Loopy Belief Propagation for Bipartite Maximum Weight *b*-Matching

Bert Huang and Tony Jebara
Computer Science Department
Columbia University
New York, NY 10027

Outline

1. Bipartite Weighted *b*-Matching
2. Edge Weights As a Distribution
3. Efficient Max-Product
4. Convergence Proof Sketch
5. Experiments
6. Discussion

Outline

1. Bipartite Weighted *b*-Matching
2. Edge Weights As a Distribution
3. Efficient Max-Product
4. Convergence Proof Sketch
5. Experiments
6. Discussion

Bipartite Weighted *b*-Matching

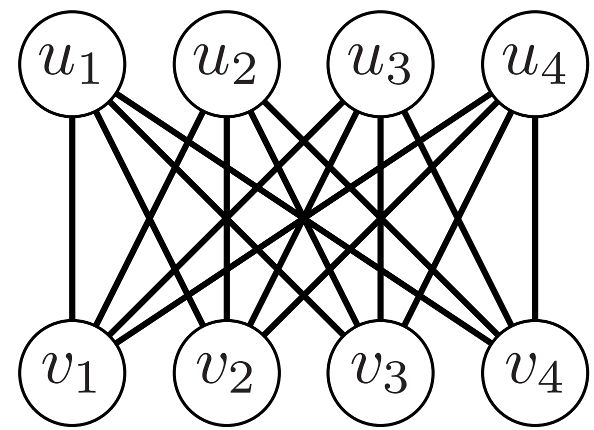
Bipartite Weighted *b*-Matching

On bipartite graph, $G = (U, V, E)$

$\{u_1, \ldots, u_n\} \in U$

$\{v_1, \ldots, v_n\} \in V$

$E = (u_i, v_j), \forall i \forall j$



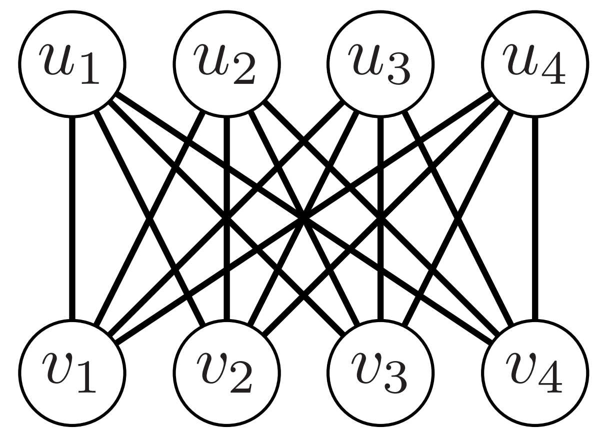
Bipartite Weighted *b*-Matching

On bipartite graph, $G = (U, V, E)$

$\{u_1, \ldots, u_n\} \in U$

$\{v_1, \ldots, v_n\} \in V$

$E = (u_i, v_j), \forall i \forall j$



A = weight matrix

s.t. weight of edge $(u_i, v_j) = A_{ij}$

Bipartite Weighted b-Matching

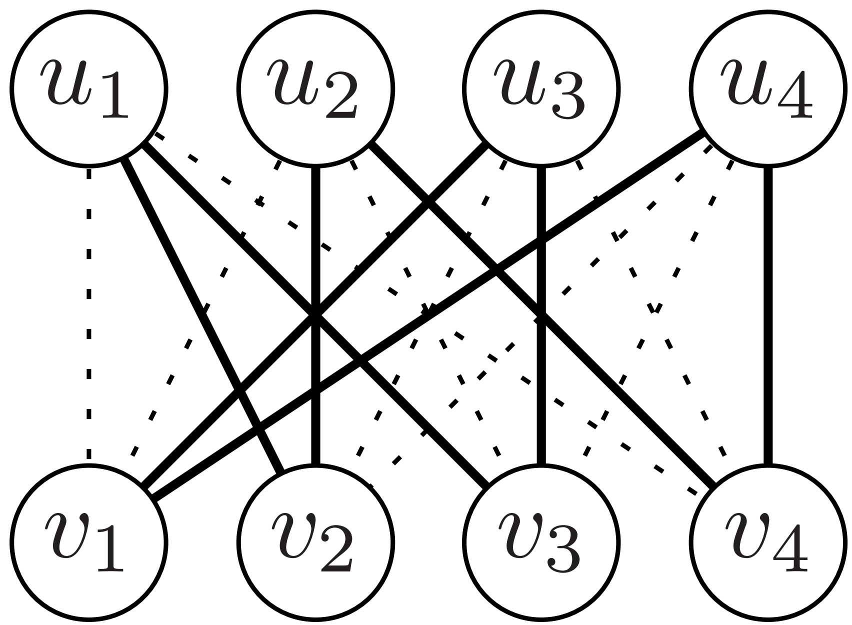
Task: Find the maximum weight subset of E such that each vertex has exactly b neighbors.

Bipartite Weighted *b*-Matching

Task: Find the maximum weight subset of E such that each vertex has exactly b neighbors.

Example:

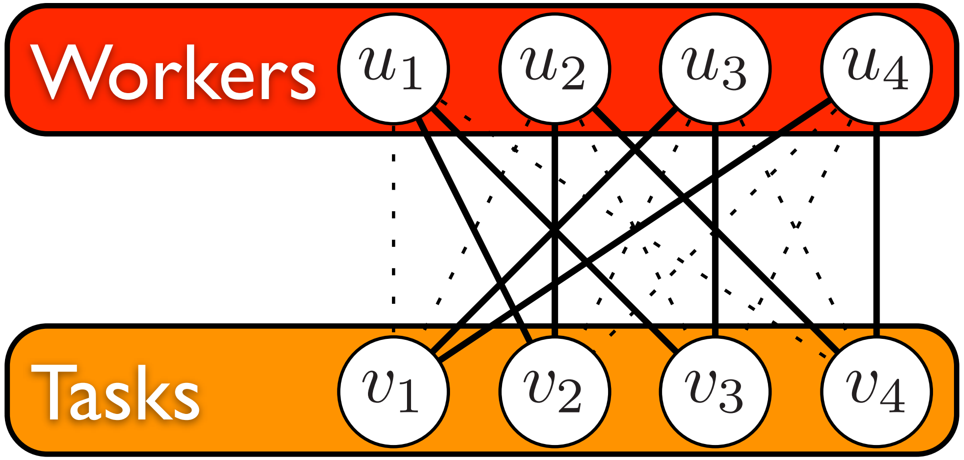
$n = 4$



$b = 2$

Bipartite Weighted *b*-Matching

Classical Application: Resource Allocation



- Manual labor
- n workers
- n tasks
- Team of b workers needed per task.
- A_{ij} skill of worker at performing task.

Bipartite Weighted *b*-Matching

Alternate uses of *b*-matching:

- Balanced *k*-nearest-neighbors
  - Each node can only be picked *k* times.
- Robust to translations of test data.
  - When test data is collected under different conditions (e.g. time, location, instrument calibration).

Bipartite Weighted b-Matching

Classical algorithms solve Max-Weighted b-Matching in $O(bn^3)$ running time, such as:

- Blossom Algorithm (Edmonds 1965)
- Balanced Network Flow

(Fremuth-Paeger, Jungnickel 1999)

Outline

1. Bipartite Weighted *b*-Matching
2. Edge Weights As a Distribution
3. Efficient Max-Product
4. Convergence Proof Sketch
5. Experiments
6. Discussion

Edge Weights As a Distribution

- Bayati, Shah, and Sharma (2005) formulated the 1-matching problem as a probability distribution.

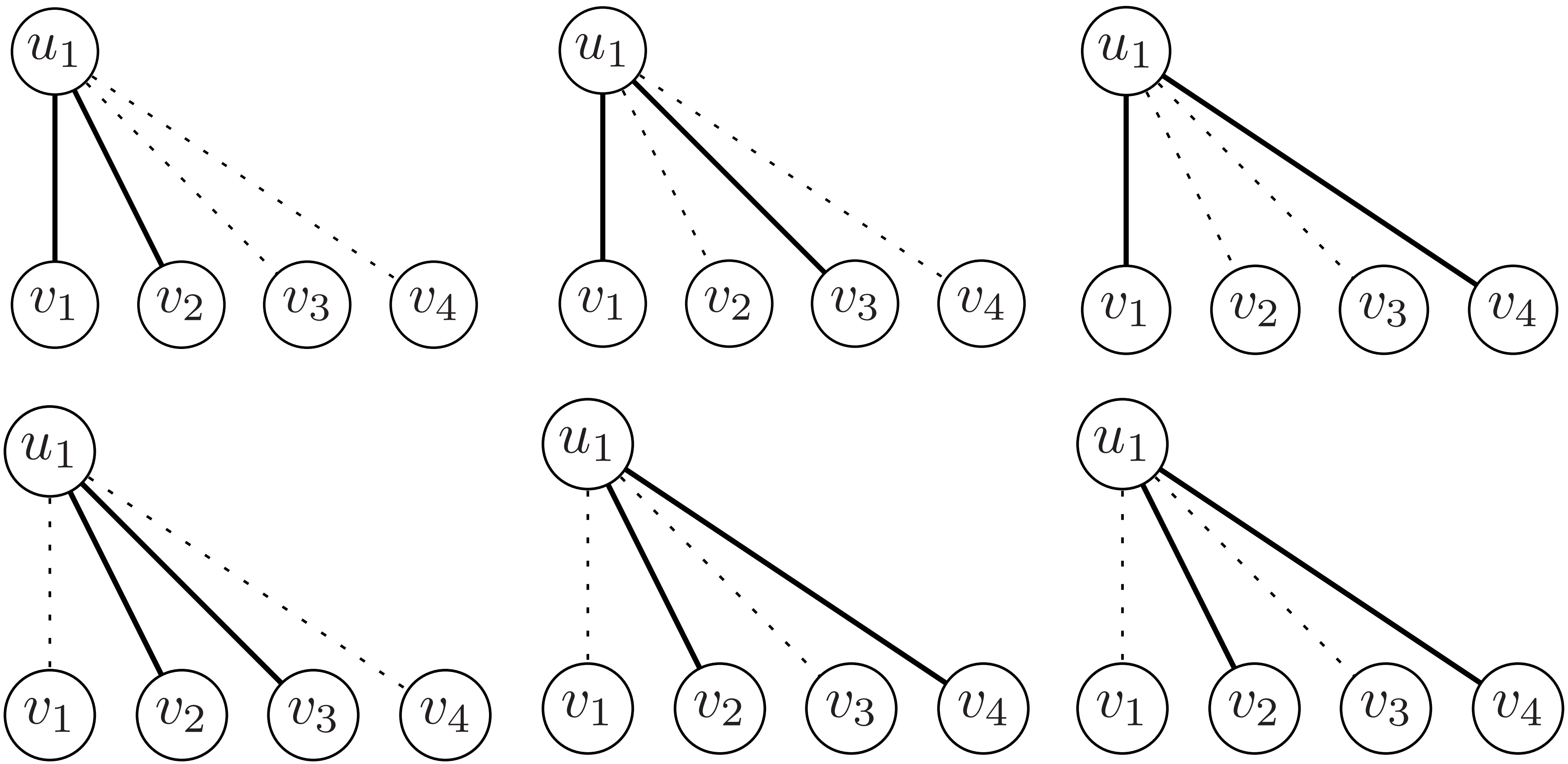
- This work generalizes to arbitrary *b*.

Edge Weights As a Distribution

Variables:

Each vertex “chooses” b neighbors.

Example: u_i



u_1
v_1
v_2
v_3
v_4
u_1
v_1
v_2
v_3
v_4
u_1
v_1
v_2
v_3
v_4
u_1
v_1
v_2
v_3
v_4
u_1
v_1
v_2
v_3
v_4
u_1
v_1
v_2
v_3
v_4

Edge Weights As a Distribution

Variables:

Each vertex “chooses” b neighbors.

Edge Weights As a Distribution

Variables:

Each vertex "chooses" b neighbors.

For vertex u_i,

$$X_i \subset V, \quad |X_i| = b$$

Similarly, for v_j have variable Y_j

Note: variables have $\binom{n}{b}$ possible settings.

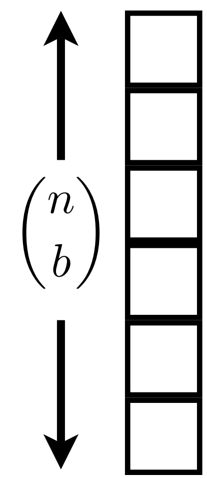
Edge Weights As a Distribution

Weights as probabilities:

Since we sum weights but multiply probabilities, exponentiate.

$$\phi(X_i) = \exp\left(\frac{1}{2} \sum_{v_j \in X_i} A_{ij}\right)$$

$$\phi(Y_j) = \exp\left(\frac{1}{2} \sum_{u_i \in Y_j} A_{ij}\right)$$

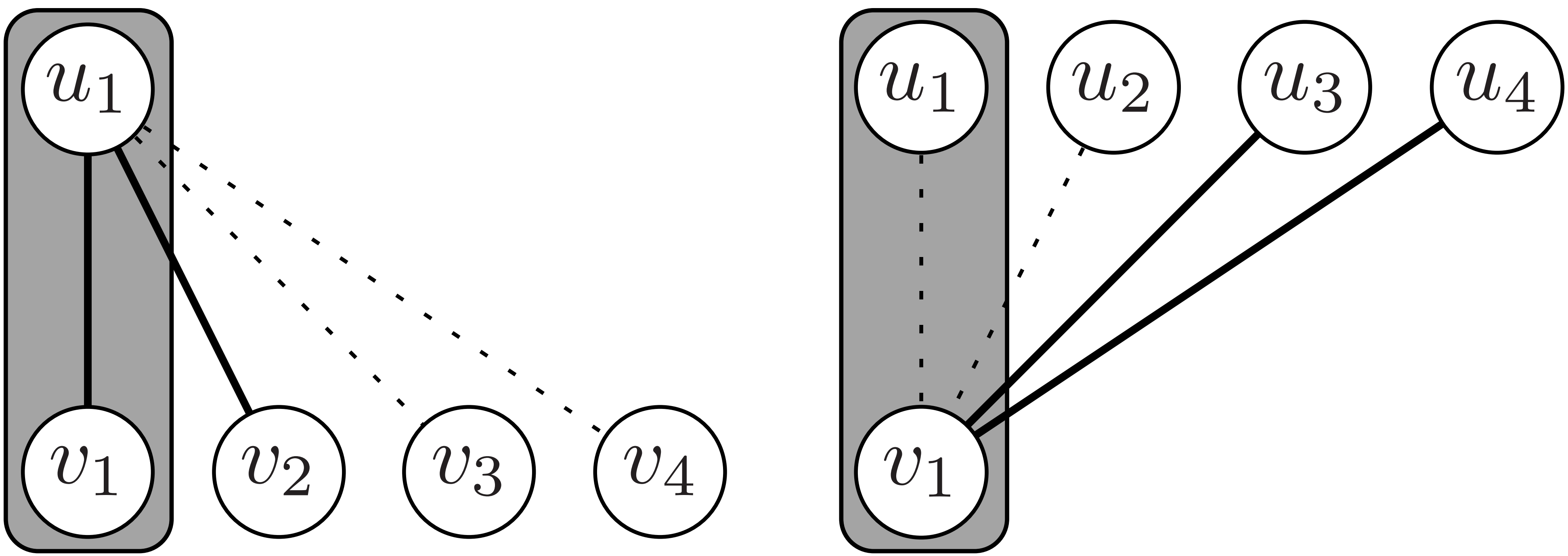


Edge Weights As a Distribution

Enforce *b*-matching:

Neighbor "choices" must agree

Example: Invalid settings



Edge Weights As a Distribution

Enforce *b*-matching:

- Neighbor “choices” must agree

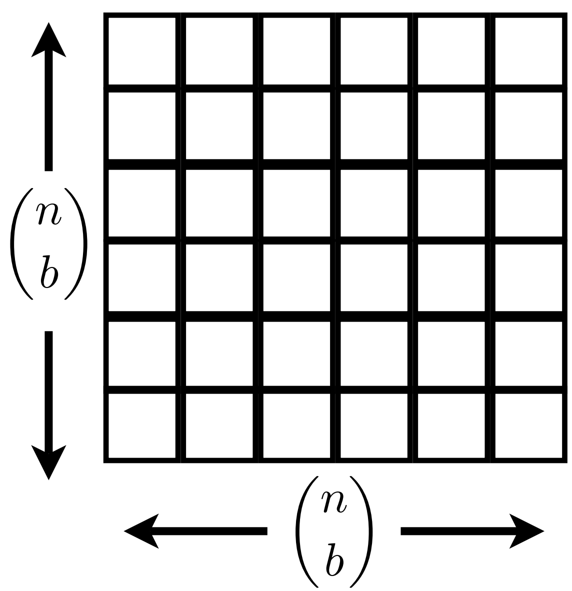
Edge Weights As a Distribution

Enforce b-matching:

Neighbor "choices" must agree

Pairwise compatibility function:

$$\psi(X_i, Y_j) =$$



$$= \neg(v_j \in X_i \oplus u_i \in Y_j).$$

Edge Weights As a Distribution

$$P(X, Y) = \frac{1}{Z} \prod_{i,j=1}^{n} \psi(X_i, Y_j) \prod_{k=1}^{n} \phi(X_k)\phi(Y_j)$$

$$\phi(X_i) = \exp(\frac{1}{2} \sum_{v_j \in X_i} A_{ij}) \qquad \phi(Y_j) = \exp(\frac{1}{2} \sum_{u_i \in Y_j} A_{ij})$$

$$\psi(X_i, Y_j) = \neg(v_j \in X_i \oplus u_i \in Y_j).$$

Edge Weights As a Distribution

$$P(X,Y) = \frac{1}{Z} \prod_{i,j=1}^{n} \psi(X_i, Y_j) \prod_{k=1}^{n} \phi(X_k)\phi(Y_j)$$

$$\phi(X_i) = \exp(\frac{1}{2} \sum_{v_j \in X_i} A_{ij}) \qquad \phi(Y_j) = \exp(\frac{1}{2} \sum_{u_i \in Y_j} A_{ij})$$

$$\psi(X_i, Y_j) = \neg(v_j \in X_i \oplus u_i \in Y_j).$$

Ignore the *Z* normalization, *P(X,Y)* is exactly the exponentiated weight of the *b*-matching.

Edge Weights As a Distribution

$$P(X,Y) = \frac{1}{Z} \prod_{i,j=1}^{n} \psi(X_i, Y_j) \prod_{k=1}^{n} \phi(X_k)\phi(Y_j)$$

$$\phi(X_i) = \exp\left(\frac{1}{2} \sum_{v_j \in X_i} A_{ij}\right) \qquad \phi(Y_j) = \exp\left(\frac{1}{2} \sum_{u_i \in Y_j} A_{ij}\right)$$

$$\psi(X_i, Y_j) = \neg(v_j \in X_i \oplus u_i \in Y_j).$$

Also, since we're maximizing, ignore the 1/2 (makes the math more readable).

Outline

1. Bipartite Weighted *b*-Matching
2. Edge Weights As a Distribution
3. Efficient Max-Product
4. Convergence Proof Sketch
5. Experiments
6. Discussion

Standard Max-Product

Send messages between variables:

$$m_{X_i}(Y_j) = \frac{1}{Z} \max_{X_i} \left[\phi(X_i)\psi(X_i, Y_j) \prod_{k \neq j} m_{Y_k}(X_i) \right]$$

Fuse messages to obtain beliefs (or estimate of max-marginals):

$$b(X_i) = \frac{1}{Z} \phi(X_i) \prod_{k} m_{Y_k}(X_i)$$

Standard Max-Product

Converges to true maximum on any tree structured graph (Pearl 1986).

We show that it converges to the correct maximum on our graph.

Standard Max-Product

Converges to true maximum on any tree structured graph (Pearl 1986).

We show that it converges to the correct maximum on our graph.

But what about the $\binom{n}{b}$-length message and belief vectors?

Efficient Max-Product

- Use algebraic tricks to reduce $\binom{n}{b}$-length message vectors to scalars.

- Derive new update rule for scalar messages.

- Use similar trick to maximize belief vectors efficiently.

Let's speed through the math.

Efficient Max-Product

Take advantage of binary $\psi(x_i, y_j)$ function:

Message vectors consist of only two values

$$m_{x_i}(y_j) \propto \max_{v_j \in x_i} \phi(x_i) \prod_{k \neq j} m_{y_k}(x_i), \text{ if } u_i \in y_j$$

$$m_{x_i}(y_j) \propto \max_{v_j \notin x_i} \phi(x_i) \prod_{k \neq j} m_{y_k}(x_i), \text{ if } u_i \notin y_j$$

Efficient Max-Product

Take advantage of binary $\psi(x_i, y_j)$ function:

Message vectors consist of only two values

$$m_{x_i}(y_j) \propto \max_{v_j \in x_i} \phi(x_i) \prod_{k \neq j} m_{y_k}(x_i), \text{ if } u_i \in y_j$$

$$m_{x_i}(y_j) \propto \max_{v_j \notin x_i} \phi(x_i) \prod_{k \neq j} m_{y_k}(x_i), \text{ if } u_i \notin y_j$$

If we rename these two values we can break up the product.

Efficient Max-Product

Take advantage of binary $\psi(x_i, y_j)$ function:

Message vectors consist of only two values

$$\mu_{x_i y_j} \quad \propto \quad \max_{v_j \in x_i} \phi(x_i) \prod_{u_k \in x_i \setminus v_j} \mu_{ki} \prod_{u_k \notin x_i \setminus v_j} \nu_{ki}$$

$$\nu_{x_i y_j} \quad \propto \quad \max_{v_j \notin x_i} \phi(x_i) \prod_{u_k \in x_i \setminus v_j} \mu_{ki} \prod_{u_k \notin x_i \setminus v_j} \nu_{ki}.$$

If we rename these two values we can break up the product.

Efficient Max-Product

“Normalize” messages by dividing whole vector by $\nu_{x_i y_j}$

$$\hat{\mu}_{x_i y_j} = \frac{\mu_{x_i y_j}}{\nu_{x_i y_j}} \quad \text{and} \quad \hat{\nu}_{x_i y_j} = 1$$

Efficient Max-Product

"Normalize" messages by dividing whole vector by $\nu_{x_i y_j}$

$$\hat{\mu}_{x_i y_j} = \frac{\mu_{x_i y_j}}{\nu_{x_i y_j}} \quad \text{and} \quad \hat{\nu}_{x_i y_j} = 1$$

$\binom{n}{b}$-length vector $\longrightarrow$ scalar

Efficient Max-Product

Derive update rule:

$$\hat{\mu}_{x_i y_j} \quad = \quad \frac{\max_{j \in x_i} \phi(x_i) \prod_{k \in x_i \setminus j} \hat{\mu}_{ki}}{\max_{j \notin x_i} \phi(x_i) \prod_{k \in x_i \setminus j} \hat{\mu}_{ki}}$$

Efficient Max-Product

Derive update rule:

$$\hat{\mu}_{x_i y_j} = \frac{\max_{j \in x_i} \phi(x_i) \prod_{k \in x_i \setminus j} \hat{\mu}_{ki}}{\max_{j \notin x_i} \phi(x_i) \prod_{k \in x_i \setminus j} \hat{\mu}_{ki}}$$

$$\phi(x_i) \propto \prod_{k \in x_i} exp(A_{ik})$$

Efficient Max-Product

Derive update rule:

$$\hat{\mu}_{x_i y_j} = \frac{\max_{j \in x_i} \phi(x_i) \prod_{k \in x_i \setminus j} \hat{\mu}_{ki}}{\max_{j \notin x_i} \phi(x_i) \prod_{k \in x_i \setminus j} \hat{\mu}_{ki}}$$

$$= \frac{\max_{j \in x_i} \prod_{k \in x_i} \exp(A_{ik}) \prod_{k \in x_i \setminus j} \hat{\mu}_{ki}}{\max_{j \notin x_i} \prod_{k \in x_i} \exp(A_{ik}) \prod_{k \in x_i \setminus j} \hat{\mu}_{ki}}$$

Efficient Max-Product

Derive update rule:

$$
\begin{aligned}
\hat{\mu}_{x_i y_j} &= \frac{\max_{j \in x_i} \phi(x_i) \prod_{k \in x_i \setminus j} \hat{\mu}_{ki}}{\max_{j \notin x_i} \phi(x_i) \prod_{k \in x_i \setminus j} \hat{\mu}_{ki}} \\
&= \frac{\max_{j \in x_i} \prod_{k \in x_i} \exp(A_{ik}) \prod_{k \in x_i \setminus j} \hat{\mu}_{ki}}{\max_{j \notin x_i} \prod_{k \in x_i} \exp(A_{ik}) \prod_{k \in x_i \setminus j} \hat{\mu}_{ki}} \\
&= \frac{\exp(A_{ij}) \max_{j \in x_i} \prod_{k \in x_i \setminus j} \exp(A_{ik}) \hat{\mu}_{ki}}{\max_{j \notin x_i} \prod_{k \in x_i} \exp(A_{ij}) \hat{\mu}_{ki}}
\end{aligned}
$$

Efficient Max-Product

After canceling terms message update simplifies to

$$\hat{\mu}_{x_i y_j} = \frac{\exp(A_{ij})}{\exp(A_{i\ell})\hat{\mu}_{y_\ell x_i}}. \quad \ell =$$

*b*th greatest setting of *k* for the term $\exp(A_{ik}) m_{y_k}(x_i), \textbf{s. t. } k \neq j$

and we maximize beliefs with

$$\begin{aligned} \max_{x_i} b(x_i) &\propto \max_{x_i} \phi(x_i) \prod_{k \in x_i} \hat{\mu}_{y_k x_i} \\ &\propto \max_{x_i} \prod_{k \in x_i} \exp(A_{ik}) \hat{\mu}_{y_k x_i} \end{aligned}$$

Both these updates take *O(bn)* time per vertex.

Outline

1. Bipartite Weighted *b*-Matching
2. Edge Weights As a Distribution
3. Efficient Max-Product
4. Convergence Proof Sketch
5. Experiments
6. Discussion

Convergence Proof Sketch

Convergence Proof Sketch

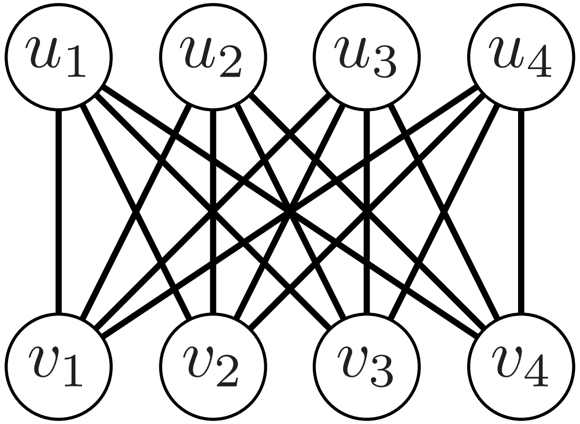
Assumptions:

- Optimal b-matching is unique.
- $\epsilon =$ difference between weight of best and 2nd best b-matching is constant.

- Weights treated as constants.

Convergence Proof Sketch

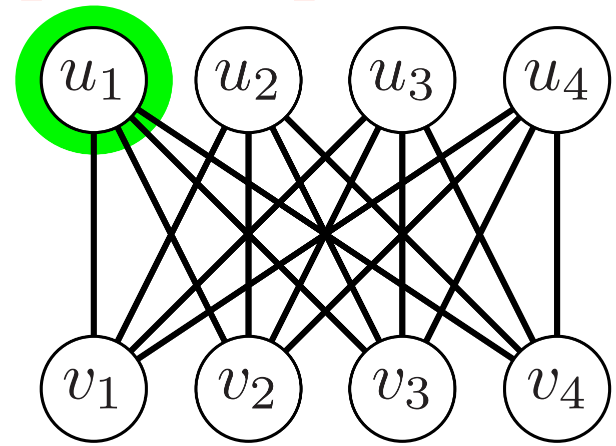
Basic mechanism: Unwrapped Graph, *T*



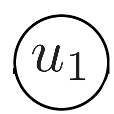
1. Pick root node
2. Copy all neighbors
3. Continue but don't backtrack
4. Continue to depth *d*

Convergence Proof Sketch

Basic mechanism: Unwrapped Graph, T

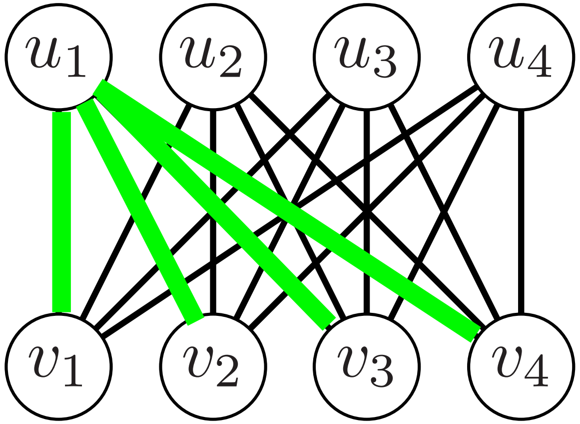


1. Pick root node
2. Copy all neighbors
3. Continue but don't backtrack
4. Continue to depth d

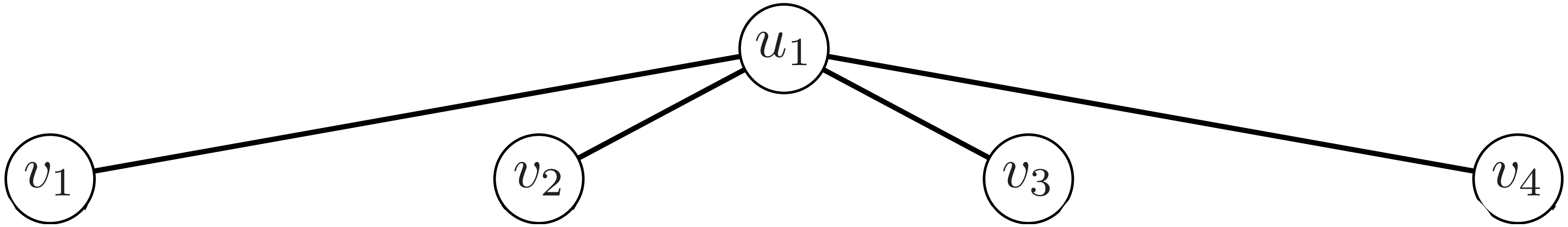


Convergence Proof Sketch

Basic mechanism: Unwrapped Graph, T

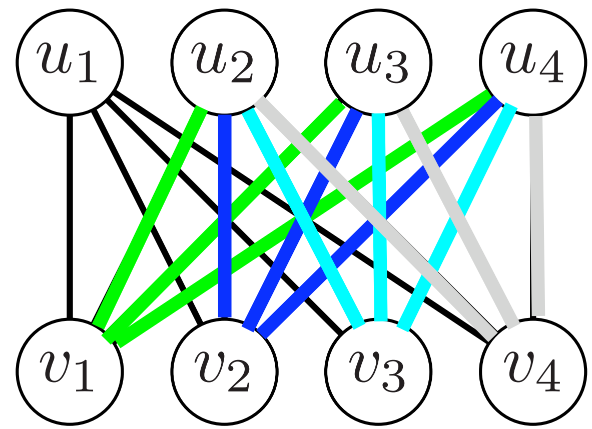


1. Pick root node
2. Copy all neighbors
3. Continue but don't backtrack
4. Continue to depth d

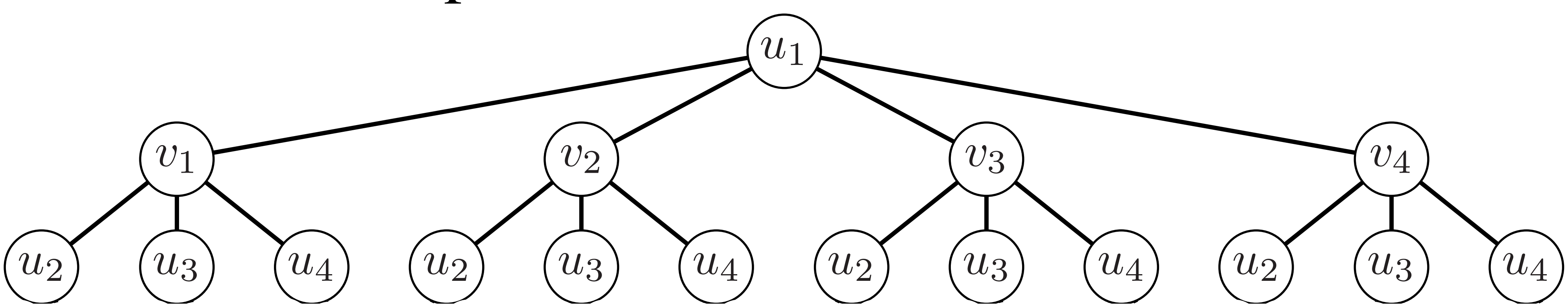


Convergence Proof Sketch

Basic mechanism: Unwrapped Graph, *T*

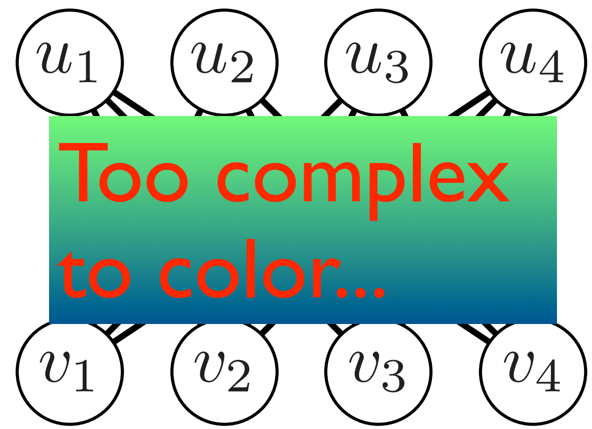


1. Pick root node
2. Copy all neighbors
3. Continue but don't backtrack
4. Continue to depth *d*

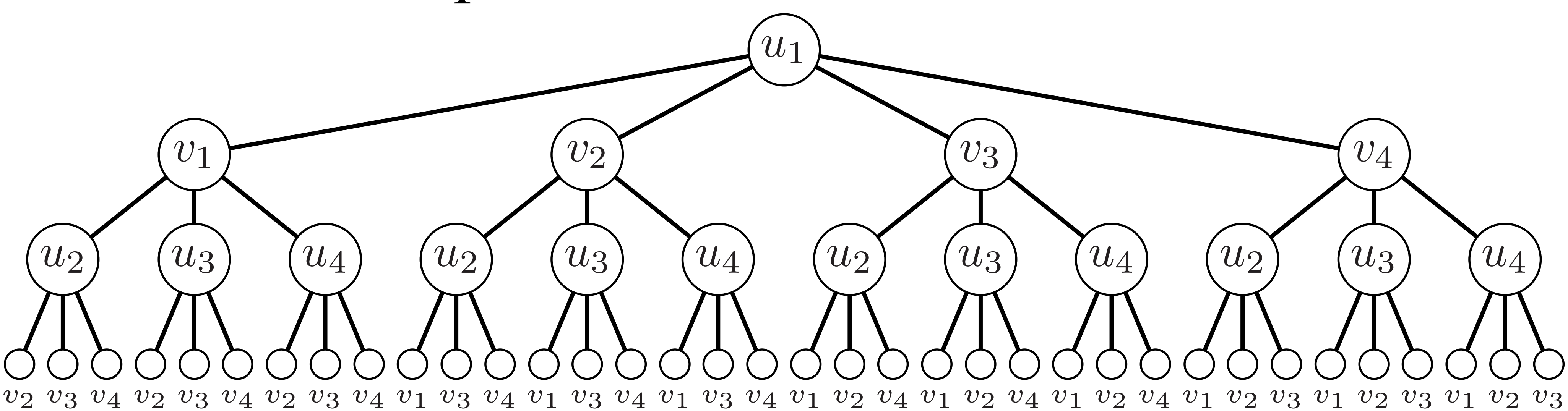


Convergence Proof Sketch

Basic mechanism: Unwrapped Graph, *T*



1. Pick root node
2. Copy all neighbors
3. Continue but don't backtrack
4. Continue to depth *d*



Convergence Proof Sketch

Basic mechanism: Unwrapped Graph, T

- Construction follows loopy BP messages in reverse.

- True max-marginals of root node are exactly belief at iteration d.

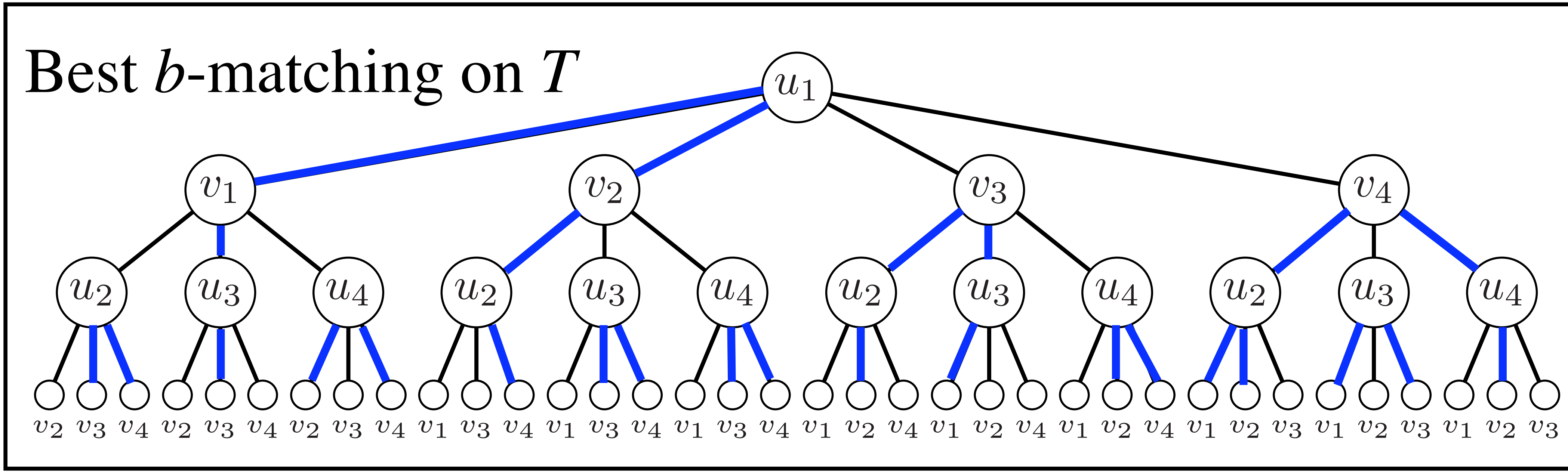
- Max of unwrapped graph distribution is the maximum weight b-matching on tree.

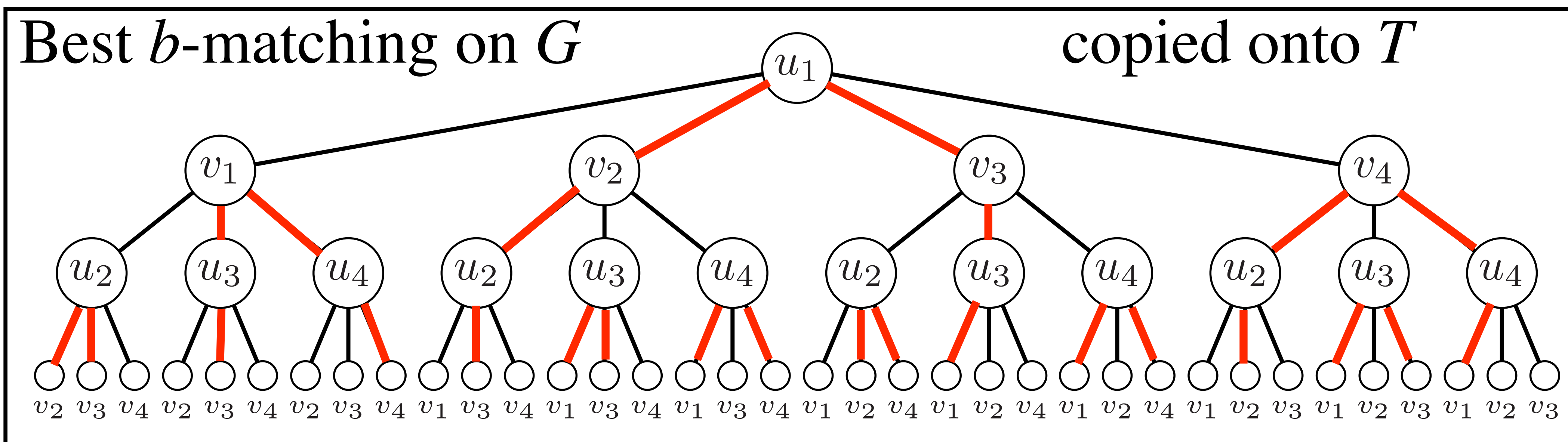
Convergence Proof Sketch

Proof by contradiction:

What happens if optimal b-matching on T differs from optimal b-matching on G at root?

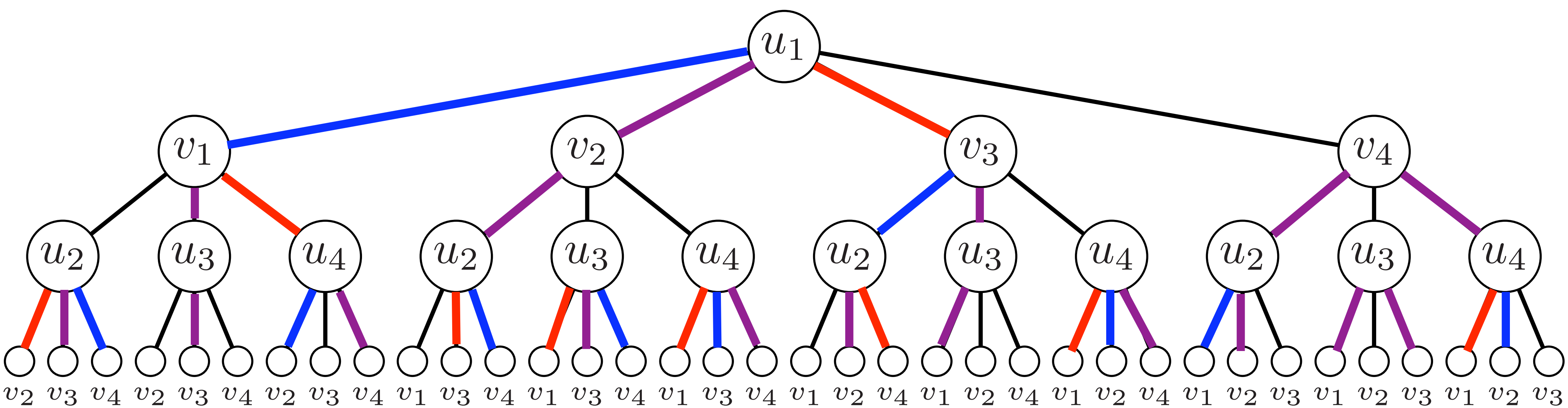
Convergence Proof Sketch





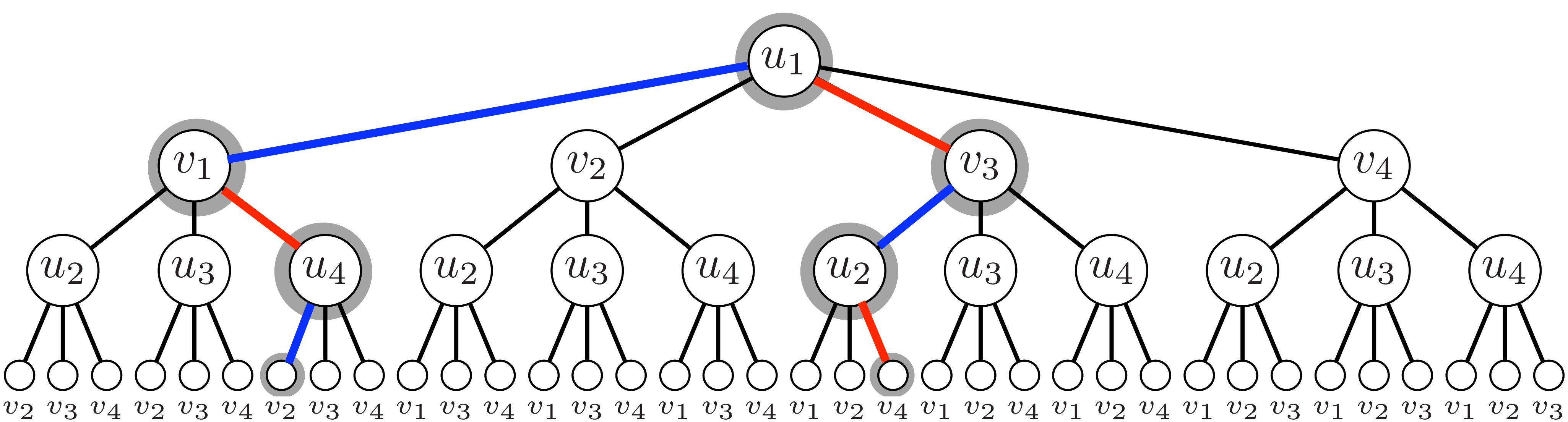
Convergence Proof Sketch

There exists at least one path on T that alternates between edges that are b-matched in each b-matching.



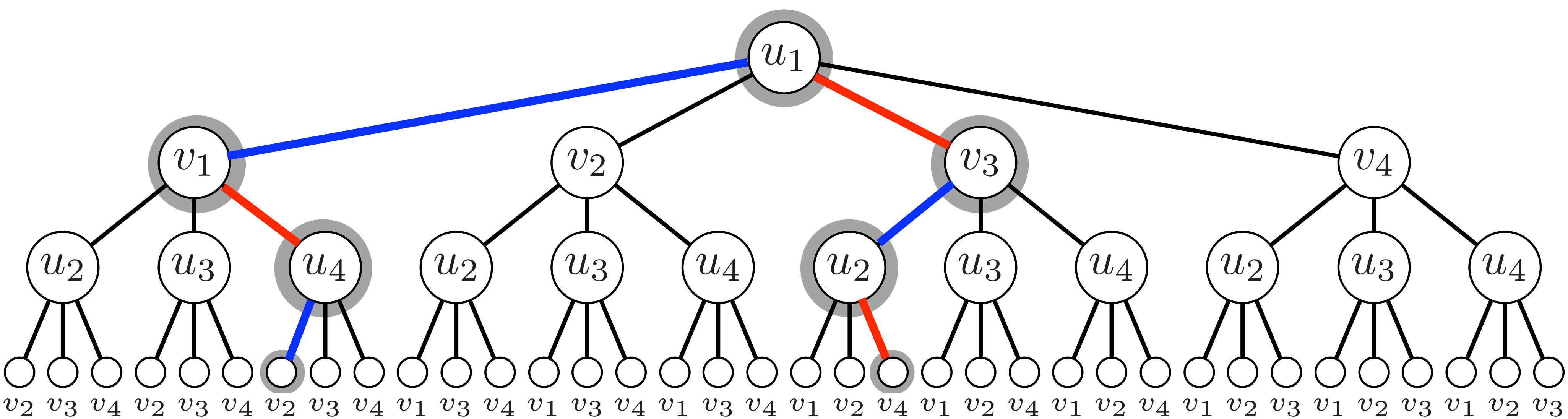
Convergence Proof Sketch

There exists at least one path on T that alternates between edges that are b-matched in each b-matching.



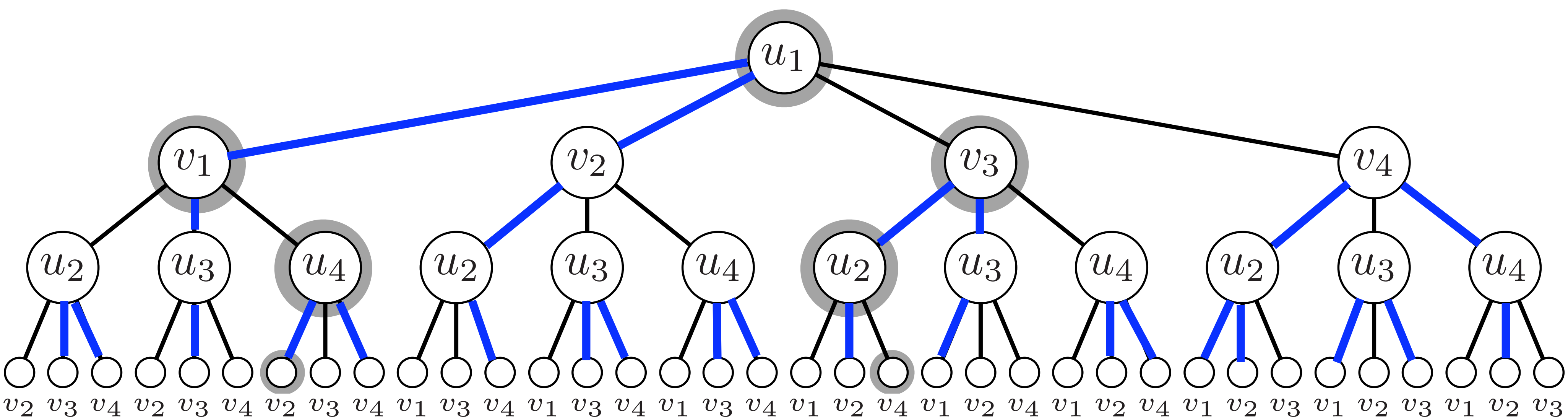
Convergence Proof Sketch

Claim: If depth d is great enough, if we replace the blue edges of this path with the red edges in the optimal b-matching on T, we get a new b-matching with greater weight.



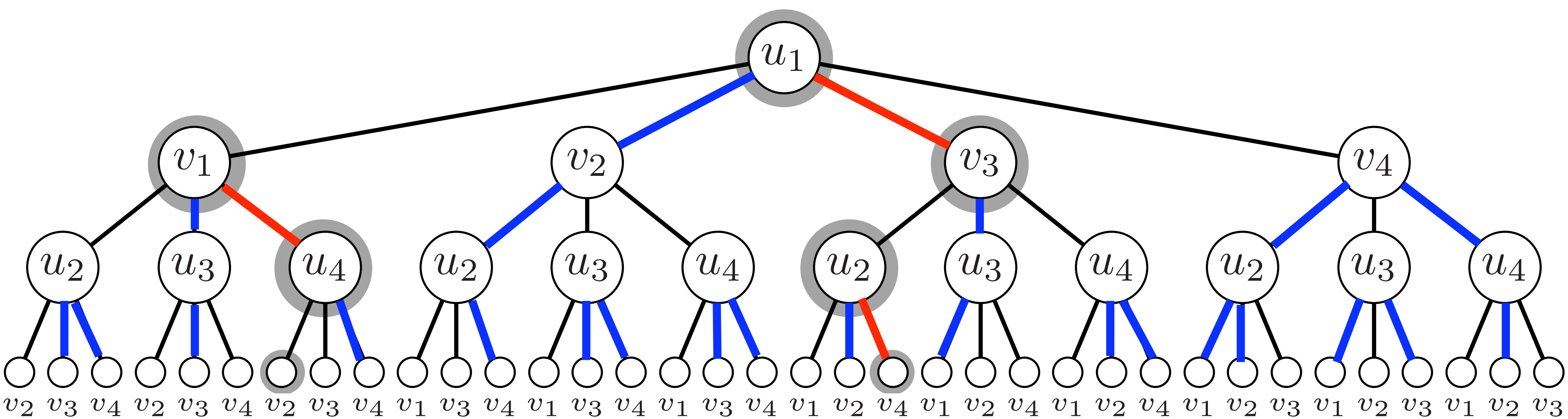
Convergence Proof Sketch

Claim: If depth d is great enough, if we replace the blue edges of this path with the red edges in the optimal b-matching on T, we get a new b-matching with greater weight.



Convergence Proof Sketch

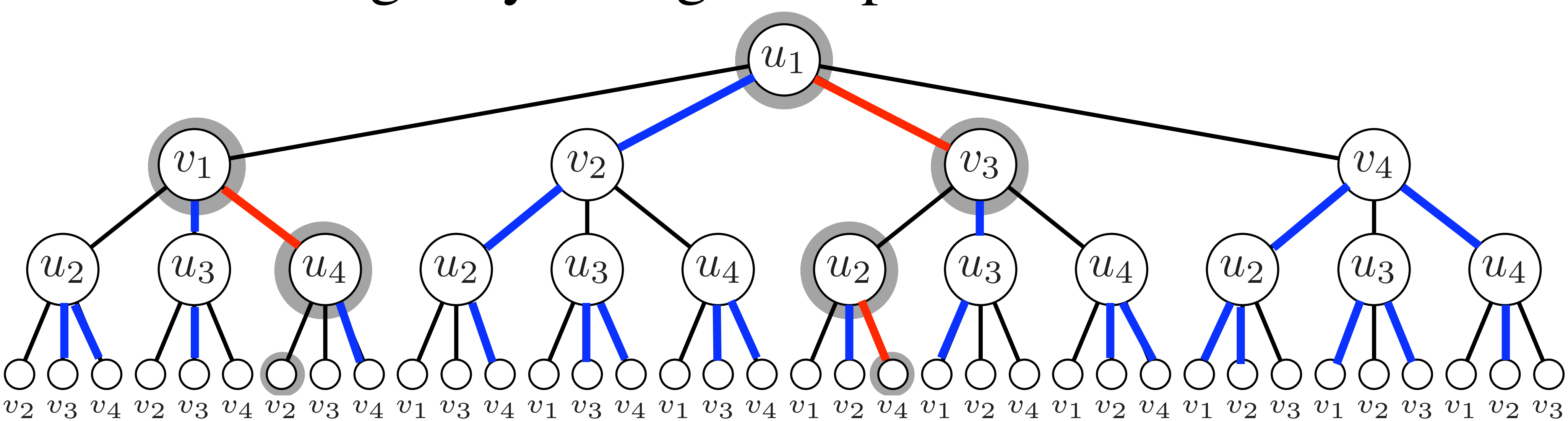
Claim: If depth d is great enough, if we replace the blue edges of this path with the red edges in the optimal b-matching on T, we get a new b-matching with greater weight.



Convergence Proof Sketch

Modifying optimal *b*-matching produced better *b*-matching → Original contradiction impossible.

We can analyze the change in weight by looking only at edges on path.



Convergence Proof Sketch

Loopy BP converges to true maximum weight b-matching in d iterations

$$d \geq \frac{n}{\epsilon} \max_{i,j} A_{ij} = O(n)$$

$\epsilon =$ difference between weight of best and 2nd best b-matching.

Convergence Proof Sketch

Loopy BP converges to true maximum weight b-matching in d iterations

$$d \geq \frac{n}{\epsilon} \max_{i,j} A_{ij} = O(n)$$

$\epsilon =$ difference between weight of best and 2nd best b-matching.

Running time of full algorithm: $O(bn^3)$

Outline

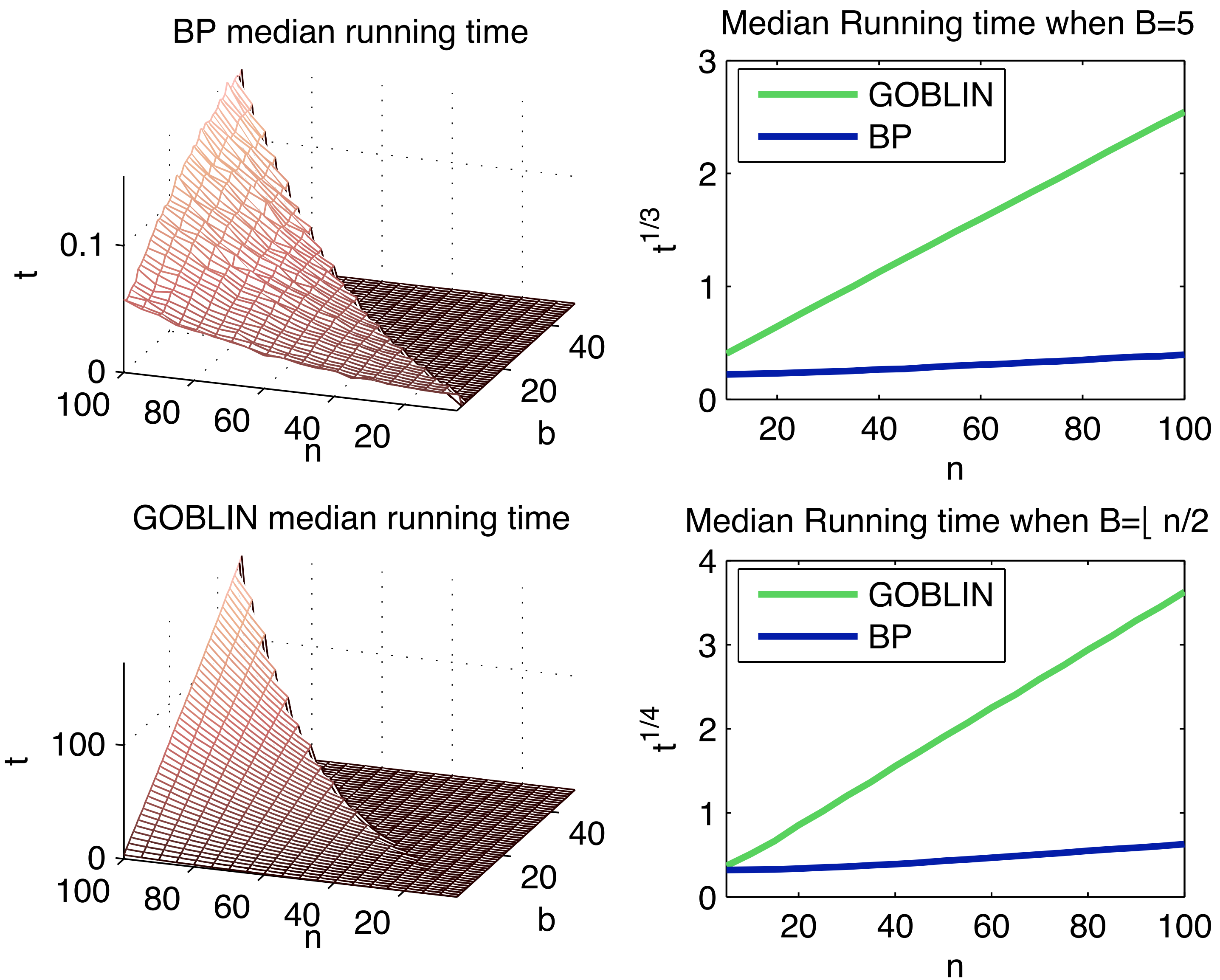
1. Bipartite Weighted *b*-Matching
2. Edge Weights As a Distribution
3. Efficient Max-Product
4. Convergence Proof Sketch
5. Experiments
6. Discussion

Experiments

Running time comparison against GOBLIN graph optimization library.

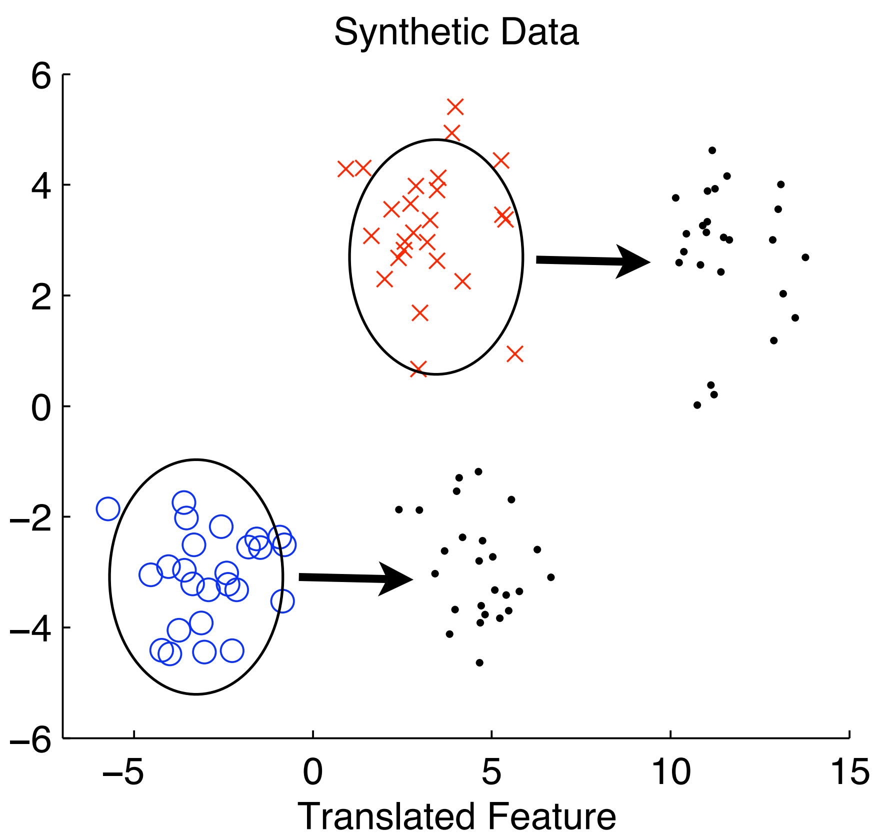
- Random weights.
- Varied graph size n from 3 to 100
- Varied b from 1 to $\lfloor n/2 \rfloor$

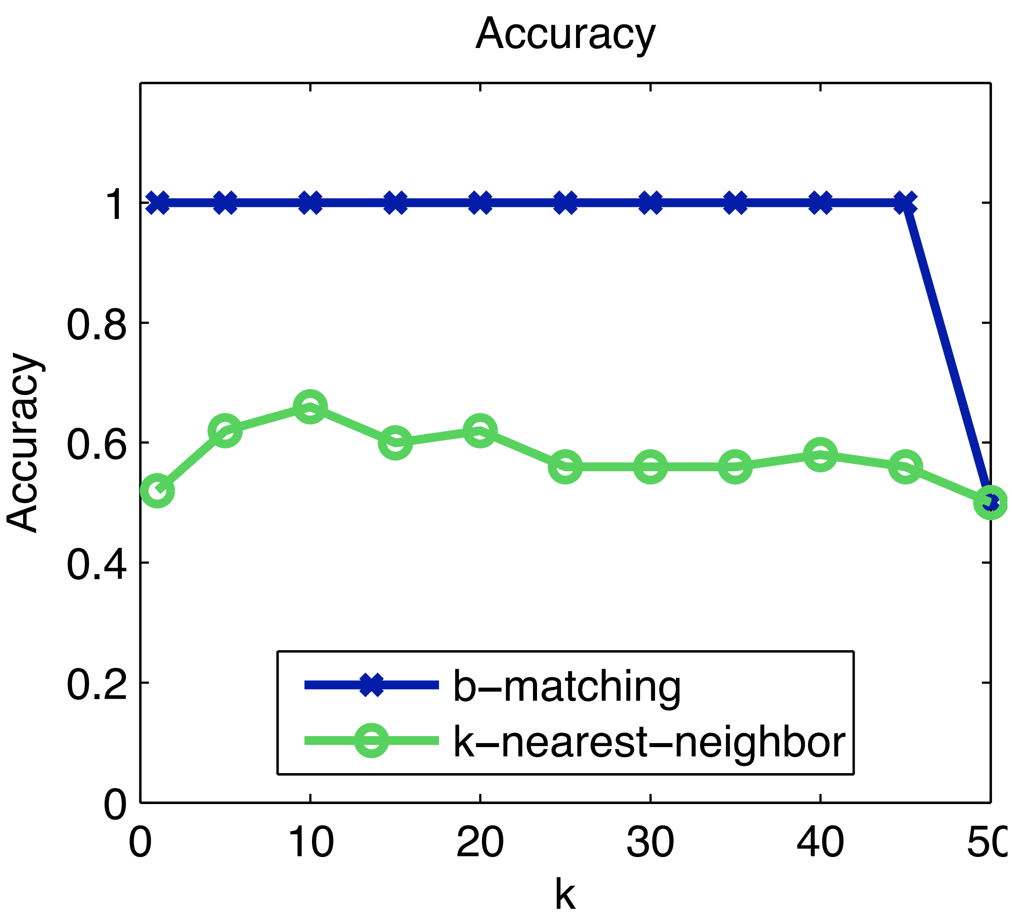
Experiments



Experiments: Translated Test Data

On toy data, translation cripples KNN but *b*-matching makes no classification errors.





Experiments

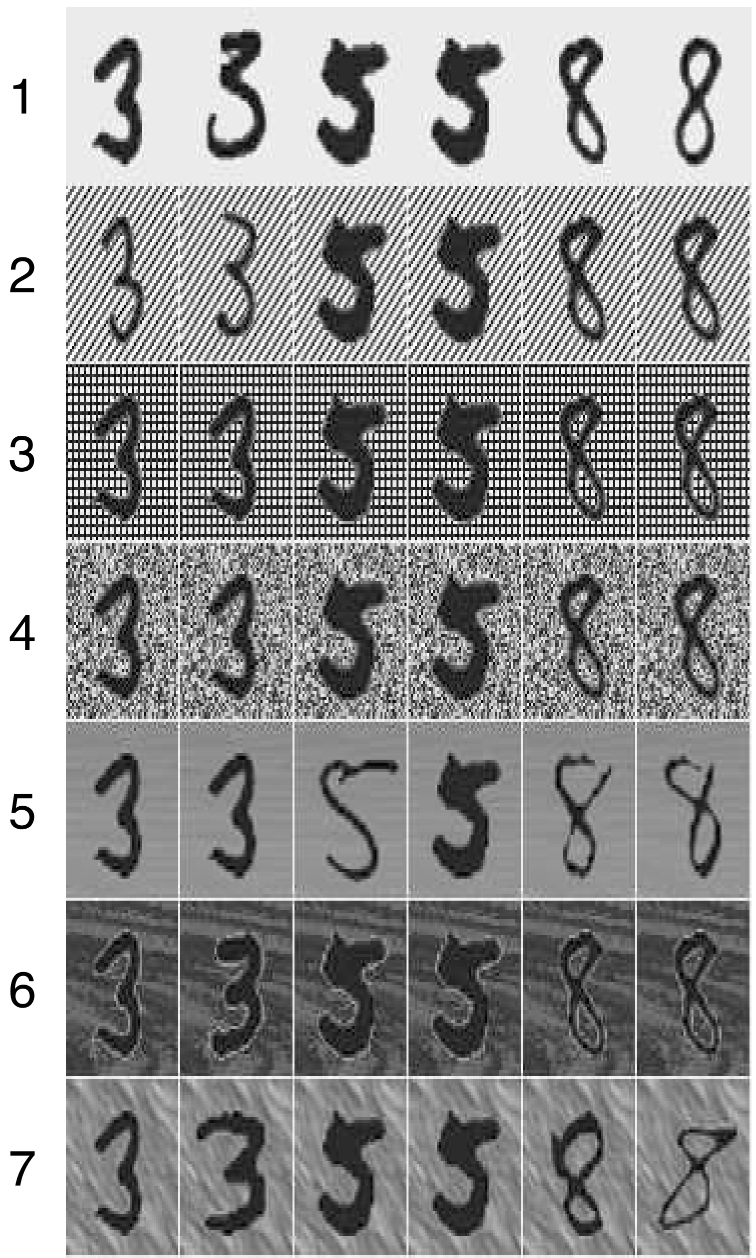
MNIST Digits with pseudo-translation

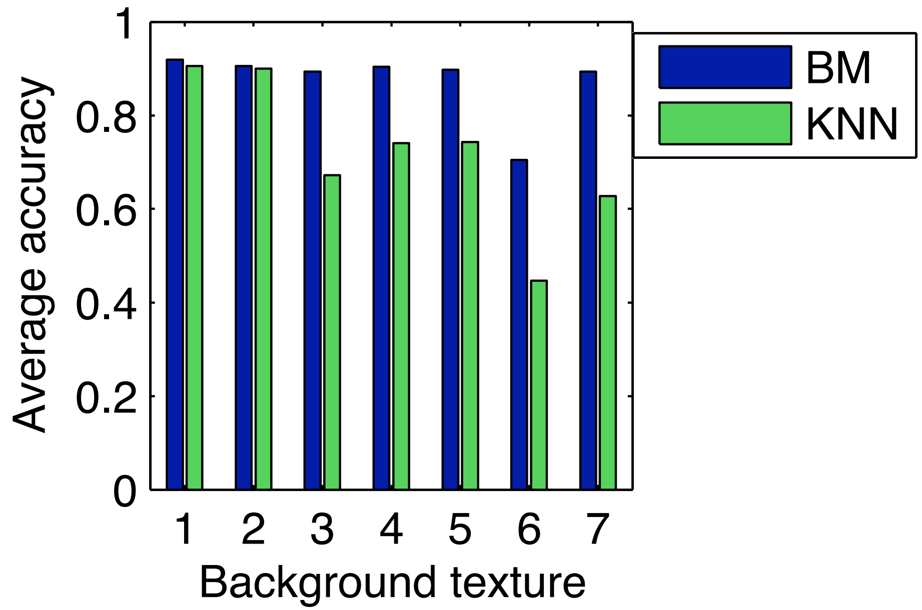
- Image data with background changes is like translation.

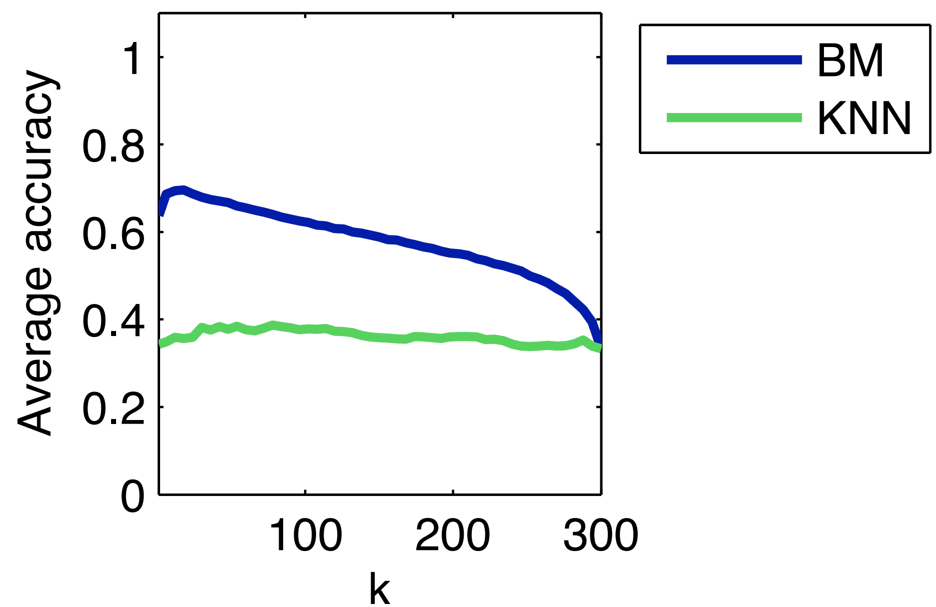
- Train on MNIST digits 3, 5, and 8.

- Test on new examples with various "bluescreen" textures.

Experiments







Outline

1. Bipartite Weighted *b*-Matching
2. Edge Weights As a Distribution
3. Efficient Max-Product
4. Convergence Proof Sketch
5. Experiments
6. Discussion

Discussion

Provably convergent belief propagation for a new type of graph (b-matchings).

+ Empirically faster than previous algorithms.

+ Parallelizeable

- Only bipartite case.

- Requires unique maximum.

Interesting theoretical results coming out of sum-product for approximating marginals.