Bipartite Matching Heuristics with Quality Guarantees on Shared Memory Parallel Computers

Bora Uçar

CNRS & ENS Lyon, France

9th Scheduling for Large Scale Systems, Lyon, France

Joint work with:

Fanny Dufossé
Inria, Lille,
France

Kamer Kaya
Sabancı University
Istanbul, Turkey

Motivation and context

- Bipartite matching is important in many real life applications.
- A whole family of practical, exact algorithms start with a fast heuristic to obtain a large set of initial matching.
- Some applications are ok with a suboptimal matching (see [Lotker et al., SPAA'08], citing routers)

- We propose heuristics guided by parallelism and designed for parallel implementation.
- Good theoretical and demonstrated approximation guarantee.
- We focus on bipartite graphs corresponding to $n \times n$ sparse matrices.
- If a maximum matching is sought, not an exact answer.

Some recent work

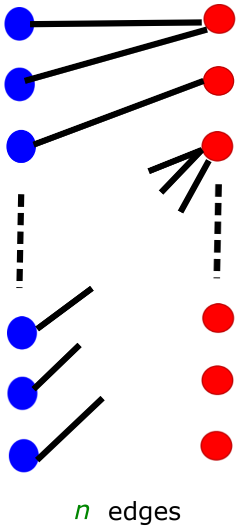
- Lotker, Patt-Shamir, Pettie, SPAA'08: $(1 - 1/k)$ approximate matching in $O(k^3 \log \Delta + k^2 \log n)$ time steps with message length of $O(\log \Delta)$ bits (for distributed memory).
- Blelloch, Fineman, Shun, SPAA'12: maximal matching, $O(m)$ work $O(\log^3 m)$ depth algorithm, whp, on a bipartite graph with m edges. Shared memory implementation.
- Birn, Osipov, Sanders, Schulz, Sitchinava, Euro-Par 2013: $1/2$ approximate, maximal matching in $O(\log^2 n)$ time and $O(m)$ work, implementation in distributed memory, and GPUs.
- Azad, Halappanavar, Rajamanickam, Boman, Khan, Pothen, IPDPS'12: heuristics and exact algorithms on shared memory.
- Deveci, Kaya, U., Çatalyürek, ICPP'13, Euro-Par 2013: exact algorithms on GPUs.
- Langguth, Manne, Sanders, ACM JEA, 2010: analysis of heuristics.

Algorithm I

for $j = 1$ **to** n **do**
    $cmatch(j) \leftarrow NIL$
for $i = 1$ **to** n **do**
    Randomly pick a column j in row i (uniformly random)
    $cmatch(j) \leftarrow i$

- Some columns are picked by many rows but only one of them gets to be matched.
- Some columns are not matched at all, $cmatch(\cdot)$ value remains NIL.

Example



For each selected red vertex, we can have a mate.

Count the number of red vertices that are not selected to obtain a bound on the size of the matching.

Algorithm I – The expected size of a matching

Assumption: all rows and columns have d nonzeros (a row picks one of its columns with $1/d$). Then the probability that the column j is not picked

$$(1 - 1/d)^d \ .$$

Bounded by the limit (increasing function),

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

The expected number of unmatched columns is less than:

$$\frac{n}{e} \ .$$

Therefore, the above algorithm obtains a matching of size

$$n\left(1 - \frac{1}{e}\right) \approx n \, 0.6321 \ .$$

Algorithm II – OneSided

- The assumption that each row and column has d nonzeros is very restrictive.
- Remedy: Any nonnegative matrix $\mathbf{A}$ with a total support can be scaled to a doubly stochastic matrix with two positive diagonal matrices, yielding $\mathbf{S} = \mathbf{D}_1\mathbf{A}\mathbf{D}_2$.
- We first do a scaling of the $\{0,1\}^{n\times n}$ adjacency matrix $\mathbf{A}$ and then perform matching as before.

```
S ← doubly stochastic scaling of A
for j = 1 to n do
    cmatch(j) ← NIL
for i = 1 to n do
    Randomly pick column j, according to probabilities S_ij
    cmatch(j) ← i
```

Algorithm II – The expected size of a matching

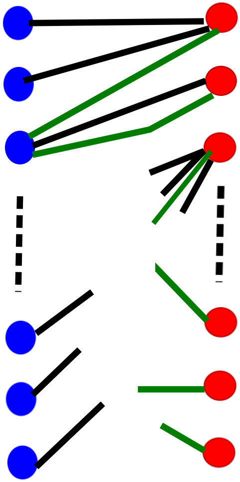
Again, the above algorithm obtains a matching of size (proof uses the arithmetic-geometric mean inequality as an additional machinery)

$$n\left(1-\frac{1}{e}\right)\approx n\,0.6321$$

Algorithm III – TwoSided

$\mathbf{S} \leftarrow$ doubly stochastic scaling of $\mathbf{A}$
$E \leftarrow \emptyset$
for $i = 1$ **to** n **do**
    Randomly pick a column j, according to probabilities S_{ij}
    $E \leftarrow E \cup \{(i, j)\}$
for $j = 1$ **to** n **do**
    Randomly pick a row i, according to probabilities S_{ij}
    $E \leftarrow E \cup \{(i, j)\}$
Run the Karp-Sipser algorithm on the edges E.

Algorithm III – TwoSided



$2n$ edges

More complicated than before. We need an exact matching algorithm.

Any matching algorithm would do, but we can do better by taking advantage of the special structure of $2n$ edges.

- Karp-Sipser heuristic: match a degree-one vertex, if none, match a random pair.
- $O(|E|)$ time complexity, matches all but $\tilde{O}(n^{1/5})$ vertices of a random undirected graph.

KS heuristics become an exact algorithm for the type of graphs that are constructed by TwoSided

Conjecture: given a bipartite graph (having perfect matchings), we can choose a spanning 1-out sub-graph that has a maximum matching of cardinality $0.866n$.

Parallelization on shared memory systems

Scaling algorithms are not our concern here;
(computations are similar to repeated SpMxV; virtually any technique used in parallelizing SpMxV can be used)

Algorithm II ONESIDED:

1: $\mathbf{S} \leftarrow$ doubly stochastic scaling of $\mathbf{A}$
2: **for** $j = 1$ **to** n **do**
3: $\quad cmatch(j) \leftarrow NIL$
4: **for** $i = 1$ **to** n **do**
5: $\quad$ Randomly pick column j, according to probabilities S_{ij}
6: $\quad cmatch(j) \leftarrow i$

Split the rows among the threads with a **parallel for** construct. No synchronization or conflict detection.

Assumption about the computer: one write survives, in case of concurrent writes to the same memory location.

Parallelization on shared memory systems

Algorithm III: TwoSided:

1: $\mathbf{S} \leftarrow$ doubly stochastic scaling of $\mathbf{A}$
2: $E \leftarrow \emptyset$
3: **for** $i = 1$ **to** n **do**
4: Randomly pick a column j
5: $E \leftarrow E \cup \{(i,j)\}$
6: **for** $j = 1$ **to** n **do**
7: Randomly pick a row i
8: $E \leftarrow E \cup \{(i,j)\}$
9: Karp-Sipser on the edges E.

With a standard Karp-Sipser, speedup is hard to achieve.

We exploit the structure of the graph :

- E not kept as a regular edge set
- each connected component has at most one cycle
- if degree-one vertices are handled, we end up with cycles (any remaining $\langle i, pick[i] \rangle$ is valid along a cycle)
- if one degree-one vertex is matched, then at most one degree-one vertex is created

TwoSided – appx guarantee and further notes

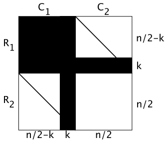
Conjecture: Let **A** be an $n \times n$ matrix with total support. Then, TwoSided obtains, asymptotically always surely, a matching of size $0.866n$.

- There are experiments supporting the conjecture.
- There is some theory too:
  Meir and Moon,'74 show: In a random 1-out graph (of n vertices on each side), the expected maximum cardinality matching is $0.866n$.

If we apply TwoSided to a square dense matrix (complete bipartite graph) we obtain a random 1-out bipartite graph.

- Need to run the scaling algorithm for only a few iterations.
- We assumed that the initial graph has perfect matchings (for analysis). This seems to be enough to show appx guarantee.
- For practical purposes, we are ok on all bipartite graphs.

Experiments I: Matching quality

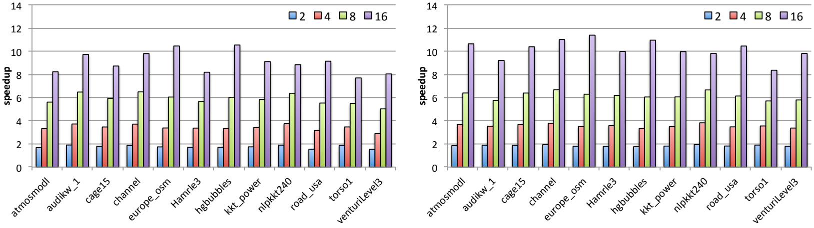
- 743 matrices from UFL: 10 iterations of scaling is enough to obtain the approximation in all but 37 matrices. They needed 10 more scaling iterations.
- 

On these graphs can beat standard Karp-Sipser heuristics dearly.
- Graphs without perfect matching: use `sprand` of Matlab (Erdös-Rényi matrices); 10 iterations of scaling was again enough to surpass the approximation guarantees.

Experiments II: Running time

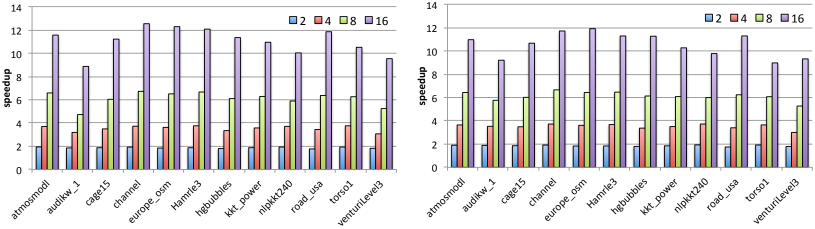
- Machine: two Intel Sandybridge-EP CPUs (each 8 cores) clocked at 2.00Ghz and 256GB of memory split across the two NUMA domains.
- With 2, 4, 8, 16 threads on a few large matrices from UFL.
- Using C and OpenMP parallelism; `gcc 4.4.5` with the `-O2` optimization flag; gcc atomic operations
- (`dynamic, 512`) OpenMP scheduling policy is employed while running all the algorithms except Karp-Sipser; it uses (`guided`).
- speed up values: against a single thread execution (geometric mean of 15/20 executions).

Experiments II: Speed up



(a) SCALESK (b) ONESIDEDMATCH

Fig. 3. Speedups for SCALESK (left) and ONESIDEDMATCH (right) with a single scaling iteration.

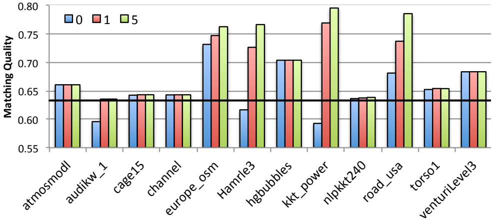


(a) KARPSIPSERMT (b) TWOSIDEDMATCH

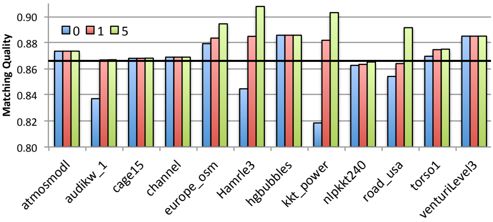
Fig. 4. Speedups for KARPSIPSERMT (left) and TWOSIDEDMATCH (right) with a single scaling iteration.

Experiments III: Quality wrt the scaling iterations

Matching quality of ONESIDED



Matching quality of TWOSIDED



The horizontal lines are at 0.632 and 0.866....

the approximation guarantees for the heuristics (conjectured for TWOSIDED).

Concluding remarks

- Two heuristics: doubly stochastic scaling + random choices.
- One heuristic obtains $1 - 1/e$ appx solutions; other is claimed to obtain 0.866 appx
- One works on n edges, other $2n$ edges.
- One has virtually no parallelization overhead, other runs a specialized Karp-Sipser.
- Speed-ups on unto 16 cores beyond 10 are realized on large bipartite graphs

---

- What about bipartite graphs corresponding to rectangular matrices?
- Extensions to undirected graphs?

Thanks!

Thank you!

http://perso.ens-lyon.fr/bora.ucar/

The sequential algorithm (Ruiz'01)

$\mathbf{D}_r^{(0)} \leftarrow \mathbf{I}_{m\times m} \qquad \mathbf{D}_c^{(0)} \leftarrow \mathbf{I}_{n\times n}$

for $k = 1, 2, \ldots$ **until** convergence **do**

$\quad \mathbf{D}_1 \leftarrow \mathrm{diag}\left(\sqrt{\|\mathbf{r}_i^{(k)}\|_\ell}\right) \; i = 1, \ldots, m$

$\quad \mathbf{D}_2 \leftarrow \mathrm{diag}\left(\sqrt{\|\mathbf{c}_j^{(k)}\|_\ell}\right) \; j = 1, \ldots, n$

$\quad \mathbf{A}^{(k+1)} \leftarrow \mathbf{D}_1^{(k+1)}\mathbf{A}\mathbf{D}_2^{(k+1)}$

$\quad \mathbf{D}_r^{(k+1)} \leftarrow \mathbf{D}_r^{(k)}\,\mathbf{D}_1^{-1}$

$\quad \mathbf{D}_c^{(k+1)} \leftarrow \mathbf{D}_c^{(k)}\,\mathbf{D}_2^{-1}$

Reminder

$\mathbf{r}_i^{(k)}$: ith row at it. k

$\|\mathbf{x}\|_\infty = \max\{|x_i|\}$

$\|\mathbf{x}\|_1 = \sum |x_i|$

Notes

ℓ: any vector norm (usually ∞- and 1-norms)

Convergence is achieved when

$$\max_{1\le i\le m}\left\{\left|1 - \|\mathbf{r}_i^{(k)}\|_\ell\right|\right\} \le \varepsilon \text{ and } \max_{1\le j\le n}\left\{\left|1 - \|\mathbf{c}_j^{(k)}\|_\ell\right|\right\} \le \varepsilon$$

Experiments : Machine specs

- Machine: two Intel Sandybridge-EP CPUs (each 8 cores) clocked at 2.00Ghz and 256GB of memory split across the two NUMA domains.
- CPUs: Each CPU has eight-cores (16 cores in total) and HyperThreading is enabled.
- Cores: Each core has its own 32kB L1 cache and 256kB L2 cache. The 8 cores on a CPU share a 20MB L3 cache.
- OS: 64-bit Debian with Linux 2.6.39-bpo.2-amd64.
- Using C and OpenMP parallelism; `gcc 4.4.5` with the `-O2` optimization flag
- (`dynamic,512`) OpenMP scheduling policy is employed while running all the algorithms except KarpSipser; it uses (`guided`).
- With 2, 4, 8, 16 threads.
- For atomic operations, gcc's built-in functions are used.