# Two approximation algorithms for bipartite matching on multicore architectures 

Fanny Dufossé, Kamer Kaya, Bora Uçar

## To cite this version:

Fanny Dufossé, Kamer Kaya, Bora Uçar. Two approximation algorithms for bipartite matching on multicore architectures. Journal of Parallel and Distributed Computing, 2015, 85, pp.62-78. 10.1016/j.jpdc.2015.06.009 . hal-01242516

HAL Id: hal-01242516
https://inria.hal.science/hal-01242516
Submitted on 12 Dec 2015

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

# Two approximation algorithms for bipartite matching on multicore architectures 

Fanny Dufosséa ${ }^{\text {a }}$, Kamer Kaya ${ }^{\text {b,* }}$, Bora Uçar ${ }^{\text {c }}$<br>${ }^{a}$ Inria Lille, Nord Europe, 59650, Villeneuve d'Ascq, France<br>${ }^{b}$ Sabancı University, Faculty of Engineering and Natural Sciences, Istanbul, Turkey<br>${ }^{c}$ 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, $68 \mathrm{~W} 10,68 \mathrm{~W} 25$

## 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.

[^0]The lowest worst-case time complexity of the known algorithms is $\mathcal{O}(\sqrt{n} \tau)$ for a bipartite graph with $n$ vertices and $\tau$ 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) 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 25 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 $\tau$ 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 35 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 40 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 ${ }_{45}$ 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 back-
${ }_{50}$ ground 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

## 2. Notation and background

Let $G=\left(V_{R} \cup V_{C}, E\right)$ 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 $a_{i j}=1$ 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 $a_{i j} \neq 0$ 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 $G=\left(V_{R} \cup V_{C}, E\right)$ 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 $\tilde{O}\left(n^{1 / 5}\right)$ 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]
present a distributed $1-1 / k$ approximate matching algorithm for bipartite graphs. This nice theoretical algorithm has $O\left(k^{3} \log \Delta+k^{2} \log n\right)$ time steps with a message length of $O(\log \Delta)$ where $\Delta$ 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 and $O\left(\log ^{3} \tau\right)$ depth with high probability on a bipartite graph with $\tau$ 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 same matching (including those found in the sequential version). Birn et al. [22 discuss maximal ( $1 / 2$ approximate) matchings in $O\left(\log ^{2} n\right)$ 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.

### 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 connected. A square sparse matrix $\mathbf{A}$ is called fully indecomposable if for a permutation matrix $\mathbf{Q}$, the matrix $\mathbf{B}=\mathbf{A Q}$ has a zero free diagonal and the directed graph associated with 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 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 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 $\mathbf{D}_{R} \mathbf{A D} \mathbf{D}_{C}$ is equal to one). If $\mathbf{A}$ has support but 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 (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, o 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 $i$ th row and $j$ th 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 A: an \(n \times n\) matrix with total support, \(\varepsilon\) : the error threshold
Output \(\mathbf{d}_{r}, \mathbf{d}_{c}\) : row/column scaling arrays
    for \(i=1\) to \(n\) in parallel do
        \(\mathbf{d}_{r}[i] \leftarrow 1\)
        \(\mathbf{d}_{c}[i] \leftarrow 1\)
    for \(j=1\) to \(n\) in parallel do
        \(\operatorname{csum}[j] \leftarrow \sum_{i \in \mathbf{A}_{* j}} a_{i j}\)
    repeat
        for \(j=1\) to \(n\) in parallel do
            \(\mathbf{d}_{c}[j] \leftarrow 1 / \operatorname{csum}[j]\)
        for \(i=1\) to \(n\) in parallel do
            \(r s u m \leftarrow \sum_{j \in \mathbf{A}_{i *}} a_{i j} \times \mathbf{d}_{c}[j]\)
            \(\mathbf{d}_{r}[i] \leftarrow 1 /\) rsum
        for \(j=1\) to \(n\) in parallel do
            \(\operatorname{csum}[j] \leftarrow \sum_{i \in \mathbf{A}_{* j}} \mathbf{d}_{r}[i] \times a_{i j}\)
    until \(|\max \{1-\operatorname{csum}[j]: 1 \leq j \leq n\}| \leq \varepsilon\)
```

threshold, the method runs until convergence, where we want to stop when both the row sums and column sums are in the $\varepsilon$ 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 the column sums and one as small as possible. At the end, $\mathbf{d}_{r}[i] \times a_{i j} \times \mathbf{d}_{c}[j]$ 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.

### 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 $\left(u_{i}\right)_{i \in S}$ and $\left(v_{i}\right)_{i \in S}$ be two sequences and $S=\{1,2, \ldots,|S|\}$ be their index set. Then we have

$$
\prod_{i \in S}\left(u_{i}+v_{i}\right)=\sum_{S^{\prime} \subseteq S}\left(\prod_{i \in S^{\prime}} u_{i}\right) \cdot\left(\prod_{i \in S \backslash S^{\prime}} v_{i}\right)
$$

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 $S^{\prime \prime}$ be $S \backslash\{k\}$. Then by the inductive hypothesis, the
product $\prod_{i \in S}\left(u_{i}+v_{i}\right)$ is equal to

$$
\begin{aligned}
& \left(u_{k}+v_{k}\right) \cdot \sum_{S^{\prime} \subseteq S^{\prime \prime}}\left(\prod_{i \in S^{\prime}} u_{i}\right) \cdot\left(\prod_{i \in S^{\prime \prime} \backslash S^{\prime}} v_{i}\right) \\
= & u_{k} \sum_{S^{\prime} \subseteq S^{\prime \prime}}\left(\prod_{i \in S^{\prime}} u_{i}\right) \cdot\left(\prod_{i \in S^{\prime \prime} \backslash S^{\prime}} v_{i}\right)+v_{k} \sum_{S^{\prime} \subseteq S^{\prime \prime}}\left(\prod_{i \in S^{\prime}} u_{i}\right) \cdot\left(\prod_{i \in S^{\prime \prime} \backslash S^{\prime}} v_{i}\right) \\
= & \sum_{S^{\prime} \subseteq S, k \in S^{\prime}}\left(\prod_{i \in S^{\prime}} u_{i}\right) \cdot\left(\prod_{i \in S \backslash S^{\prime}} v_{i}\right)+\sum_{S^{\prime} \subseteq S, k \notin S^{\prime}}\left(\prod_{i \in S^{\prime}} u_{i}\right) \cdot\left(\prod_{i \in S \backslash S^{\prime}} v_{i}\right) \\
= & \sum_{S^{\prime} \subseteq S}\left(\prod_{i \in S^{\prime}} u_{i}\right) \cdot\left(\prod_{i \in S \backslash S^{\prime}} v_{i}\right) .
\end{aligned}
$$

## 3. Two matching heuristics

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 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 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 $a_{i j}$ is originally either 0 or 1 ) and uses the scaled entries to randomly choose a column as a match for each row. The pseudocode 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 $\mathbf{S}$ (line1). 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 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., $\{\{\operatorname{cmatch}[j], j\}: \operatorname{cmatch}[j] \neq N I L\}$. We now analyze its approximation guarantee in terms of the matching cardinality.

```
Algorithm 2 OneSidedMatch
Input A: an \(n \times n\), ( 0,1 )-matrix with total support
Output cmatch[•]: the rows matched to columns
    \(\left(\mathbf{d}_{r}, \mathbf{d}_{c}\right) \leftarrow \operatorname{ScaleSK}(\mathbf{A})\)
    for \(j=1\) to \(n\) in parallel do
        cmatch \([j] \leftarrow N I L\)
    for \(i=1\) to \(n\) in parallel do
        Pick a random column \(j \in \mathbf{A}_{i *}\) by using the probability density function
    \(\frac{s_{i k}}{\sum_{\ell \in \mathbf{A}_{i *}} s_{i \ell}}\), for all \(k \in \mathbf{A}_{i *}\)
```

        where \(s_{i k}=\mathbf{d}_{r}[i] \times \mathbf{d}_{c}[k]\) is the corresponding entry in the scaled matrix \(\mathbf{S}=\)
        \(\mathrm{D}_{R} \mathrm{AD}_{C}\).
        cmatch \([j] \leftarrow i\)
    Theorem 1. Let A be an $n \times n$, (0,1)-matrix with total support. Then, OnESidEDMATCH obtains a matching of size at least $n(1-1 / e) \approx 0.632 n$ in expectation as $n \rightarrow \infty$.

Proof. To compute the matching cardinality, we will count the columns that are not picked by any row and subtract it from $n$. Since $\Sigma_{k \in \mathbf{A}_{i *}} s_{i k}=1$ 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 $\prod_{i \in \mathbf{A}_{* j}}\left(1-s_{i j}\right)$. By applying the arithmetic-geometric mean inequality, we obtain

$$
\sqrt[d_{j}]{\prod_{i \in \mathbf{A}_{* j}}\left(1-s_{i j}\right)} \leq \frac{d_{j}-\sum_{i \in \mathbf{A}_{* j}} s_{i j}}{d_{j}}
$$

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

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

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

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

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

$$
\lim _{d_{j} \rightarrow \infty}\left(1-\frac{1}{d_{j}}\right)^{d_{j}}=\frac{1}{e}
$$

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 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 $\left(0, \sum_{k \in \mathbf{A}_{i *}} s_{i k}\right]$. Then, the last nonzero column index $j$ for which $\sum_{1 \leq k \leq j} s_{i k} \leq r$ is found and cmatch $[j]$ is set to $i$.


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 0 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 A: an \(n \times n,(0,1)\)-matrix with total support
Output match[•]: the mate of each vertex or NIL
    \(\left(\mathbf{d}_{r}, \mathbf{d}_{c}\right) \leftarrow \operatorname{SCaleSK}(\mathbf{A})\)
    for \(i=1\) to \(n\) in parallel do
            Pick a random column \(j \in \mathbf{A}_{i *}\) by using the probability density function
                \(\frac{s_{i k}}{\Sigma_{\ell \in \mathbf{A}_{i *}} s_{i \ell}}\), for all \(k \in \mathbf{A}_{i *}\)
        where \(s_{i k}=\mathbf{d}_{r}[i] \times \mathbf{d}_{c}[k]\) is the corresponding entry in the scaled matrix \(\mathbf{S}=\)
        \(\mathrm{D}_{R} \mathrm{AD}_{C}\).
        rchoice \([i] \leftarrow j\)
    for \(j=1\) to \(n\) in parallel do
            Pick a random row \(i \in \mathbf{A}_{* j}\) by using the probability density function
                                    \(\frac{s_{k j}}{\sum_{\ell \in \mathbf{A}_{* j}} s_{\ell j}}\), for all \(k \in \mathbf{A}_{* j}\).
            cchoice \([j] \leftarrow i\)
    Construct a bipartite graph \(G=\left(V_{R} \cup V_{C}, E\right)\) where
        \(E=\{\{i\), rchoice \([i]\}: i \in\{1, \ldots, n\}\} \cup\)
                            \(\{\{\) cchoice \([j], j\}: j \in\{1, \ldots, n\}\}\).
    match \(\leftarrow \operatorname{KARPSIPSER}(G)\)
```

of the cycle can be used to match a pair of vertices. This decision necessarily 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 vertex 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 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 replaced. The final content is returned. The third operation _ADDAndFETCH (memory, value) atomically 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 \([\cdot]\) : the chosen vertex for each \(u \in V\}\)
Output match \([\cdot]\) : the match array for \(u \in V\)
    for all \(u \in V\) in parallel do
        \(\operatorname{mark}[u] \leftarrow 1\)
        \(\operatorname{deg}[u] \leftarrow 1\)
        \(\operatorname{match}[u] \leftarrow N I L\)
    for all \(u \in V\) in parallel do
        \(v \leftarrow\) choice \([u]\)
        \(\operatorname{mark}[v] \leftarrow 0\)
        if choice \([v] \neq u\) then
            _Add (deg \([v], 1)\)
    for each vertex \(u\) in parallel do Phase 1: out-one vertices
        if \(\operatorname{mark}[u]=1\) then
        curr \(\leftarrow u\)
        while curr \(\neq\) NIL do
            \(n b r \leftarrow\) choice[curr]
            if _CompAndSwap \((\) match \([n b r], N I L\), curr \()=\) curr then
                match \([\) curr \(] \leftarrow n b r\)
                curr \(\leftarrow N I L\)
                \(n e x t \leftarrow\) choice \([n b r]\)
                if match \([n e x t]=N I L\) then
                    if _AddAndFetch \((\operatorname{deg}[n e x t],-1)=1\) then
                        curr \(\leftarrow n e x t\)
                else
                    curr \(\leftarrow\) NIL
    for each column vertex \(u\) in parallel do \(\quad\) Phase 2: the rest
        \(v \leftarrow\) choice \([u]\)
        if match \([u]=N I L\) and \(\operatorname{match}[v]=N I L\) then
        \(\operatorname{match}[u] \leftarrow v\)
        \(\operatorname{match}[v] \leftarrow u\)
```

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 $v=$ choice $[u]$. Then $u$ is called:

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

The first phase of KarpSipserMT (for loop of line 10) does not track and 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 $\operatorname{mark}[u]=1$. 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 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 $K S$.
Observation 2. A degree-one vertex in $K S$ is either an out-one or an inone vertex, or it is one of the two vertices $u$ and $v$ in a 2-clique such that $v=$ choice $[u]$ and $u=$ 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 choice $[v]=u$. Let $\mathcal{P}$ be the longest vertex sequence $w_{1}, w_{2}, \ldots, w_{k}, u$ such that all $w_{i}$ s are unmatched, choice $\left[w_{i}\right]=w_{i+1}$ 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 cycle, is uniquely defined by choice. But $u$ is an in-one vertex and choice $[u]$ is already matched. Thus $\mathcal{P}$ has a finite length, and an out-one vertex $\left(w_{1}\right)$ 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 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.

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

Lemma 4. Let $G^{\prime}=\left(V_{R}^{\prime} \cup V_{C}^{\prime}, E^{\prime}\right)$ be the graph induced by the remaining vertices after the first phase of KarpSipsermT. Then, the set

$$
\left\{(u, \text { choice }[u]): u \in V_{R}^{\prime}, \text { choice }[u] \in V_{C}^{\prime}\right\}
$$

is a maximum cardinality matching in $G^{\prime}$.
Proof. Apart from 2-cliques, no out-one or in-one (that is no degree-one) vertex remains after Phase 1. A component of $G^{\prime}$ can be a trivial (a singleton vertex), unmatched, and there is no other unmatched vertex $y$ with 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 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).
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 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 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 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 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 1 s ; that is $a_{i j}=1$ for all $1 \leq i, j \leq n$, then the doubly stochastic matrix $\mathbf{S}$ is such that $s_{i j}=\frac{1}{n}$ for all $1 \leq i, j \leq n$. In this case, the graph $G$ created by Algorithm 3 is a random 1out 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 $2(1-\Omega) n \approx 0.866 n$ 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 any square matrix $\mathbf{A}$ with total support. We state this as a theorem.

Theorem 2. Let A be an $n \times n$ matrix with total support. Then, TwoSidedMATCH obtains a matching of size $2(1-\Omega) n \approx 0.866 n$ in expectation as $n \rightarrow \infty$, 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 $\tilde{O}\left(n^{1 / 5}\right)$ 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 $G=\left(A, B, E^{\prime}\right)$ 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 $S^{\prime} \subseteq B, g\left(S^{\prime}\right)=\bigcup_{b \in S^{\prime}}\{g(b)\}$. 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 $f\left(A_{1}\right)$. If $A_{1}=\emptyset$ all the vertices of $A$ belong to a cycle, and hence, to a perfect matching. Otherwise, $\left|A_{1}\right|-\left|B_{1}\right|$ 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 $G^{\prime}=\left(A \backslash A_{1}, B \backslash B_{1}, E^{\prime} \cap\left(A \backslash A_{1}\right) \times\left(B \backslash B_{1}\right)\right)$.

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 $B_{k}=f\left(A_{k}\right)$, and let $A_{k}$ be the set of vertices not picked by any $B \backslash B_{k-1}$ vertex, that is $A_{k}=A \backslash g\left(B \backslash B_{k-1}\right)$. 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, $\left|A_{k}\right|,\left|B_{k}\right|$ and $\left|A_{k}\right|-\left|B_{k}\right|$ are increasing sequences of $k$. The analysis simulates a batched KS variant where at each step $k$, all $B_{k} \backslash B_{k-1}$ vertices are matched to the $A_{k} \backslash A_{k-1}$ vertices at once leaving some of $A_{k} \backslash A_{k-1}$ 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 $A \backslash A_{n}$ is perfectly matchable to $B \backslash B_{n}$. Therefore, the number of $A$ vertices that will remain unmatched is $\left|A_{n}\right|-\left|B_{n}\right|$ and the cardinality of the matching found by KS will be $|A|-\left|A_{n}\right|+\left|B_{n}\right|$.

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 $P\left(a_{i} \in A_{k}\right)$ 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 $P\left(b_{j} \in B_{k}\right)$ that $b_{j}$ belongs to $B_{k}$ for $j, k \in\{1,2, \ldots, n\}$. Note that $\sum_{i} \alpha_{i}^{(n)}-\sum_{j} \beta_{j}^{(n)}=$ $E\left(\left|A_{n}\right|-\left|B_{n}\right|\right)$.

Theorem 2 will be proved by bounding the deficiency of the matching by upper bounding $\left|A_{n}\right|-\left|B_{n}\right|$.

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

$$
\beta_{j}^{(k)} \geq 1-e^{-\left(\sum_{i=1}^{n} s_{i j} \cdot \alpha_{i}^{(k)}\right)} .
$$

Proof. We write $\beta_{j}^{(k)}=P\left(b_{j} \in B_{k}\right)$ as

$$
\begin{equation*}
1-\sum_{A^{\prime} \subseteq A} P\left(f^{-1}\left(b_{j}\right)=A^{\prime}\right) \cdot P\left(\bigcap_{a_{i} \in A^{\prime}} a_{i} \notin A_{k}\right) \tag{1}
\end{equation*}
$$

where the first term of the summation is

$$
\begin{equation*}
P\left(f^{-1}\left(b_{j}\right)=A^{\prime}\right)=\left(\prod_{a_{i} \in A^{\prime}} s_{i j}\right) \cdot\left(\prod_{a_{i} \notin A^{\prime}}\left(1-s_{i j}\right)\right) . \tag{2}
\end{equation*}
$$

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 $B \backslash B_{k-1}$. We first prove by induction on $\left|A^{\prime}\right|$ that $P\left(\bigcap_{a_{i} \in A^{\prime}} E_{i}\right) \leq \prod_{a_{i} \in A^{\prime}} P\left(E_{i}\right)$. For $\left|A^{\prime}\right|=1$, the (in)equality is evident; assume that it is correct also for all $A^{\prime}$ with $\left|A^{\prime}\right|<\ell$. Let $A^{\prime}$ be a subset of $A$ of size $\ell$ having the vertex $a_{u}$. Let $A^{\prime \prime}=A^{\prime} / a_{u}$. By Bayes law and the induction assumption, we have:

$$
P\left(\bigcap_{a_{i} \in A^{\prime}} E_{i}\right)=P\left(\bigcap_{a_{i} \in A^{\prime \prime}} E_{i}\right) \cdot P\left(E_{u} \mid \bigcap_{a_{i} \in A^{\prime \prime}} E_{i}\right) \leq \prod_{a_{i} \in A^{\prime \prime}} P\left(E_{i}\right) \cdot P\left(E_{u} \mid \bigcap_{a_{i} \in A^{\prime \prime}} E_{i}\right) .
$$

We now prove that $P\left(E_{u} \mid \bigcap_{a_{i} \in A^{\prime \prime}} E_{i}\right) \leq P\left(E_{u}\right)$. Intuitively, the $A$ vertices will be selected by the previously decided $B \backslash B_{k-1}$ 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 $A^{\prime \prime}$ are already selected by the vertices in $B \backslash B_{k-1}$, then fewer selection options and less chances remain for $a_{u}$ to be selected. More formally, $P\left(E_{u} \mid \bigcap_{a_{i} \in A^{\prime \prime}} E_{i}\right)$ can be written as

$$
\begin{aligned}
& \sum_{B^{\prime} \subseteq B / B_{k-1}} P\left(E_{u} \mid\left(\bigcap_{a_{i} \in A^{\prime \prime}} E_{i}\right) \cap\left(B^{\prime}=g^{-1}\left(A^{\prime \prime}\right)\right)\right) \cdot P\left(B^{\prime}=g^{-1}\left(A^{\prime \prime}\right)\right) \\
= & \sum_{B^{\prime} \subseteq B / B_{k-1}} P\left(a_{u} \in g\left(B \backslash\left(B^{\prime} \cup B_{k-1}\right)\right)\right) \cdot P\left(B^{\prime}=g^{-1}\left(A^{\prime \prime}\right)\right) \\
\leq & \sum_{B^{\prime} \subseteq B / B_{k-1}} P\left(a_{u} \in g\left(B \backslash B_{k-1}\right)\right) \cdot P\left(B^{\prime}=g^{-1}\left(A^{\prime \prime}\right)\right) \\
\leq & P\left(E_{u}\right) .
\end{aligned}
$$

This implies that for all $A^{\prime} \subseteq A$, we have $P\left(\bigcap_{a_{i} \in A^{\prime}} E_{i}\right) \leq \prod_{a_{i} \in A^{\prime}} P\left(E_{i}\right)$ and therefore,

$$
\begin{equation*}
P\left(\bigcap_{a_{i} \in A^{\prime}} a_{i} \notin A_{k}\right) \leq \prod_{a_{i} \in A^{\prime}} P\left(a_{i} \notin A_{k}\right) . \tag{3}
\end{equation*}
$$

Based on these observations, $\beta_{j}^{(k)}=P\left(b_{j} \in B_{k}\right)$ can be bounded as follows:

$$
\beta_{j}^{(k)}=1-\sum_{A^{\prime} \subseteq A} P\left(f^{-1}\left(b_{j}\right)=A^{\prime}\right) \cdot P\left(\bigcap_{a_{i} \in A^{\prime}} a_{i} \notin A_{k}\right)
$$

by (2) and (3), we have

$$
\geq 1-\sum_{A^{\prime} \subseteq A}\left(\prod_{a_{i} \in A^{\prime}} s_{i j}\right) \cdot\left(\prod_{a_{i} \notin A^{\prime}}\left(1-s_{i j}\right)\right) \cdot\left(\prod_{a_{i} \in A^{\prime}} P\left(a_{i} \notin A_{k}\right)\right)
$$

then by Lemma 1

$$
\begin{aligned}
& =1-\prod_{a_{i} \in A}\left(1-s_{i j}+s_{i j} \cdot P\left(a_{i} \notin A_{k}\right)\right) \\
& =1-\prod_{a_{i} \in A}\left(1-s_{i j} \cdot P\left(a_{i} \in A_{k}\right)\right) \\
& =1-\prod_{a_{i} \in A}\left(1-s_{i j} \cdot \alpha_{i}^{(k)}\right)
\end{aligned}
$$

and by the arithmetic geometric inequality,

$$
\begin{aligned}
& \geq 1-\left(1-\frac{\left(\sum_{i \in A} s_{i j} \cdot \alpha_{i}^{(k)}\right)}{n}\right)^{n} \\
& \geq 1-e^{-\left(\sum_{i \in A} s_{i j} \cdot \alpha_{i}^{(k)}\right)}
\end{aligned}
$$

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

$$
\alpha_{i}^{(k)} \leq e^{-\left(1-\sum_{j=1}^{n} s_{i j} \cdot \beta_{j}^{(k-1)}\right)}
$$

Proof. The proof will follow the previous one. We write $\alpha_{i}^{(k)}$, the probability that $a_{i}$ belongs to $A_{k}$, as

$$
\sum_{B^{\prime} \subseteq B} P\left(g^{-1}\left(a_{i}\right)=B^{\prime}\right) \cdot P\left(\bigcap_{b_{j} \in B^{\prime}} b_{j} \in B_{k-1}\right)
$$

where the first term of the summation is

$$
\begin{equation*}
P\left(g^{-1}\left(a_{i}\right)=B^{\prime}\right)=\left(\prod_{b_{j} \in B^{\prime}} s_{i j}\right) \cdot\left(\prod_{b_{j} \notin B^{\prime}} 1-s_{i j}\right) \tag{4}
\end{equation*}
$$

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 $\left|B^{\prime}\right|$ that $P\left(\bigcap_{b_{j} \in B^{\prime}} F_{j}\right) \leq$ $\prod_{b_{j} \in B^{\prime}} P\left(F_{j}\right)$. For $\left|B^{\prime}\right|=1$, the (in)equality is evident; assume that it holds also for all $B^{\prime}$ with $\left|B^{\prime}\right|<\ell$. Let $B^{\prime}$ be a subset of $B$ of size $\ell$ having the vertex $b_{u}$. Let $B^{\prime \prime}=B^{\prime} / b_{u}$. By Bayes law and the induction assumption, we have:

$$
P\left(\bigcap_{b_{j} \in B^{\prime}} F_{j}\right)=P\left(\bigcap_{b_{j} \in B^{\prime \prime}} F_{j}\right) \cdot P\left(F_{u} \mid \bigcap_{j \in B^{\prime \prime}} F_{j}\right) \leq \prod_{b_{j} \in B^{\prime \prime}} P\left(F_{j}\right) \cdot P\left(F_{u} \mid \bigcap_{j \in B^{\prime \prime}} F_{j}\right)
$$

We now prove that $P\left(F_{u} \mid \bigcap_{j \in B^{\prime \prime}} F_{j}\right) \leq P\left(F_{u}\right)$. 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, $P\left(F_{u} \mid \bigcap_{j \in B^{\prime \prime}} F_{j}\right)$ can be written as

$$
\begin{aligned}
& \sum_{A^{\prime} \subseteq A_{k}} P\left(F_{u} \mid\left(\bigcap_{b_{j} \in B^{\prime \prime}} F_{j}\right) \cap\left(A^{\prime}=f^{-1}\left(B^{\prime \prime}\right) \cap A_{k}\right)\right) \cdot P\left(A^{\prime}=f^{-1}\left(B^{\prime \prime}\right) \cap A_{k}\right) \\
\leq & \sum_{A^{\prime} \subseteq A_{k}} P\left(b_{u} \in f\left(A_{k} / A^{\prime}\right)\right) \cdot P\left(A^{\prime}=f^{-1}\left(B^{\prime \prime}\right) \cap A_{k}\right) \\
\leq & \sum_{A^{\prime} \subseteq A_{k}} P\left(b_{u} \in f\left(A_{k}\right)\right) \cdot P\left(A^{\prime}=f^{-1}\left(B^{\prime \prime}\right) \cap A_{k}\right) \\
\leq & P\left(F_{u}\right) .
\end{aligned}
$$

This implies that for all $B^{\prime} \subseteq B$, we have $P\left(\bigcap_{b_{j} \in B^{\prime}} F_{j}\right) \leq \prod_{b_{j} \in B^{\prime}} P\left(F_{j}\right)$. Therefore,

$$
\begin{equation*}
P\left(\bigcap_{b_{j} \in B^{\prime}} b_{j} \in B_{k-1}\right) \leq \prod_{b_{j} \in B^{\prime}} P\left(b_{j} \in B_{k-1}\right) \tag{5}
\end{equation*}
$$

Based on these observations, $\alpha_{i}^{(k)}=P\left(a_{i} \in A_{k}\right)$ can be upper bounded as
follows:

$$
\begin{aligned}
\alpha_{i}^{(k)} & =\sum_{B^{\prime} \subseteq B} P\left(g^{-1}\left(a_{i}\right)=B^{\prime}\right) \cdot P\left(B^{\prime} \subseteq B_{k-1}\right) \\
& \text { by (4) and (5), we have } \\
& \leq \sum_{B^{\prime} \subseteq B}\left(\prod_{b_{j} \in B^{\prime}} s_{i j}\right) \cdot\left(\prod_{b_{j} \notin B^{\prime}}\left(1-s_{i j}\right)\right) \cdot\left(\prod_{b_{j} \in B^{\prime}} P\left(b_{j} \in B_{k-1}\right)\right)
\end{aligned}
$$

then by Lemma 1

$$
=\prod_{j \in B}\left(1-s_{i j}+s_{i j} \cdot P\left(b_{j} \in B_{k-1}\right)\right)
$$

and by the arithmetic geometric inequality

$$
\begin{aligned}
& \leq\left(1-\frac{\left(\sum_{b_{j} \in B} s_{i j} \cdot\left(1-\beta_{j}^{(k)}\right)\right)}{n}\right)^{n} \\
& \leq e^{-\left(1-\sum_{b_{j} \in B} s_{i j} \cdot \beta_{j}^{(k)}\right)}
\end{aligned}
$$

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 $\alpha_{i}$ and $\beta_{j}$, respectively. Let $\alpha$ and $\beta$ be the vectors of $\alpha_{i}$ s and $\beta_{j}$ s, i.e.,

$$
\begin{aligned}
\alpha=\left(\alpha_{1}, \alpha_{2}, \ldots, \alpha_{n}\right) & \in[0,1]^{n} \\
\beta=\left(\beta_{1}, \beta_{2}, \ldots, \beta_{n}\right) & \in[0,1]^{n}
\end{aligned}
$$

where by Theorems 6 and 7

$$
\begin{array}{ll}
\alpha_{i} \leq e^{-\left(1-\sum_{1 \leq j \leq n} s_{i j} \cdot \beta_{j}\right)} & \text { for all } 1 \leq i \leq n \\
\beta_{j} \geq 1-e^{-\sum_{1 \leq i \leq n} s_{i j} \cdot \alpha_{i}} & \text { for all } 1 \leq j \leq n
\end{array}
$$

We now prove that for $2 n$ values respecting these properties, $\sum_{i} \alpha_{i}-\sum_{j} \beta_{j} \leq$ $n \cdot(2 \Omega-1)$, where $\Omega$ equals to $W(1)$ of Lambert's $W$ function $\left(\Omega=e^{-\Omega}\right)$. 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 $X=\left(x_{i}\right)_{1 \leq i \leq n} \in[0,1]^{n}, Y=\left(y_{j}\right)_{1 \leq j \leq n} \in[0,1]^{n}$ such that:

$$
\begin{align*}
& x_{i} \leq e^{-\left(1-\sum_{1 \leq j \leq n} s_{i j} \cdot y_{j}\right)} .  \tag{6}\\
& y_{j} \geq 1-e^{-\sum_{1 \leq i \leq n} s_{i j} \cdot x_{i}} \tag{7}
\end{align*}
$$

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 $\Omega$ is the unique solution of $\Omega e^{\Omega}=1$.
. Let $\left(X_{\Omega}, Y_{\Omega}\right)$ denote the solution where $x_{i}=\Omega$ and $y_{j}=1-\Omega$ for all $i$ and $j$. The solution $\left(X_{\Omega}, Y_{\Omega}\right)$ 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 $(X, Y) \neq\left(X_{\Omega}, Y_{\Omega}\right)$ and investigate three 70 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 $\left(X_{\Omega}, Y_{\Omega}\right)$.
${ }_{475}$ 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 $\left(X^{\prime}, Y^{\prime}\right)$ be defined as follows: $x_{i}^{\prime}=\frac{x_{i}-(1-p) \cdot \Omega}{p}$ and $y_{j}^{\prime}=\frac{y_{j}-(1-p) \cdot(1-\Omega)}{p}$ for any $p \in \mathbb{R}$ with $0<p<1$. In other words,

$$
\begin{align*}
x_{i} & =p \cdot x_{i}^{\prime}+(1-p) \cdot \Omega  \tag{8}\\
y_{j} & =p \cdot y_{j}^{\prime}+(1-p) \cdot(1-\Omega) \tag{9}
\end{align*}
$$

Then from (6),

$$
\begin{aligned}
x_{i} & \leq e^{-\left(1-\sum_{1 \leq j \leq n} s_{i j} \cdot y_{j}\right)} \\
p \cdot x_{i}^{\prime}+(1-p) \cdot \Omega & \leq e^{-\left(1-\sum_{1 \leq j \leq n} s_{i j} \cdot\left(p \cdot y_{j}^{\prime}+(1-p) \cdot(1-\Omega)\right)\right)} \\
& \text { and by the convexity of the function } x \rightarrow e^{-(1-x)} \\
& \leq p \cdot e^{-\left(1-\sum_{1 \leq j \leq n} s_{i j} \cdot y_{j}^{\prime}\right)}+(1-p) \cdot \Omega \\
x_{i}^{\prime} & \leq e^{-\left(1-\sum_{1 \leq j \leq n} s_{i j} \cdot y_{j}^{\prime}\right)}
\end{aligned}
$$

The same way, from (7),

$$
\begin{aligned}
y_{j} & \geq 1-e^{-\sum_{1 \leq i \leq n} s_{i j} \cdot x_{i}} \\
p \cdot y_{j}^{\prime}+(1-p) \cdot(1-\Omega) & \geq 1-e^{-\sum_{1 \leq i \leq n} s_{i j} \cdot\left(p \cdot x_{i}^{\prime}+(1-p) \cdot \Omega\right)} \\
& \text { and by the concavity of the function } x \rightarrow 1-e^{-x}, \\
& \geq p \cdot\left(1-e^{-\sum_{1 \leq i \leq n} s_{i j} \cdot x_{i}^{\prime}}\right)+(1-p) \cdot(1-\Omega) \\
y_{j}^{\prime} & \geq 1-e^{-\sum_{1 \leq i \leq n} s_{i j} \cdot x_{i}^{\prime}} .
\end{aligned}
$$

Thus, $\left(X^{\prime}, Y^{\prime}\right)$ satisfies (6) and (7).
We now define a $p$ with the property that $0<p<1$ which leads to $0 \leq x_{i}^{\prime}, y_{j}^{\prime} \leq 1$. Let $m_{x}=\min _{i}\left\{x_{i}\right\}, m_{y}=\min _{j}\left\{y_{j}\right\}, M_{x}=\max _{i}\left\{x_{i}\right\}$ and $M_{y}=\max _{j}\left\{y_{j}\right\}$. Let $p=\max \left\{\frac{M_{x}-\Omega}{1-\Omega}, \frac{\Omega-m_{x}}{\Omega}, \frac{M_{y}-1+\Omega}{\Omega}, \frac{1-\Omega-m_{y}}{1-\Omega}\right\}$. Since, $0<x_{i}, y_{j}<1$, we have $0<p<1$. We note that by definition $p \cdot\left(x_{i}^{\prime}-\Omega\right)=x_{i}-\Omega$ and $p \cdot\left(y_{j}^{\prime}-1+\Omega\right)=y_{j}-1+\Omega$. We now analyze $x_{i}^{\prime}$. If $x_{i}^{\prime} \geq \Omega$,

$$
\begin{aligned}
p \cdot\left(x_{i}^{\prime}-\Omega\right) & =x_{i}-\Omega \\
\left(x_{i}^{\prime}-\Omega\right) & =\frac{x_{i}-\Omega}{p} \\
x_{i}^{\prime}-\Omega & \leq \frac{1-\Omega}{M_{x}-\Omega} \cdot\left(x_{i}-\Omega\right) \\
& \leq 1-\Omega \\
x_{i}^{\prime} & \leq 1
\end{aligned}
$$

and the same way, if $x_{i}^{\prime} \leq \Omega$,

$$
\begin{aligned}
p \cdot\left(x_{i}^{\prime}-\Omega\right) & =x_{i}-\Omega \\
x_{i}^{\prime}-\Omega & \geq \frac{\Omega}{\Omega-m_{x}} \cdot\left(x_{i}-\Omega\right) \\
& \geq-\Omega \\
x_{i}^{\prime} & \geq 0
\end{aligned}
$$

Hence, $0 \leq x_{i}^{\prime} \leq 1$. The same can be shown for $y_{j}^{\prime}$ by using the other two inequalities for $p$. Thus, $\left(X^{\prime}, Y^{\prime}\right)$ is a feasible solution.
We now show that $\sum_{i} x_{i}^{\prime}-\sum_{j} y_{j}^{\prime}>\sum_{i} x_{i}-\sum_{j} y_{j}$, contradicting the optimality of ( $X, Y$ ). By Equations (8) and (9), we have
$\sum\left(p \cdot x_{i}^{\prime}+(1-p) \cdot \Omega\right)=\sum x_{i}$ and $\sum\left(y_{j}^{\prime}+(1-p) \cdot(1-\Omega)\right)=\sum y_{j}$.
Therefore,

$$
\begin{equation*}
p \cdot\left(\sum x_{i}^{\prime}-\sum y_{j}^{\prime}-n \cdot(2 \Omega-1)\right)=\sum x_{i}-\sum y_{j}-n \cdot(2 \Omega-1)>0 . \tag{10}
\end{equation*}
$$

Hence,

$$
\begin{aligned}
\sum_{i} x_{i}^{\prime}-\sum_{j} y_{j}^{\prime} & =\sum x_{i}^{\prime}-\sum y_{j}^{\prime}-n \cdot(2 \Omega-1)+n \cdot(2 \Omega-1) \\
& >p \cdot\left(\sum x_{i}^{\prime}-\sum y_{j}^{\prime}-n \cdot(2 \Omega-1)\right)+n \cdot(2 \Omega-1) \\
& \text { by using Equation } 10 \text { and its analogue for } y_{j}^{\prime} \\
& =\sum x_{i}-\sum y_{j}
\end{aligned}
$$

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 $\left(X^{\prime}, Y^{\prime}\right)$ such that $\sum x_{i}^{\prime}-\sum y_{j}^{\prime}>\sum x_{i}-\sum y_{j}$, and hence $(X, Y)$ cannot be optimal. But if 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 $s_{i j}>0$.
Suppose that $x_{i}=0$. Let $x_{i}^{\prime \prime}=e^{-1}$, and $x_{k}^{\prime \prime}=x_{k}$ for all $k \neq i$ and let $y_{j}^{\prime \prime}=1-\left(1-y_{j}\right) \cdot\left(1-s_{i j} \cdot e^{-1}\right)$ for all $j$. Note that $x_{i}^{\prime \prime}=e^{-1} \leq e^{-\left(1-\sum_{i} s_{i j} \cdot y_{j}^{\prime \prime}\right)}$ and $y_{j}^{\prime \prime} \geq y_{j}$. Then,

$$
\begin{aligned}
y_{j}^{\prime \prime} & =1-\left(1-y_{j}\right) \cdot\left(1-s_{i j} \cdot e^{-1}\right) \\
& \geq 1-e^{-\sum_{i} s_{i j} \cdot x_{i}}\left(1-s_{i j} \cdot e^{-1}\right) \\
& \geq 1-e^{-\sum_{i} s_{i j} \cdot x_{i}} e^{-s_{i j} \cdot e^{-1}} \\
& \geq 1-e^{-\sum_{i} s_{i j} \cdot x_{i}^{\prime \prime}}
\end{aligned}
$$

and for $k \neq i$,

$$
\begin{aligned}
x_{k}^{\prime \prime} & =x_{k} \\
& \leq e^{-\left(1-\sum_{j} s_{k j} \cdot y_{j}\right)} \\
& \leq e^{-\left(1-\sum_{j} s_{k j} \cdot y_{j}^{\prime \prime}\right)}
\end{aligned}
$$

Then, $\left(X^{\prime \prime}, Y^{\prime \prime}\right)$ is a solution, and

$$
\sum_{i} x_{i}^{\prime \prime}-y_{i}^{\prime \prime}=\sum_{i} x_{i}-y_{i}+e^{-1}\left(1-\sum_{j} s_{i j} \cdot\left(1-y_{j}\right)\right)
$$

### 3.3. Further discussions

The Sinkhorn-Knopp scaling algorithm converges linearly (when A has total support) where the convergence rate is equivalent to the square of the second 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 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 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

$$
\mathbf{A}=\left(\begin{array}{ccc}
H & * & * \\
O & S & * \\
O & O & V
\end{array}\right), \quad S=\left(\begin{array}{cc}
S_{1} & * \\
O & S_{2}
\end{array}\right)
$$

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 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 2.00 Ghz and 256 GB of memory split across 50 the two NUMA domains. Each CPU has eight-cores (16 cores in total) and HyperThreading is enabled. Each core has its own 32 kB L1 cache and 256 kB L2 cache. The 8 cores on a CPU share a 20 MB 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 -02 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 oper595 ation 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 600 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.
605 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.

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.

### 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 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 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.

### 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 A's first $n / 2$ rows, and $R_{2}$ be the set of 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 $n / 2$ 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, 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 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.

|  |  |  | Results with different number of scaling iterations |  |  |  |  |  |
| ---: | ---: | :---: | ---: | :--- | :--- | :--- | :--- | :--- |
|  | KARP | 0 | 1 |  | 5 |  | 10 |  |
| $k$ | SIPSER | 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 |

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 $k \in\{2,4,8,16,32\}$.

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 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 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 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 | $\begin{aligned} & \text { One } \\ & \text { Sided } \\ & \text { Match } \end{aligned}$ | Two Sided Match | $d$ | iter | sprank | One <br> Sided <br> Match | Two Sided Match |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
| Square $100000 \times 100000$ |  |  |  |  |  |  |  |  |  |
| 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 $100000 \times 120000$ |  |  |  |  |  |  |  |  |  |
| 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

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 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, 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 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.

|  |  |  |  | Avg. |  |  |
| :--- | ---: | ---: | ---: | ---: | ---: | ---: |
| Name | $n$ | \# of edges | deg. | std $_{A}$ | std $_{B}$ | $\frac{\text { sprank }}{n}$ |
| 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, $s t d_{A}$ and $s t d_{B}$ are the standard deviations of $A$ and $B$ vertex (row and column) degrees, respectively.

### 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, respectively. 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 ignored. 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 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 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 their degree distributions. As Figure 3(b) shows, OneSidedMatch obtains its worst speedups on these matrices. This is not a coincidence; when the coefficient


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

| Name | Scaling error (number of iters.) |  |  | Exec. times with single thread (secs) |  |  |  |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
|  |  |  |  | ScaleSK | OneSidedMatch | $\begin{aligned} & \hline \text { KARP } \\ & \text { SIPSER } \\ & \text { MT } \end{aligned}$ | Two <br> Sided <br> Match |
|  | 1 | 5 | 10 |  |  |  |  |
| 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 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.

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 proposed 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 execution, 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 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 $2 \cdot 7$ 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 KARPSipserMT phases of TwoSidedMatch and measured the imbalance. Here we give the details of these experiments.

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, $2 n$, for the 12 matrices in Table 3 . As expected, the average number 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 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 RMAT 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 used, the standard deviations $s t d_{A}$ and $s t d_{B}$ are computed as 183.5 and 99.5. In this experiment with 20 repetitions, the average ratio of the out-one paths to $2 n$ 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 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 $6(\mathrm{~b})$ 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 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 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) 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 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


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.

| Name | $n$ | \# of edges | Degree |  |  | Speedup |  |  |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
|  |  |  | 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


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.

| Name | Run time in seconds |  |  |  |  |  |  |  |  |  |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
|  | One thread |  |  |  |  | 16 threads |  |  |  |  |
|  | 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]. 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

## 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, 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-

## 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 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 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) 13:113:31.
[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 (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 online 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 Symposium (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) 111177.
[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, SpringerVerlag 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 ( $1+$ $\left.e^{-1}\right)$-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


[^0]:    * Corresponding author

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

