Towards parallel bipartite matching algorithms

Bora Uçar

CNRS and GRAAL, ENS Lyon, France

Scheduling for large-scale systems, 13–15 May 2009, Knoxville

Joint work with
Patrick R. Amestoy (ENSEEIHT-IRIT, Toulouse)
Iain S. Duff (CERFACS, Toulouse and RAL, Oxon)
Daniel Ruiz (ENSEEIHT-IRIT)

Financé par ANR

Supported by Agence Nationale de la Recherche through SOLSTICE project number ANR-06-CIS6-010.

Outline

Matching

Definitions

Given an $n \times n$ matrix **A**, find a permutation **M** such that the diagonal product of the permuted matrix, $\prod \mathrm{diag}(\mathbf{AM})$, is maximum (in magnitude) among all permutations. Assume $a_{ij} \geq 0$ and there is at least one nonzero product diagonal (full structural rank).

Alternatively, select n entries from a given matrix such that no two are in a common row and column, and their product is maximum. Also called transversal and bipartite matching.

Motivations

Our driving application is direct solvers (e.g., MUMPS [Amestoy, Duff, L'Excellent, Comput. Methods in Appl. Mech. Eng., (2000)]).

Combined with scaling can avoid many numerical difficulties in factorization and linear system solution [Duff and Pralet, SIAM SIMAX(2005)].

Current state-of-the-art

Sequential

- Polynomial time solvable; best known polynomial algorithm $O(n(\tau + n \log n))$, where $\tau = \mathrm{nnz}(\mathbf{A})$ [Fredman and Tarjan, J. ACM (1987)],
- HSL subroutine MC64 [Duff and Koster, SIAM SIMAX(1999)] provides algorithms for a family of bipartite matching problems,
- MC64 has a higher polynomial (worst case) time complexity; but behaves faster than that bound.

Parallel

- Standard algorithms use depth-first/breadth-first search; inherently sequential,
- Some newer efforts [Riedy and Demmel, PP04]; some moderate speed-ups (around 5 across 5–30 processors); slow downs too.
- $\frac{1}{2}$-approximation algorithm: [Manne and Bisseling, PPAM 2007]–Scales well up to 32 processors; [Halappanavar and Pothen]–CSCAPES Seminar, 2008.

Key points

Invariance

If **Q** and **R** are two matchings and

$$\prod \mathrm{diag}(\mathbf{AQ}) > \prod \mathrm{diag}(\mathbf{AR})$$

then

$$\prod \mathrm{diag}(\hat{\mathbf{A}}\mathbf{Q}) > \prod \mathrm{diag}(\hat{\mathbf{A}}\mathbf{R})$$

for $\hat{\mathbf{A}} = \mathbf{D}_1\mathbf{A}\mathbf{D}_2$ with $\mathbf{D}_1$ and $\mathbf{D}_2$ being diagonal matrices.

Invariance under scaling

- **Q** is optimal for **A** iff it is optimal for $\hat{\mathbf{A}}$.
- In other words, the matching that gives the maximum diagonal product is invariant under row/column scaling. Also discussed in [Olschowka and Neumaier, Linear Algebra Appl., (1996)].

Key points (cont')

Suppose we have obtained a scaled matrix $\hat{\mathbf{A}} = \mathbf{D}_1\mathbf{A}\mathbf{D}_2$ such that

- $\hat{a}_{ij} \leq 1.0$,
- all rows and columns has at least one entry equal to 1.0.

Observation

Any perfect matching $\mathbf{Q}$ with $\mathrm{diag}(\hat{\mathbf{A}}\mathbf{Q})$ consisting only entries of magnitude 1.0 is optimal.

Algorithm starts to shape up...

1: $\hat{\mathbf{A}} \leftarrow \mathrm{scale}(\mathbf{A})$
2: $\hat{\mathbf{A}}_f \leftarrow \mathrm{filter}(\hat{\mathbf{A}} = 1)$
3: **if** there exist a perfect matching in $\hat{\mathbf{A}}_f$ **then**
4: **return** the matching
5: **else**
6: ...

Matrix scaling

Definition

Given an $m \times n$ sparse matrix $\mathbf{A}$, find diagonal matrices $\mathbf{D}_1 > O$ and $\mathbf{D}_2 > O$ such that all rows and columns of the scaled matrix

$$\hat{\mathbf{A}} = \mathbf{D}_1 \mathbf{A} \mathbf{D}_2$$

have equal norm.

The sequential algorithm [Ruiz 2001]

1: $\mathbf{D}_1^{(0)} \leftarrow \mathbf{I}_{m\times m}$ $\quad$ $\mathbf{D}_2^{(0)} \leftarrow \mathbf{I}_{n\times n}$
2: **for** $k = 1, 2, \ldots$ **until** convergence **do**
3: $\quad \mathbf{D}_R \leftarrow \operatorname{diag}\left(\sqrt{\|\mathbf{r}_i^{(k)}\|_\ell}\right)\ i = 1, \ldots, m$
4: $\quad \mathbf{D}_C \leftarrow \operatorname{diag}\left(\sqrt{\|\mathbf{c}_j^{(k)}\|_\ell}\right)\ j = 1, \ldots, n$
5: $\quad \mathbf{D}_1^{(k+1)} \leftarrow \mathbf{D}_1^{(k)}\,\mathbf{D}_R^{-1}$
6: $\quad \mathbf{D}_2^{(k+1)} \leftarrow \mathbf{D}_2^{(k)}\,\mathbf{D}_C^{-1}$
7: $\quad \mathbf{A}^{(k+1)} \leftarrow \mathbf{D}_1^{(k+1)}\mathbf{A}\mathbf{D}_2^{(k+1)}$

Reminder

$\|\mathbf{x}\|_\infty = \max\{|x_i|\}$
$\|\mathbf{x}\|_1 = \sum |x_i|$

Notes

ℓ: any vector norm (usually ∞- and 1-norms)
Convergence is achieved when

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

Features

Some properties

- Preserves symmetry; permutation independent; amenable to parallelization [Amestoy, Duff, Ruiz, and U. (proc. VecPar'08)].
- In ∞-norm, linear convergence with asymptotic rate of $1/2$,
- Scaling in ∞-norm is not unique,
- With 1-norm, results are similar to those of the other well-known algorithms [Sinkhorn and Knopp, Pacific J. Math (1967)]; convergence under certain conditions.
  - If each entry lie in a perfect matching, there is a unique scaled matrix,
  - If there exists a perfect matching but not all entries can be made to be in a perfect matching, iteration converges; those kind of entries must tend to zero.

Summary of computational and communication requirements

Computations (sequential execution) per iteration

Op.	SpMxV	1-norm	∞-norm
add	$\mathrm{nnz}(\mathbf{A})$	$2 \times \mathrm{nnz}(\mathbf{A})$	0
mult	$\mathrm{nnz}(\mathbf{A})$	$2 \times \mathrm{nnz}(\mathbf{A}) + m + n$	$2 \times \mathrm{nnz}(\mathbf{A}) + m + n$
comparison	0	0	$2 \times \mathrm{nnz}(\mathbf{A})$

Communication

The communication operations both in the 1-norm and ∞-norm algorithms are the same as those in the computations

$$\mathbf{y} \leftarrow \mathbf{A}\mathbf{x}$$
$$\mathbf{x} \leftarrow \mathbf{A}^T\mathbf{y}$$

when the partitions on $\mathbf{x}$ and $\mathbf{y}$ are equal to the partitions on $\mathbf{D}_2$ and $\mathbf{D}_1$.

Parallelization results: Speed-up values

matrix	Seq. Time	Number of processors 2	4	8	16
olesnik	46.08	1,9	3.7	6.9	12.3
c-71	51.60	1.8	3.3	5.4	7.6
boyd1	70.34	1.9	3.6	6.3	10.2
twotone	74.76	1.9	3.7	7.0	11.8
lhr71	78.25	2.0	3.8	7.3	13.5
aug3dcqp	8.30	1.7	2.9	4.1	4.5
a5esindl	15.09	1.8	3.0	4.1	4.8
a2nnsnsl	20.71	1.8	3.1	4.0	4.8
a0nsdsil	20.92	1.8	3.1	4.0	4.6
blockqp1	32.55	1.9	3.4	5.5	7.4

- Averages of 10 different partitions obtained using PaToH [Çatalyürek and Aykanat, Tech.Rep (1999)],
- PC cluster with a Gigabit Ethernet switch. 16 nodes, each having Intel Pentium IV 2.6 GHz processor, 1GB RAM,

Algorithm

In the scaled matrix $\hat{a}_{ij} \leq 1.0$.

Algorithm: scaling (with ε tolerance) is efficiently performed

1: $\hat{\mathbf{A}} \leftarrow \text{scale}(\mathbf{A})$
2: $\hat{\mathbf{A}}_f \leftarrow \text{filter}(1.0 - \varepsilon \leq \hat{\mathbf{A}} \leq 1.0 + \varepsilon)$
3: **if** there exist a perfect matching in $\hat{\mathbf{A}}_f$ **then**
4: **return** the matching
5: **else**
6: ...

What remains to be done?

- Step 3 can be performed sequentially, if there is only a little number of nonzeros in the filtered matrix $\hat{\mathbf{A}}_f$.
- the "**else**" part can be addressed in two ways:
  - Solve the problem exactly,
  - Or, find a sub-optimal solution (quickly).

Solving the "else" part exactly

New entries scaled to ≤ 1.0

Bring on new entries to the filtered matrix $\hat{\mathbf{A}}_f$ by updating the scaling factors so that we have perfect matching at the end.

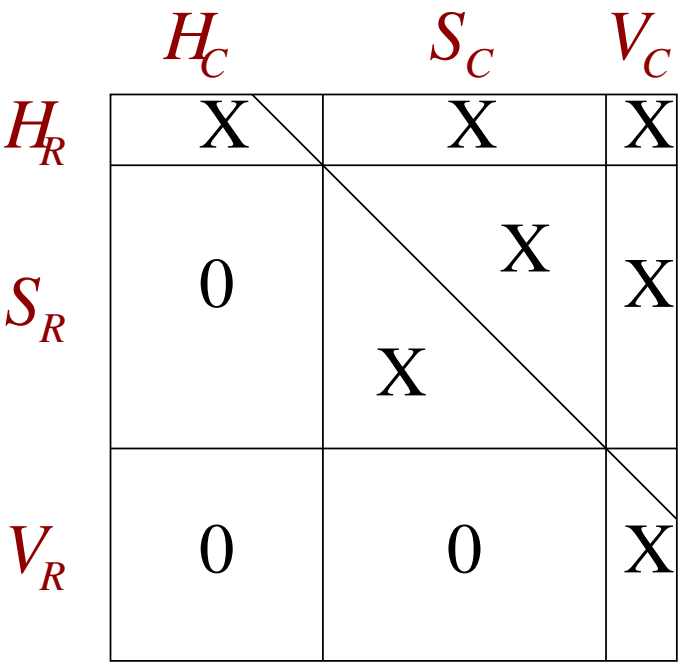
DuImage-Mendelsohn decomposition

(from [Pothen and Fan, ACM TOMS (1990)])

	H_C	S_C	V_C
H_R	X	X	X
S_R	0	X / X	X
V_R	0	0	X

- Unique Horizontal, Square, and Vertical blocks (defined by any maximum cardinality matching)
- H_Rs are perfectly matched to H_Cs,
- S_Rs are perfectly matched to S_Cs,
- V_Cs are perfectly matched to V_Rs.

Dulmage-Mendelsohn decomposition



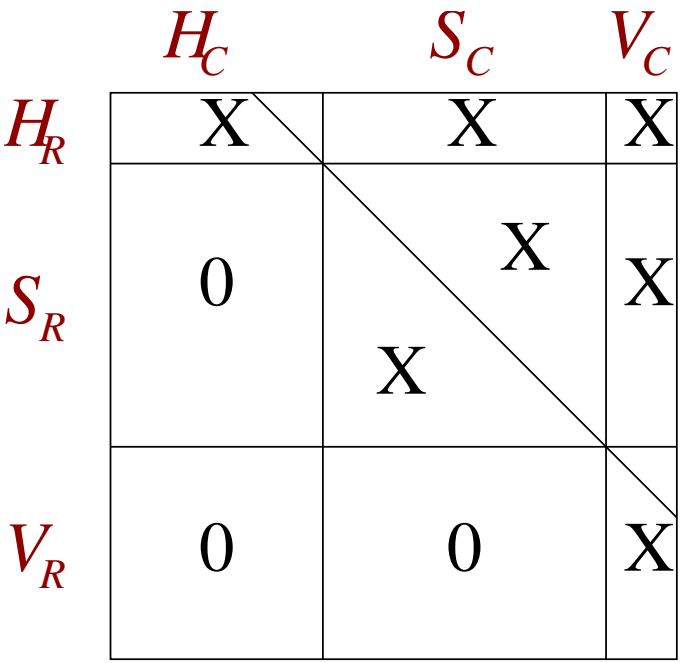
- The entries in

$$(H_R, S_C)$$
$$(H_R, V_C)$$
$$(S_R, V_C)$$

cannot be in a maximum matching.

- All maximum matchings contain entries from the three diagonal blocks.

Implications for us



- The filtered matrix $\hat{\mathbf{A}}_f$ is in this form,
- Matrix **A** (hence $\hat{\mathbf{A}}$) must have nonzeros in the blocks shown with 0,
- Find the maximum scaled entries (< 1.0) from each of those blocks,

- With a rule update the scaling matrices,
  - keep the 1s in the diagonal blocks,
  - keep everything else ≤ 1,
  - have an entry from the 0 block scaled to 1.

Implications for us

Define the three max entries

$$m_1 = \sqrt{\max \hat{\mathbf{A}}(V_R, H_C)}$$
$$m_2 = \max \hat{\mathbf{A}}(S_R, H_C)$$
$$m_3 = \max \hat{\mathbf{A}}(V_R, S_C)$$

Set the updates

If m_1 is largest then

$$\alpha = \frac{1}{m_1}$$

else if m_2 is the largest then $\alpha = \frac{1}{m_2}$

$$\vdots$$

		α H_C	1 S_C	$1/\alpha$ V_C
$1/\alpha$	H_R	=1	=1	=1
1	S_R	<1	=1	=1
α	V_R	<1	<1	=1

$\Longrightarrow$

=1	<1	<1
<=1	=1	<1
=1	<=1	=1

Algorithm exposed

Algorithm

1: $\hat{\mathbf{A}}_1 \leftarrow$ scale-1-norm($\mathbf{A}$)
2: $\hat{\mathbf{A}} \leftarrow$ scale-∞-norm($\hat{\mathbf{A}}_1$)
3: $\hat{\mathbf{A}}_f \leftarrow$ filter($1.0 - \varepsilon \leq \hat{\mathbf{A}} \leq 1.0 + \varepsilon$)
4: **if** there exist a perfect matching in $\hat{\mathbf{A}}_f$ **then**
5: **return** the matching
6: **else**
7: Compute the dmperm of $\hat{\mathbf{A}}_f$
8: **for** $k = 1, 2, \ldots$ **do**
9: Scale a particular entry in $\hat{\mathbf{A}}$ to $1.0 \pm \varepsilon$
10: Update dm-structure and scaling matrices
11: **if** perfect matching exists **then**
12: **return** the matching

Reminder

- In 1-norm scaling, any entry not in a perfect matching tends to zero,
- 1-norm scaling is unique; ∞-norm is not,

Experiments (Looking for an exact solution)

- Matrices from University of Florida sparse matrix collection, satisfying the following properties
  - Square, with $1000 \leq n < nnz \leq 2.0e{+}6$,
  - total support (no nonzeros in off diagonal blocks of the dmperm),
  - no explicit zeros, real, not $\{0, 1, -1\}^{n \times n}$.

  A total of 276 matrices. 8 required special attention; excluding those 268. 192 are symmetric and 76 are unsymmetric.

Fast solutions

In 180 matrices, no iterations after the initial 1-norm (at most 40 iterations) and ∞-norm (at most 20 iterations) scaling steps with $\varepsilon = 1.0e\text{-}3$ (126/192 symmetric; 54/76 unsymmetric).

Experiments (Looking for an exact solution)–Cont'

Algorithm: first few steps

1: $\hat{\mathbf{A}}_1 \leftarrow$ scale-1-norm($\mathbf{A}$)
2: $\hat{\mathbf{A}} \leftarrow$ scale-∞-norm($\hat{\mathbf{A}}_1$)
3: $\hat{\mathbf{A}}_f \leftarrow$ filter($1.0 - \varepsilon \leq \hat{\mathbf{A}} \leq 1.0 + \varepsilon$)
4: **if** there exist a perfect matching in $\hat{\mathbf{A}}_f$ **then**
5: **return** the matching
6: **else**
7: ...

Details of the fast solutions (among 180 matrices)

in 155, $nnz(\hat{\mathbf{A}}_f) = n$; in the rest maximum three of $nnz(\hat{\mathbf{A}}_f)/n$ are $\{5.87, 5.74, 1.03\}$

Memory requirements: $\hat{\mathbf{A}}_f$ vs $\mathbf{A}$ (of the 180 instances)

	min	avg	max
$nnz(\mathbf{A})/n$	2.25	22.05	132.36
$nnz(\hat{\mathbf{A}}_f)/n$	1.00	1.06	5.87

Experiments (Looking for an exact solution)–Cont'

Others (88/268 matrices, select and re-scale loop executed)

- averaging 8294 iterations after the initial 1-norm (at most 40 iterations) and ∞-norm (at most 20 iterations) scalings, mostly belonging to the matrix families `Schenk_IBMNA` (27 matrices), `GHS_indef` (25 matrices), and `Nemeth` (13 matrices).
- considerable savings in memory requirements

	min	avg	max
$nnz(\mathbf{A})/n$	3.22	24.97	159.03
$nnz(\hat{\mathbf{A}}_f)/n$	1.00	1.53	2.60

- However, we do not want to do iterations.
  - although very sparse, the dmperm update requires DFS/BFS-like algorithms—inherently sequential.
  - we can reduce to a single processor and solve the problem there—too much iterations.

Sub-optimal alternatives may be acceptable.

A sub-optimal solution

Algorithm

1: $\hat{\mathbf{A}}_1 \leftarrow$ scale-1-norm($\mathbf{A}$)
2: $\hat{\mathbf{A}} \leftarrow$ scale-∞-norm($\hat{\mathbf{A}}_1$)
3: $\hat{\mathbf{A}}_f \leftarrow$ filter($1.0 - \varepsilon \leq \hat{\mathbf{A}} \leq 1.0 + \varepsilon$)
4: **if** there exist a perfect matching in $\hat{\mathbf{A}}_f$ **then**
5: **return** the matching
6: **else**
7: Compute a maximum matching using only the entries in $\hat{\mathbf{A}}_f$
8: $\hat{\mathbf{A}}_w \leftarrow \hat{\mathbf{A}}_f$
9: $L \leftarrow$ sort the entries of $\hat{\mathbf{A}} - \hat{\mathbf{A}}_f$
10: **for** $k = 1, 2, \ldots$ **do**
11: add entries from L in decreasing order to $\hat{\mathbf{A}}_w$ such that all unmatched rows and columns get at most one more entry
12: if not possible, add at most one more entry per each row and column
13: Augment the matching (weighted)
14: **if** a perfect matching obtained **then**
15: **return** the matching

Experiments (sub-optimal solution)

On 88/268 matrices (solution is not obtained after the first scaling steps)

Quality of the matching

Compare $V = \sum \log \operatorname{diag}(\mathbf{AM})$ and $V^\star = \sum \log \operatorname{diag}(\mathbf{AM}^\star)$

	$(V^\star - V)/V^\star$
min	0.00
avg	0.17
max	12.15

Largest 5 values: 12.15 0.54 0.28 0.21 0.17

Iterations and memory requirements

	min	avg	max
$nnz(\mathbf{A})/n$	3.22	24.97	159.03
$nnz(\hat{\mathbf{A}}_w)/n$	2.00	2.52	3.84
iters	1	3.05	38

On 13 instances, number of augmentation iterations is greater than 3.

Summary and plans

Summary

- On 155/268 matrices, at most 40 iterations of 1-norm scaling and then at most 20 iterations of ∞-norm scaling suffices to compute a maximum product matching.
- On another 25 matrices, with a little sequential overhead an optimum matching is obtained.
- On the others (88/268): Sub-optimal solutions can be found with fairly small additional, sequential work.

On going and future work

- The effects on factorization (already done a few experiments and observed that sub-optimal solutions are not worse than the optimal ones in terms of some factorization metrics)
- Sub-optimal solutions with approximation guarantee,
- Matrices with support but without total support.

Summary and plans

So a new algorithm for an old problem?

- Not really ☹
- Follows very closely Hungarian algorithm [Kuhn, Nav. Res. Log. (1955)] for the weighted bipartite matching problem.
- Anything that forms a barrier for the efficient parallelization of the standard algorithms for the bipartite matching problems apply to our case too.

Probably some good news too ☺

- Reap the developments on Hungarian algorithm.
- Our algorithms can be used to warm-start Hungarian algorithm for the serial execution case.

Further information

Thank you for your attention.

---

http://graal.ens-lyon.fr/~bucar/

bora.ucar@ens-lyon.fr

Number of iterations with error rate of $\varepsilon = 1.0e\text{-}6$

- ∞-norm: Always converges very fast. Average 11.
- 1- and 2-norms: Did not converge on 10 and 17 matrices in 5000 iterations, respectively.
  - Average number of iterations in converged cases are 206 and 257,
  - Matrices from two groups (`GHS_indef` and `Schenk_IBMNA`) cause problems (larger number of iterations as well). 60 matrices from these groups.
  - Excluding those matrices, the averages are 26 and 29.