in: Proceedings of the SIAM International Conference on Data Mining, 2004, pp. 442–446.

[38] D. A. Bader, K. Madduri, GTgraph: A synthetic graph generator suite (Feb. 2006).
URL http://www.cse.psu.edu/~madduri/software/GTgraph/gen.pdf