Parallel Weighted Bipartite Matching and Applications

E. Jason Riedy
Dr. James Demmel

SIAM Parallel Processing for Scientific Computing 2004

The problem: Maximum weight bipartite matching

Auction algorithms

Parallel auctions

Sequential improvement (was parallel performance)

Observations and the future

Max. Weight Bipartite Matching

Given:

$$\text{a bipartite graph } G = (\mathcal{R}, \mathcal{C}; \mathcal{E}) \text{ with}$$
$$\text{weights } b(i,j) \text{ for } (i,j) \in \mathcal{E}.$$

Find:

$$\text{a maximum cardinality matching } \mathcal{M}$$
$$\text{of greatest total weight } \sum_{(i,j)\in\mathcal{M}} b(i,j).$$

- Simple enough to be understood.
- Just hard enough to be interesting.
- Has actual applications. . .

Applications

- Most-likely matches between noisily-ordered strings
  - Think genes or code sequences
- Finding the most profitable connections
  - Person willing to spend \$$x$ on flight A or \$$y$ on B
- Permuting large entries to the diagonal of a sparse matrix
  - Avoid dynamic pivoting during sparse LU factorization

Driving app: Distributed SuperLU.

Goals: Distributed memory first, absolute performance second.

Linear Optimization Problem

B: the **benefit matrix** from $b(i,j)$, and
1_c, 1_r: unit-entry vectors indexed by $\mathcal{R}$ and $\mathcal{C}$
Solve for a permutation matrix X (matching $\mathcal{M}$):

$$
\begin{aligned}
&\max_X \operatorname{Tr} B^T X \\
\text{subject to } & X 1_c = 1_r, && \text{(one entry per row)} \\
& X^T 1_r = 1_c, && \text{(one entry per col)} \\
& X \geq 0.
\end{aligned}
$$

- Also known as the **linear assignment problem**.
- If $(i,j) \notin \mathcal{E}$, $b(i,j) = -\infty$; problem always feasible.
- Only gives **perfect matchings**. . .

. . . *and Its Dual Problem*

$$\begin{aligned} &\max_X \operatorname{Tr} B^T X \\ \text{subject to } & X 1_c = 1_r, \\ & X^T 1_r = 1_c, \\ & X \geq 0. \end{aligned}$$

$$\begin{aligned} &\min_{p,\pi} 1_r^T \pi + 1_c^T p \\ \text{subject to } & 1_r p^T + \pi 1_c^T \geq B. \end{aligned}$$

- $p(j)$ is a **price** for a column j, $\pi(i)$ is row i's **profit**
  - Implicitly define $\pi(i) = \max_j b(i,j) - p(j)$

. . . and Its Dual Problem

$$\begin{aligned} &\max_X \operatorname{Tr} B^T X \\ \text{subject to } &X 1_c = 1_r, \\ &X^T 1_r = 1_c, \\ &X \geq 0. \end{aligned}$$

$$\begin{aligned} &\min_{p,\pi} 1_r^T \pi + 1_c^T p \\ \text{subject to } &1_r p^T + \pi 1_c^T \geq B. \end{aligned}$$

Perfect matching X is maximum weight if there are feasible dual variables and **complementary slackness** holds:

$$x(i,j) = 1 \Rightarrow \pi(i) + p(j) = b(i,j)$$

$$X \odot (\pi 1_c^T + 1_r p^T - B) = 0$$

Standard Problem, Standard Solver?

Why not use a standard optimization solver?

Standard-form problem:

$$\begin{aligned} \min_x \; & c^T x \\ \text{s.t. } & Ax = 1_{r+c}, \text{ and} \\ & x \geq 0. \end{aligned} \qquad \begin{aligned} x &= \operatorname{vect} X, \\ c &= -\operatorname{vect} B, \\ A &= \begin{pmatrix} 1_c^T \otimes I_n \\ I_n \otimes 1_r^T \end{pmatrix} \end{aligned}$$

- Lost problem instance's structure.
- A is big and sparse, so dual matrix is big and **dense**.
- (Pre-processing for sparse LU by solving bigger, denser systems?)

Recap

Given a sparse matrix B, find a permutation X that maximizes $\operatorname{Tr} B^T X$.

Want a **distributed memory** matcher.

- Linear optimization problem with small variables
  - $n-1$ degrees of freedom for X, n entries for p
- Need to solve primal **and dual**!
- Focus on sparse, square problems.

Which Algorithm?

Combine processors' matchings via an auction. (Bertsekas, 1987)

What isn't in an auction algorithm?

- No explicit augmenting paths, no paths crossing memory boundaries.
  - (classical flow-based methods, MC64 (Duff & Koster))
- No linear solves.
  - ("Best" PRAM algorithms (Goldberg, *at al*. 1991), graph-based preconditioners (Korimort, *et al*. 2000))
- No reduction to a slightly different problem.
  - (circulations via push-relabel (Goldberg and Tarjan, 1986))
- No dense updates.
  - (Hungarian algorithm (Kuhn, Munkres, 1957))

Auction Algorithms

Basic algorithm:

1. An unmatched row i finds a "most profitable" column j
   - $\pi(i) = \max_j b(i,j) - p(i)$
2. Row i places a bid for column j.
   - Bid price raised until j is no longer the best choice. (Min. increment μ)
3. Highest bid gets the matching (i,j).

- Any interleaving will do; bids continued until all rows matched.
- Perfect match exists $\Rightarrow$ a-priori bound on highest price.

Minimum Increments and Barrier Methods

Consider a pair of rows bidding for a pair of equally valuable columns.

1. Row 1 bids for item 1 with no price increment.
2. Row 2 bids for 1 with no increment, bumping Row 1.
3. Row 1 bids for 1 with no increment, bumping Row 2.
4. ...

Minimum Increments and Barrier Methods

Consider a pair of rows bidding for a pair of equally valuable columns. Require a minimum bid increment μ.

1. Row 1 bids for item 1 with increment μ.
2. Row 2 sees higher price, bids for item 2 with increment μ.
3. Done.

Solving a Relaxed Matching Problem

Edge (i, j) is in matching only when

$$\pi(i) + (p(j) - \mu) = b(i, j).$$

Equivalently,

$$X \odot \left(\pi 1_c^T + 1_r (p - \mu 1_c)^T\right) = 0,$$

or

$$X \odot \left(\pi 1_c^T + 1_r p^T\right) = \mu 1_r 1_c^T.$$

Solving a Relaxed Matching Problem

New CS condition

$$X \odot \left(\pi 1_c^T + 1_r p^T\right) = \mu 1_r 1_c^T$$

is for a **barrier formulation** of matching:

$$\begin{aligned}&\max_X \operatorname{Tr} B^T X + \mu \operatorname{Tr}(1_r 1_c^T)^T [\log X] \\ &\text{s.t. } X 1_c = 1_r, \text{ and } X^T 1_r = 1_c.\end{aligned}$$

Within $(n-1)\mu$ of optimal value. Solve **sequence of problems** with shrinking μ.

Basic Auction Algorithm Properties

Properties to guide parallelization:

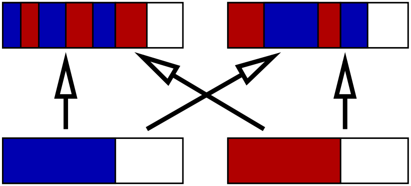
- Bids can be entered and resolved with any interleaving.
  - (Also a drawback for debugging.)
- Placing bid requires whole row.

Generally useful properties:

- Fast. ($40k \times 40k$, 1.7M entries in 5 sec. on 1.3 GHz Itanium2)
- Works for floating-point values.
  - Abs. error $\approx$ twice the worst error of evaluating primal or dual
- Works for integer values using standard double precision prices.

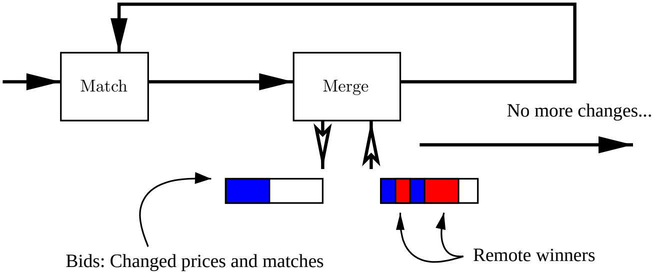
Parallelization by Distributed Bidding

- Each processor runs some local matching; prices increase.
- Local winners treated as remote bids.
- Collective "string-merge" communication.
  - Merging requires reindexing and comparisons; non-trivial.



Basic Parallel Loop

Run for each μ value:



Basic Parallel Performance...

Performed "well":

- Moderate speed-ups
  - Around 5 for many problems (`lhr` family) across 5-30 procs.
- Logarithmic slow-downs
  - Trivial matching works, still need all-to-all comm.
- (Previous drastic speed-ups were bugs.)

Most parallelism, most work in first pass over all rows for each μ.

Destroying Basic Parallel Performance

Traditionally:

- Each μ-phase begins with an empty matching.

Better:

- Each μ-phase begins with a matching satisfying its CS condition.

Requires one pass through the matrix. Reduces initial matching by factor 2-10. Reduces sequential time by at least factor of 1.5, often > 3.

Modelled New Parallel Performance

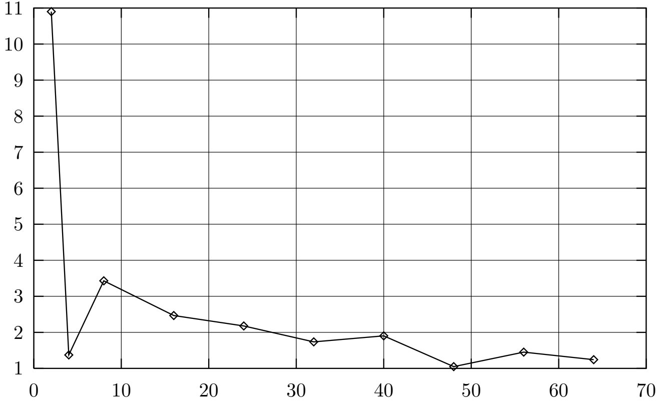
Break auction into chunks, but run and merge each chunk locally.
Assumptions:

- Longest "compute" time is longest chunk time.
  - Assume synchronized starts and no overlap of comm.
- Reduction time is (bytes sent / bandwidth + latency) $\times \log n$.

Optimistic on computation, moderately pessimistic on communication.

Modelled New Parallel Performance

1.3 GHz Itanium 2, assume gigabit rates and microsecond latency.



Observations, Future Work

Kill parallel performance by improving sequential performance.

- Need to overlap computation, communication.
  - Multi-level parallelism: One proc. works on merging while others match.
- Need better way to shrink μ.
- Estimate the tail path, migrate to one node.
- Is there an $O(|E|)$ algorithm?
  - Can **verify** a primal and dual in $O(|E|)\ldots$