Parallel Greedy Graph Matching using an Edge Partitioning Approach

Md. Mostofa Ali Patwary*
Rob H. Bisseling** Fredrik Manne*

*Department of Informatics, University of Bergen, Norway

**Department of Mathematics, Utrecht University, the Netherlands

September 25, 2010
HLPP 2010, Baltimore, Maryland
USA

Outline

Matching

- Given a graph $G = (V, E)$.
- A Matching M is a pairing of adjacent vertices such that each vertex is matched with at most one other vertex.
- In other words, M is the set of independent edges.

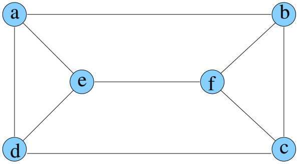


Figure: $G = (V, E)$

Matching

- Given a graph $G = (V, E)$.
- A Matching M is a pairing of adjacent vertices such that each vertex is matched with at most one other vertex.
- In other words, M is the set of independent edges.
- $|M| = 2$.

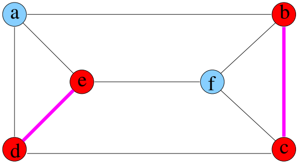


Figure: $G = (V, E)$

Matching

- Given a graph $G = (V, E)$.
- A Matching M is a pairing of adjacent vertices such that each vertex is matched with at most one other vertex.
- In other words, M is the set of independent edges.
- $|M| = 2$.
- $|M| = 3$.

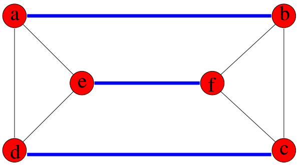


Figure: $G = (V, E)$

The Matching Problem

- Find a matching M such that
  - M has maximum cardinality.

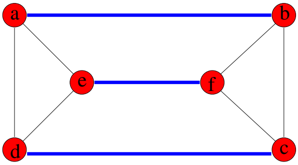


Figure: $G = (V, E)$

The Matching Problem

- Find a matching M such that
  - M has maximum cardinality.
  - Edge weight w of M is maximum for edge weighted graph.
  - $w(M) = 11$.

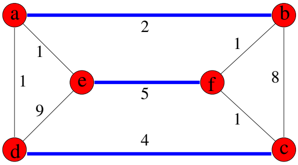


Figure: $G = (V, E)$

The Matching Problem

- Find a matching M such that
  - M has maximum cardinality.
  - Edge weight w of M is maximum for edge weighted graph.
  - $w(M) = 11$.
  - $w(M) = 17$.
- In this work we consider maximum cardinality matching.

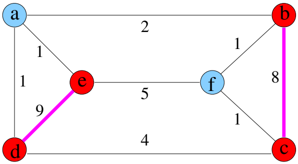


Figure: $G = (V, E)$

Applications

- Combinatorial optimization, e.g. assignment problem, stable marriage problem.
- Linear solvers, e.g. improve pivoting.
- Load balancing in parallel computation, e.g. graph partitioning.
- Bioinformatics, e.g. alignment problems.

Maximum Cardinality Matching: $G = (V, E)$

A general greedy framework:

1: $M = \emptyset$
2: **while** $E \neq \emptyset$ **do**
3: Pick the BEST remaining edge (v, w).
4: Add (v, w) to the matching M.
5: Remove all edges incident on v and w from E.

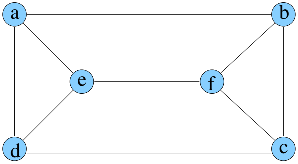


Figure: $G = (V, E)$

Maximum Cardinality Matching: Example

A general greedy framework:

$M = \emptyset$
while $E \neq \emptyset$ **do**
    Pick the BEST remaining edge (v, w).
    Add (v, w) to the matching M.
    Remove all edges incident on v and w from E.

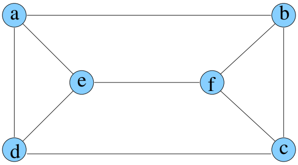


Figure: $M = \emptyset$

Maximum Cardinality Matching: Example

A general greedy framework:

1: $M = \emptyset$
2: **while** $E \neq \emptyset$ **do**
3: Pick the BEST remaining edge (v, w).
4: Add (v, w) to the matching M.
5: Remove all edges incident on v and w from E.

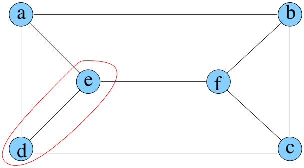


Figure: $M = (d, e)$

Maximum Cardinality Matching: Example

A general greedy framework:

1: $M = \emptyset$
2: **while** $E \neq \emptyset$ **do**
3: Pick the BEST remaining edge (v, w).
4: Add (v, w) to the matching M.
5: Remove all edges incident on v and w from E.

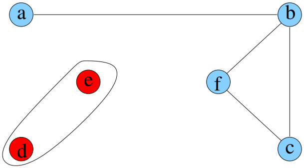


Figure: $M = (d, e)$

Maximum Cardinality Matching: Example

A general greedy framework:

1: $M = \emptyset$
2: **while** $E \neq \emptyset$ **do**
3: Pick the BEST remaining edge (v, w).
4: Add (v, w) to the matching M.
5: Remove all edges incident on v and w from E.

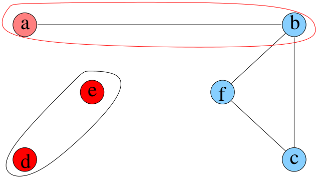


Figure: $M = (d, e); (a, b)$

Maximum Cardinality Matching: Example

A general greedy framework:

1: $M = \emptyset$
2: **while** $E \neq \emptyset$ **do**
3: Pick the BEST remaining edge (v, w).
4: Add (v, w) to the matching M.
5: Remove all edges incident on v and w from E.

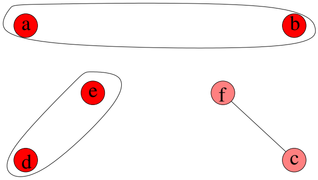


Figure: $M = (d, e); (a, b)$

Maximum Cardinality Matching: Example

A general greedy framework:

1: $M = \emptyset$
2: **while** $E \neq \emptyset$ **do**
3: Pick the BEST remaining edge (v, w).
4: Add (v, w) to the matching M.
5: Remove all edges incident on v and w from E.

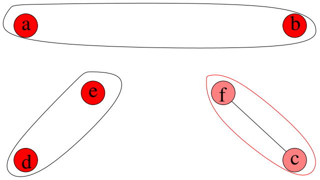


Figure: $M = (d, e); (a, b); (c, f)$

Maximum Cardinality Matching: Example

A general greedy framework:

$M = \emptyset$
while $E \neq \emptyset$ **do**
    Pick the BEST remaining edge (v, w).
    Add (v, w) to the matching M.
    Remove all edges incident on v and w from E.

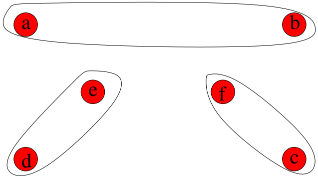


Figure: $M = (d, e); (a, b); (c, f)$

Best Edge

- HOW to choose the BEST edge of the remaining edges?
  - What should the criteria be?
- Although exact algorithms are polynomial, they could be expensive in practice.
- Therefore, the common choice is heuristics, which -
  - gives high-quality matchings in many cases.
  - is much faster for large problem sizes.
  - is easier to implement.

Heuristics - Best Edge

- Simple greedy [Möhring and Müller–Hannemann, 1995, Magun, 1998].
  - Picks an edge (v, w) where v and w are unmatched vertices.
- Static Mindegree
  - Picks the minimum degree unmatched vertex v and find a lower degree unmatched neighbour w.
- Dynamic Mindegree - Updates degree after deletion of edges.
- KARP–SIPSER algorithm - Keeps track of degree 1 vertices only + Simple greedy [Aronson et al., 1998].
  - This is the method of choice in many cases [Langguth et al., 2010].

Objectives

- Investigate the parallelization of Maximum Cardinality Matching for distributed memory computers.
- The KARP–SIPSER algorithm has been picked.
  - High quality matching quickly.

Sequential KARP–SIPSER Algorithm: Idea

- A vertex v is singleton if $d(v) = 1$.
- Idea: Match singleton vertices. If there is no singleton vertex, run simple greedy algorithm, that is, pick edges randomly.

Sequential KARP–SIPSER Algorithm: Details

$M \leftarrow \emptyset$
while $E \neq \emptyset$ **do**
    if E has singleton vertices **then**
        Pick a singleton vertex v uniformly at random.
        Let (v, w) be the only edge adjacent to v.
    else
        Pick an edge (v, w) uniformly at random.
    Add (v, w) to the matching M.
    Remove all edges incident on v and w from E.
return M

Sequential KARP–SIPSER Algorithm: Example

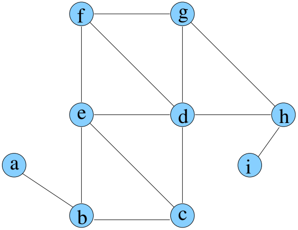


Figure: $G = (V, E)$

Sequential KARP–SIPSER Algorithm: Example

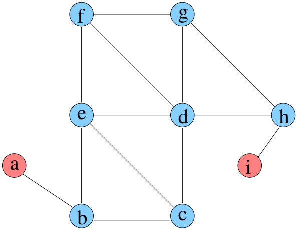


Figure: $M = \emptyset$

Sequential KARP–SIPSER Algorithm: Example

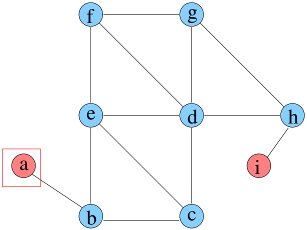


Figure: $M = \emptyset$

Sequential KARP–SIPSER Algorithm: Example

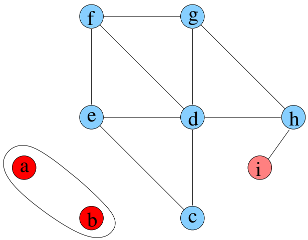


Figure: $M = (a, b)$

Sequential Karp–Sipser Algorithm: Example

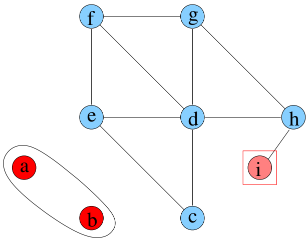


Figure: $M = (a, b)$

Sequential KARP–SIPSER Algorithm: Example

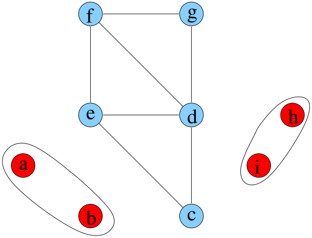


Figure: $M = (a, b); (i, h)$

Sequential KARP–SIPSER Algorithm: Example

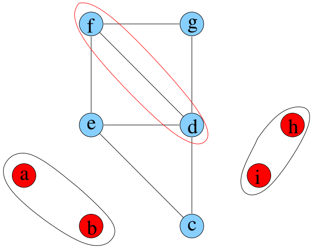


Figure: $M = (a, b); (i, h)$

Sequential KARP–SIPSER Algorithm: Example

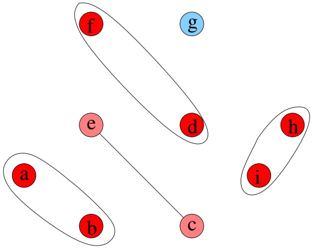


Figure: $M = (a, b); (i, h); (d, f)$

Sequential KARP–SIPSER Algorithm: Example

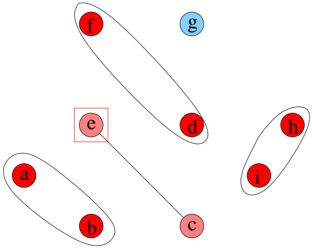


Figure: $M = (a, b); (i, h); (d, f)$

Sequential KARP–SIPSER Algorithm: Example

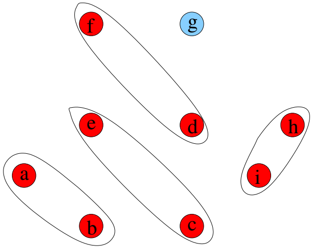


Figure: $M = (a, b); (i, h); (d, f); (c, e)$

Our Parallel Matching Algorithm

- Assume that the graph is distributed among the processors.
  - Vertex based distribution (in matrix term, 1D).
  - Edge based distribution (in matrix term, 2D).

Our Parallel Matching Algorithm: Idea

- Idea: Each processor operates in synchronized rounds (BSP).
  - Performs a local version of the sequential algorithm.
  - Communicates.
  - Processes incoming messages.
- The reason of using BSP is:
  - Enhances load balancing by detecting at an earlier stage that a processor has run out of work.
  - Takes some of the tediousness away of message-passing.
  - Many communication optimizations can be left to the system.

The Parallel Matching Algorithm: Processor P_i

while $E \neq \emptyset$ **do**
    for Pre-specified number of vertices and $E_i \neq \emptyset$ **do**
        if E_i has singleton vertices **then**
            Pick a singleton vertex v.
            Let (v, w) be the only edge adjacent to v.
        else
            Pick an edge (v, w) randomly.
        Try to match v with w.
    BSP-SYNC()
    PROCESS-MESSAGES()

The Parallel Matching Algorithm: Messages

- Original vertex (owned) and ghost vertex.
- Matching requests: Local and Non-Local.
- Confirmation back and removal of edges.

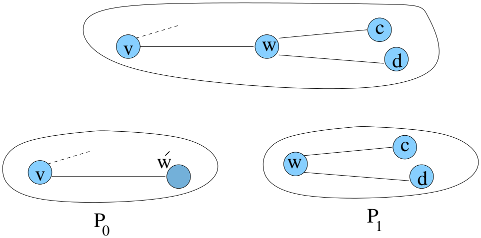


Figure: $G = (V, E)$

The Parallel Matching Algorithm: Messages

- Original vertex (owned) and ghost vertex.
- Matching requests: Local and Non-Local.
- Confirmation back and removal of edges.

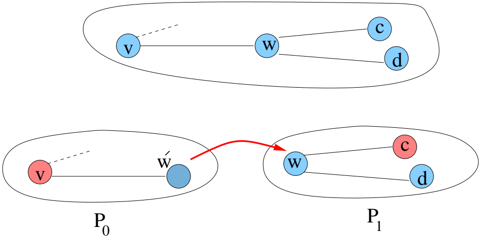


Figure: $G = (V, E)$

The Parallel Matching Algorithm: Messages

Table: Summary of message types used.

Type	Meaning
Match request	Matches a vertex v with w
Confirmation	Confirms success of matching v
Removal	Removes all edges adjacent to v
Handover	Hands over vertex v to a nonowner
Give-up	Removes a processor from $nonOwners(v)$
Criticality	Local count of vertex v became 1

Communication Volume: Upper and Lower Bounds

Parallel Matching compared to Parallel Sparse Matrix Vector Multiplication (*SpMV*):

- $\frac{1}{2} \cdot Vol(SpMV) \leq Vol(Matching) \leq \frac{3}{2} \cdot Vol(SpMV) + R$,

R represents the number of random match requests that failed during the algorithm.

Test Sets and Experimental Setup

- Huygens, an IBM pSeries 575 supercomputer, 104 nodes, each with 16 processors (IBM Power6 dual-core 4.7 GHz) and 128 GByte of memory.
- Linux, C++ using the BSPonMPI [Suijlen, 2010], IBM XL C/C++ compiler, -O3 optimization level.
- Mondriaan package [Vastenhouw and Bisseling, 2005] to distribute the graphs among the processors.

Test Sets and Experimental Setup. . .

We use 4 different type test sets.

- Set 1 (rw1-rw10): 10 real-world graphs.
- Set 2 (rw11-rw14): 4 real-world graphs.
  - Medical science, structural engineering, civil engineering, circuit simulation, DNA electrophoresis, Information Retrieval, and Automotive Industry [Davis, 1994, Koster, 1999].
- Set 3 (sw1-sw3): 3 synthetic small-world graphs.
- Set 4 (er1-er3): 3 Erdös-Rényi random graphs [Bader and Madduri, 2006].

Test Sets and Experimental Setup. . .

Table: Structural properties of the input graphs.

	$\|V\|$	$\|E\|$	Degree avg	Degree max		$\|V\|$	$\|E\|$	Degree avg	Degree max
rw1	999,999	3,995,992	3	4	rw11	281,903	3,985,272	14	38,625
rw2	1,585,478	6,075,348	3	5	rw12	16,783	9,306,644	554	14,671
rw3	52,804	10,561,406	200	2,702	rw13	683,446	13,269,352	19	83,470
rw4	2,063,494	12,964,640	6	95	rw14	343,791	26,493,322	77	434
rw5	63,838	14,085,020	220	3,422	sw1	50,000	14,112,206	282	5,096
rw6	504,855	17,084,020	33	39	sw2	75,000	24,466,808	326	6,273
rw7	503,712	36,312,630	72	842	sw3	100,000	33,727,170	337	7,989
rw8	952,203	45,570,272	47	76	er1	100,000	3,319,658	33	59
rw9	1,508,065	51,164,260	33	34	er2	150,000	6,753,302	45	76
rw10	914,898	54,553,524	59	80	er3	200,000	12,008,022	60	100

Experimental Results: Communication Volume

Table: Communication volume in 1000 words for $p = 32$.

Name	SpMV 1D	SpMV 2D	Matching 1D	Matching 2D	Name	SpMV 1D	SpMV 2D	Matching 1D	Matching 2D
rw1 (ecology2)	53	51	60	55	rw11 (Stanford)	340	141	479	234
rw2 (G3_circuit)	81	65	92	73	rw12 (gupta3)	710	44	1,305	61
rw3 (crankseg_1)	78	78	155	152	rw13 (St_Berk.)	716	448	1,152	812
rw4 (kkt_power)	118	120	106	107	rw14 (F1)	139	130	148	139
rw5 (crankseg_2)	92	90	181	171	sw1	1,007	417	2,111	303
rw6 (af_shell8)	51	47	85	65	sw2	1,957	829	3,999	563
rw7 (inline_1)	104	105	115	118	sw3	2,017	832	4,255	528
rw8 (ldoor)	131	128	140	148	er1	1,856	1,133	1,788	1,157
rw9 (af_shell10)	113	105	169	150	er2	3,451	1,841	3,721	1,635
rw10 (boneS10)	150	145	228	189	er3	5,476	2,569	6,350	1,990

- 2D takes less communication and moving from 1D to 2D gives a savings of a factor of 2 for Set 3 and 4, even larger savings for Set 2, and a modest gain in Set 1.

Experimental Results: Speedup

How many vertices, *VpR* to process per round?

Table: Speedup as a function of *VpR* for $p = 32$.

$VpR =$	100	200	400	800	1600		100	200	400	800	1600
rw1	0.67	0.74	0.62	0.40	0.24	rw11	4.25	5.32	6.15	6.17	6.45
rw2	0.66	0.72	0.59	0.38	0.20	rw12	25.36	18.99	30.55	29.55	30.35
rw3	12.65	13.07	15.13	14.53	14.42	rw13	1.18	1.59	1.83	1.85	1.73
rw4	1.55	1.30	0.72	0.31	0.17	rw14	13.15	16.67	19.54	21.63	24.23
rw5	14.11	16.62	19.69	21.09	19.99	sw1	29.49	33.38	34.63	30.58	30.82
rw6	6.26	9.29	12.92	14.03	13.82	sw2	27.87	31.16	33.85	33.91	33.75
rw7	9.19	11.17	12.09	12.85	12.88	sw3	33.35	40.83	42.18	44.64	42.43
rw8	6.93	8.45	9.22	9.25	8.83	er1	5.20	6.02	7.64	8.60	9.51
rw9	6.44	9.66	12.19	13.08	11.50	er2	7.15	9.60	11.00	12.71	13.63
rw10	7.07	8.41	8.82	7.97	6.60	er3	14.31	15.97	18.14	19.72	21.55

Speedup increases with *VpR*.

Experimental Results: Matching Quality

How many vertices, *VpR* to process per round?

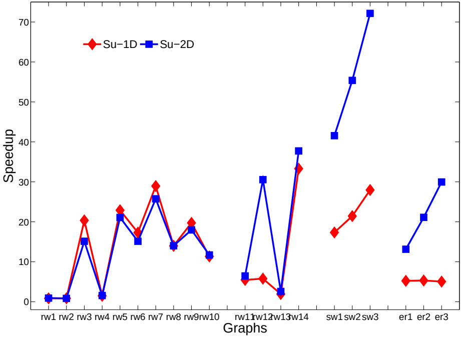
Table: Matching quality (in %) as a function of *VpR* for $p = 32$

$VpR =$	100	200	400	800	1600		100	200	400	800	1600
rw1	98.15	98.14	98.13	98.08	98.12	rw11	71.75	71.61	71.48	71.32	71.11
rw2	96.71	96.69	96.61	96.52	96.45	rw12	98.31	98.00	97.35	97.35	97.35
rw3	99.21	99.15	99.13	99.16	99.19	rw13	66.19	66.15	66.09	65.99	65.87
rw4	88.55	88.58	88.58	88.57	88.57	rw14	99.54	99.52	99.53	99.51	99.49
rw5	99.26	99.24	99.24	99.20	99.18	sw1	79.81	78.07	77.06	75.66	75.59
rw6	99.93	99.93	99.92	99.93	99.93	sw2	90.74	88.87	86.25	84.09	81.89
rw7	99.56	99.55	99.55	99.54	99.53	sw3	81.87	80.13	78.47	77.29	76.01
rw8	98.58	98.58	98.58	98.58	98.57	er1	97.50	93.45	85.67	78.69	74.13
rw9	99.94	99.94	99.94	99.94	99.94	er2	98.43	95.63	89.12	82.54	76.07
rw10	99.58	99.56	99.55	99.55	99.55	er3	95.98	93.14	88.94	83.42	77.59

The matching quality decreases with *VpR*.

Parallel Matching Algorithm: Maximum Speedup

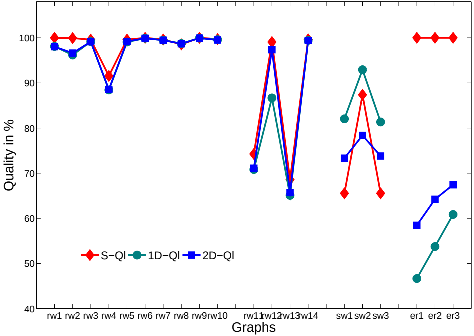
Figure: Maximum speedup obtained using 1D and 2D.



- Speedup in almost all cases (1D and 2D).
- Test Set 1 and 2 - Same speedup for 1D and 2D.
- Test Set 3 and 4 - 2D is better than 1D.

Parallel Matching Algorithm: Corresponding Quality

Figure: Matching quality in % - Sequential, 1D and 2D.



- Test Set 1 and 2 - Sequential, 1D, and 2D - same quality.
- Test Set 3 - 1D and 2D perform better than Sequential.
- Test Set 4, 2D gives better quality compared to 1D.

Conclusion

- We have parallelize a Greedy Graph Matching Algorithm for distributed memory computers.
- We have obtained good speedups for many graphs without compromising the quality of the matching.
- Edge-based partitioning (2D) gives larger scalability and better matching quality compared to vertex-based partitioning (1D).
- $\frac{1}{2} \cdot Vol(SpMV) \leq Vol(Matching) \leq \frac{3}{2} \cdot Vol(SpMV) + R$.
  In practice, the range is between 0.63 to 1.95 times $Vol(SpMV)$ for 2, 4, 8, 16, 32, and 64 processors.

Future Works

- Extend this work for Parallel Maximum Weighted Matching.
- We intend to generalize this approach for the whole class where an edge-based approach will be suitable.

Thank you.

1D and 2D

- In both 1D and 2D cases, we consider only the lower triangle and the edges are unique among the processors.
- The difference between 1D and 2D:
  - For 2D we try to divide the edges equally among the processors.
  - For 1D, we try to divide the vertices equally among the processors.
- But still for 1D case, all the edges of a vertex may not be in the same processor always.
- This way, we can view vertex partitioning as a special case of edge partitioning.
- To keep the parallel matching algorithm unchanged irrespective of partitioning, we did this modification from the conventional 1D.

Why bulk-synchronous parallel (BSP)

- BSP is characterized by alternating between computation phases and communication phases, each ended by a global barrier synchronization.
  - Enhances load balancing by detecting at an earlier stage that a processor has run out of work.
  - BSPlib communication library [Hill et al., 1998] takes some of the tediousness away of message-passing for irregular computations.
  - Many communication optimizations can be left to the system.

The Sequential KARP–SIPSER Algorithm: Analysis

- There are two phases of in the execution of the KARP–SIPSER algorithm.
  - Phase 1: Starts at the begining of the while loop and ends when the current graph has no singleton vertex.
  - Phase 2: The remainder of the algorithm.
- If M_1 is the set of vertices chosen in Phase 1, then there exists some maximum cardinality matching that contains M_1, [Aronson et al., 1998, Fact 1].
- Almost all the remaining vertices are matched by the KARP–SIPSER algorithm in the special case where G is a random graph [Aronson et al., 1998, Chebolu et al., 2008].

Bibliography I

Aronson, J., Frieze, A., and Pittel, B. G. (1998).
Maximum matchings in sparse random graphs: Karp-Sipser revisited.
Rand. Struct. Alg., 12(2):111–177.

Bader, D. A. and Madduri, K. (2006).
GTGraph: A synthetic graph generator suite.
http://www.cc.gatech.edu/~kamesh/GTgraph.

Chebolu, P., Frieze, A., and Melsted, P. (2008).
Finding a maximum matching in a sparse random graph in $O(n)$ expected time.
Proc. ICALP, pages 161 – 172.
LNCS 5125.

Bibliography II

Davis, T. A. (1994).
University of Florida sparse matrix collection.
NA Digest, 92.

Hill, J. M. D., McColl, B., Stefanescu, D. C., Goudreau, M. W., Lang, K., Rao, S. B., Suel, T., Tsantilas, T., and Bisseling, R. H. (1998).
BSPlib: The BSP programming library.
Par. Comput., 24(14):1947–1980.

Koster, J. (1999).
Parasol matrices.
http://www.parallab.uib.no/projects/.

Bibliography III

Langguth, J., Manne, F., and Sanders, P. (2010).
Heuristic initialization for bipartite matching problems.
ACM J. Exp. Alg., 15:1.3:1–1.3:22.

Magun, J. (1998).
Greeding matching algorithms, an experimental study.
ACM J. Exp. Alg., 3:6.

Möhring, R. H. and Müller–Hannemann, M. (1995).
Cardinality matching: Heuristic search for augmenting paths.
Technical Report 439, Tech. Univ. Berlin, Dept. Math.

Suijlen, W. J. (2010).
BSPonMPI: An implementation of the BSPlib standard on top of MPI, Version 0.3.

Bibliography IV

Vastenhouw, B. and Bisseling, R. H. (2005).
A two-dimensional data distribution method for parallel sparse matrix-vector multiplication.
SIAM Review, 47(1):67–95.