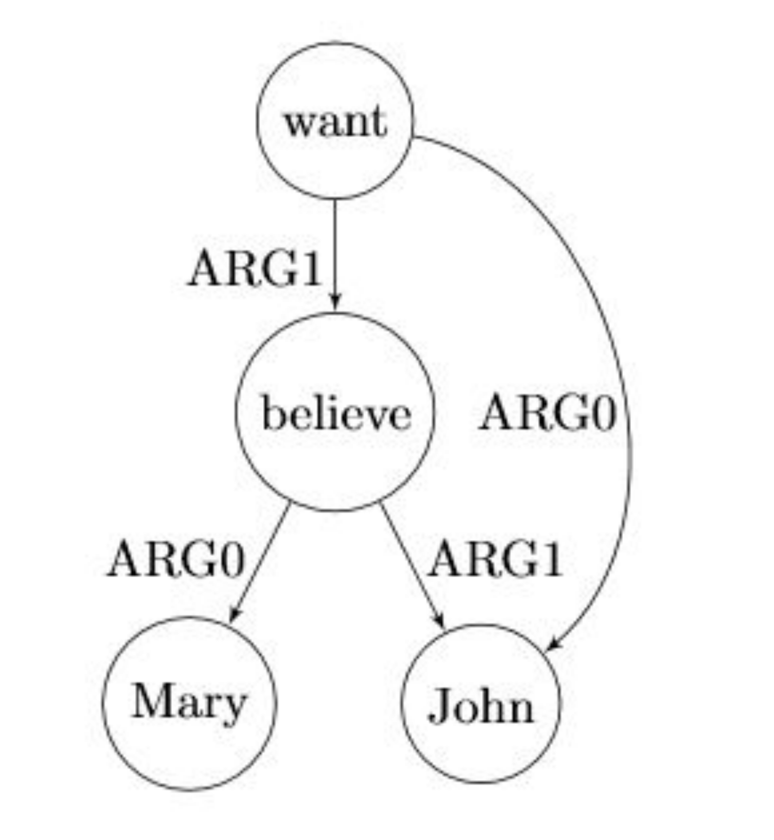
Graph Similarity Scoring
Applied to
Abstract Meaning Representation

Justin DeBenedetto

The College of Engineering
at the University of Notre Dame

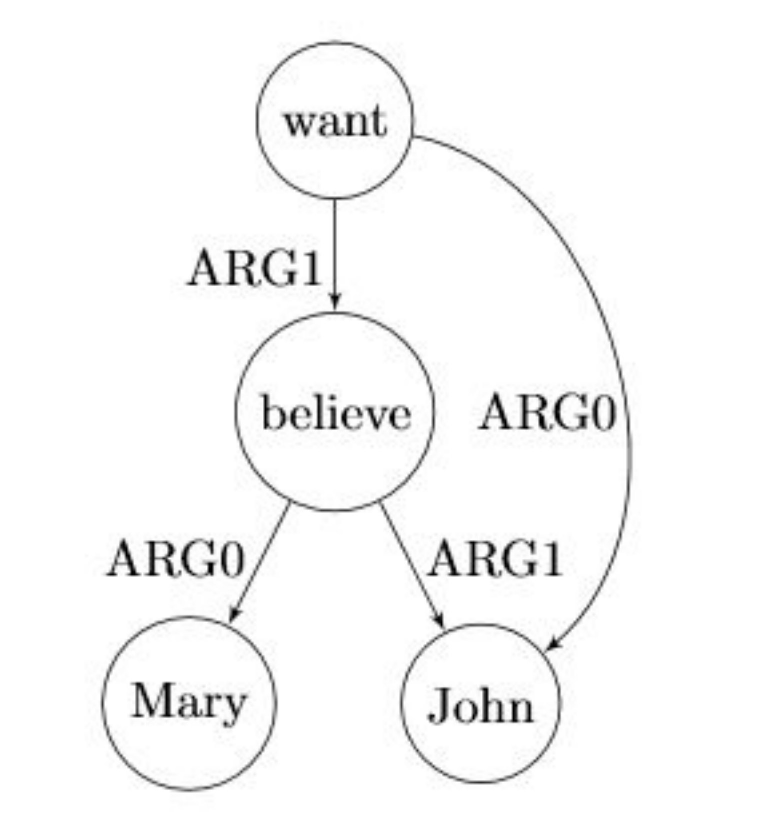
Abstract Meaning Representation (AMR)

- AMRs are a semantic formalism which models sentences



Abstract Meaning Representation (AMR)

- AMRs are a semantic formalism which models sentences
  - Nodes represent concepts
  - Edges represent relations between concepts
    - Semantic roles
    - ARG0 = Agent
    - ARG1 = Patient
    - Example AMR for sentence: "John wants Mary to believe him."



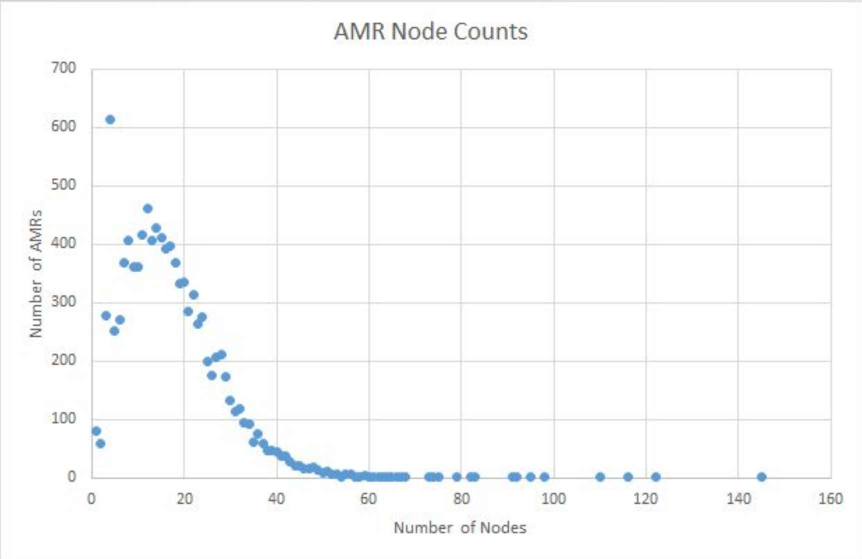
Properties of AMRS as Graphs

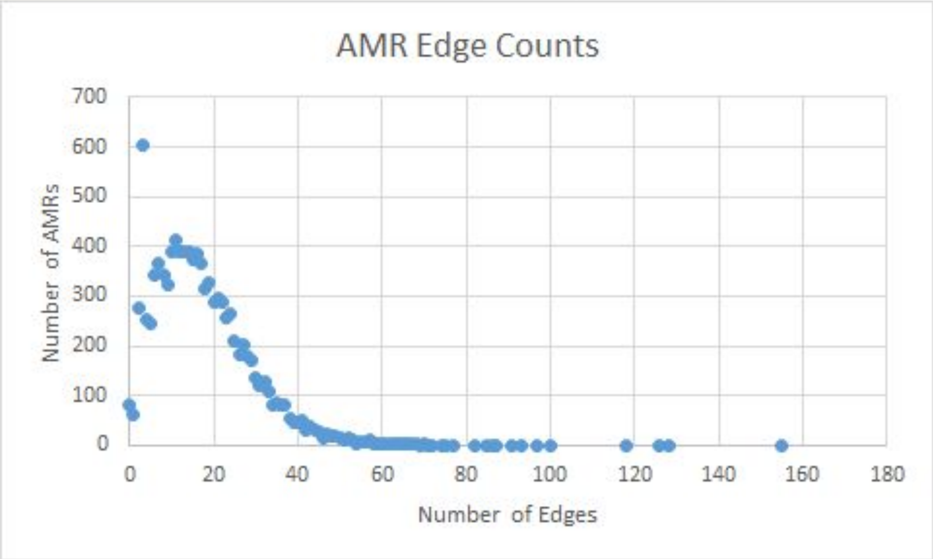
- Some properties of AMRs
  - Directed Acyclic Graphs (DAGs)
  - Single rooted (focus of sentence)
  - Each AMR represents a sentence

Dataset

- Set of 10,312 AMRs from various news sources
- Average number of nodes is: 17.1
- Average number of edges is: 17.1
- More than half are trees

Dataset


AMR Node Counts
Number of AMRs
Number of Nodes
0
100
200
300
400
500
600
700
0
20
40
60
80
100
120
140
160


AMR Edge Counts
Number of AMRs
Number of Edges
0
100
200
300
400
500
600
700
0
20
40
60
80
100
120
140
160
180

Kernel: Graph Similarity Scoring

- Use some AMRs for training
  - Given multiple candidate AMRs, choose best one
  - Need a way to score each choice
  - Want pairwise digraph similarity score
- Typical metric used for AMRs is SMATCH

SMATCH Score

- Semantic Match score
  - Find best matching of nodes
  - Score based on node and edge labels
  - F1 score
    - Node label
    - For each edge: edge type and end points

Pseudocode

Algorithm 1 Basic SMATCH pseudocode

```
procedure getSMATCH(a,b)
    maxF1 ← 0
    for mapping in nodeMapping(a,b) do
        correct ← 0
        for alignedPair in mapping do
            if labels match then
                correct ← correct + 1
        for edges in a do
            replace end-points with aligned nodes from b
            if new edge exists in b then
                correct ← correct + 1
        precisionDenominator ← number of triples in b
        recallDenominator ← number of triples in a
        precision ← correct/precisionDenominator
        recall ← correct/recallDenominator
        f1 ← (recall + precision)/2
        if f1 > maxF1 then
            maxF1 ← f1
    return maxF1
procedure nodeMapping(a,b)
    allAlignments ← empty
    Select node_a in a
    for node_b in b do
        newAlignments ← align node_a to node_b
        newA ← a − node_a
        newB ← b − node_b
        newAlignments ← nodeMapping(newA, newB)
        append newAlignments to allAlignments
    return allAlignments
```

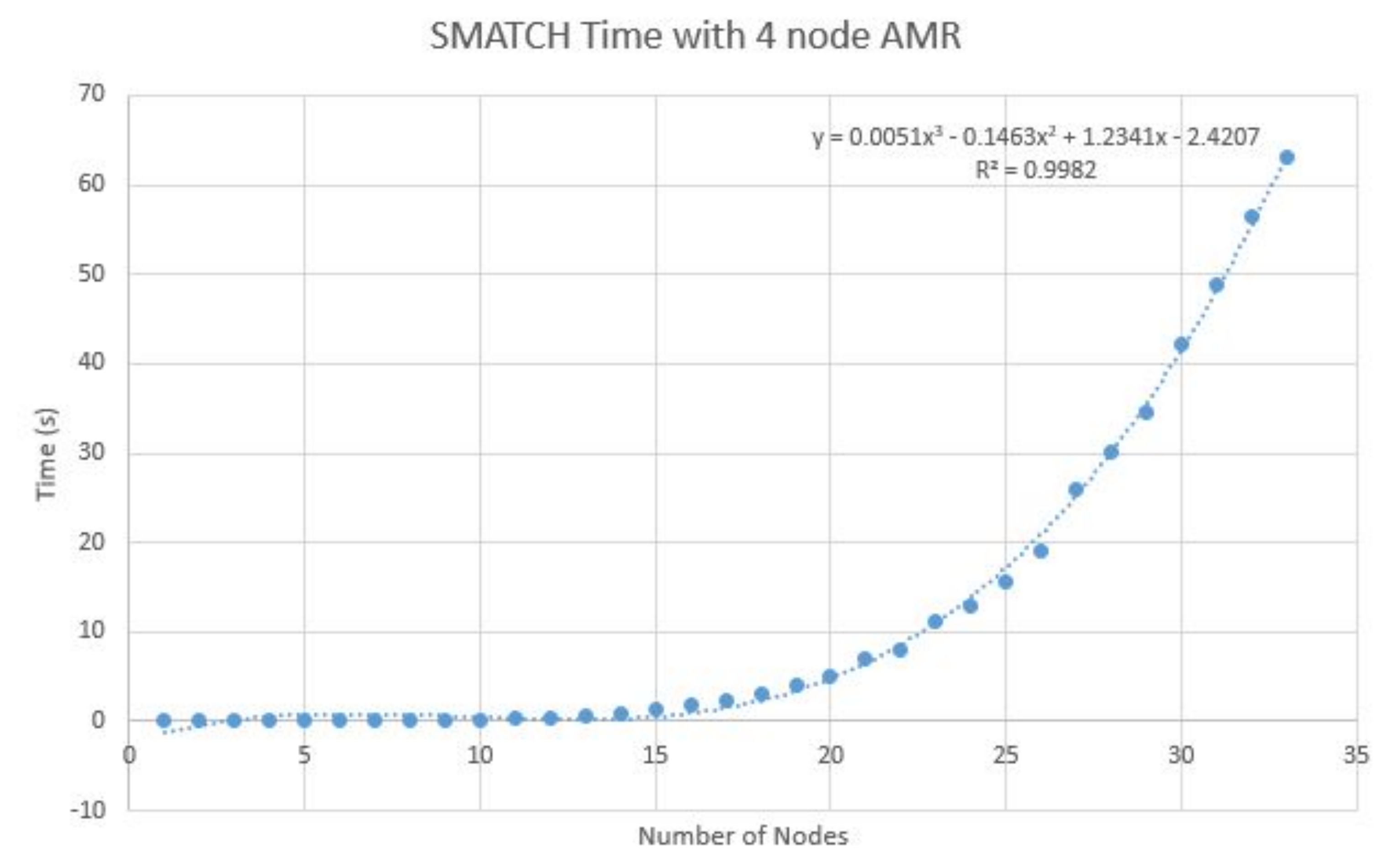
Complexity

- Most direct way (previous slide) has complexity ~O(N!/(N-M)!*|M+E|)
  - N = number of nodes in larger graph
  - M = number of nodes in smaller graph
  - E = number of edges in smaller graph
- In practice, heuristics are used
  - Faster, but no optimality guarantee
  - I want to avoid heuristics, and parallelize instead

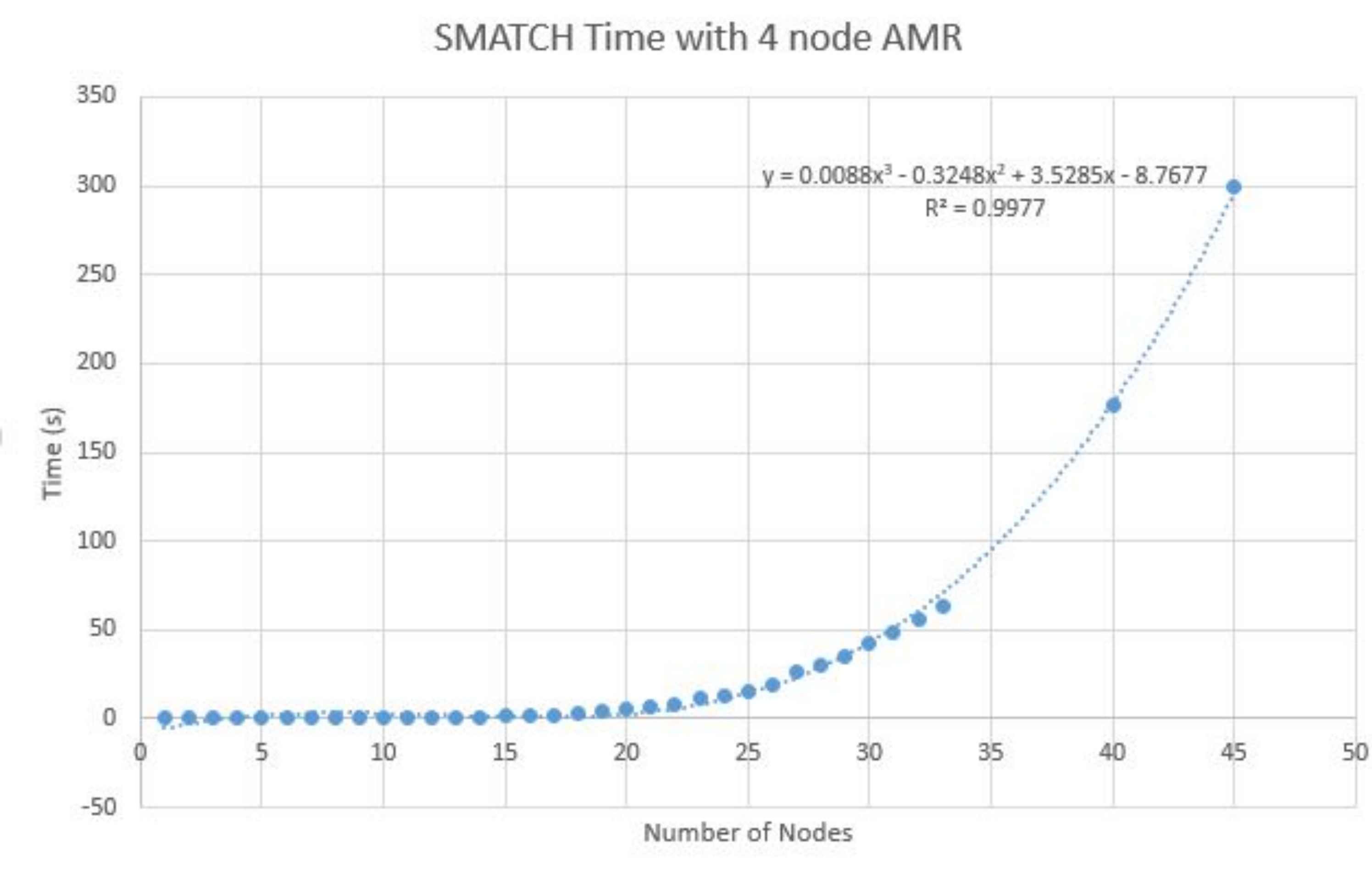
Implementation

- Python using networkX
- Just under 100 new lines (including some debugging lines)
- Highly recursive
  - Match node pair, match remaining subgraphs
  - Memory problems as problem size increases

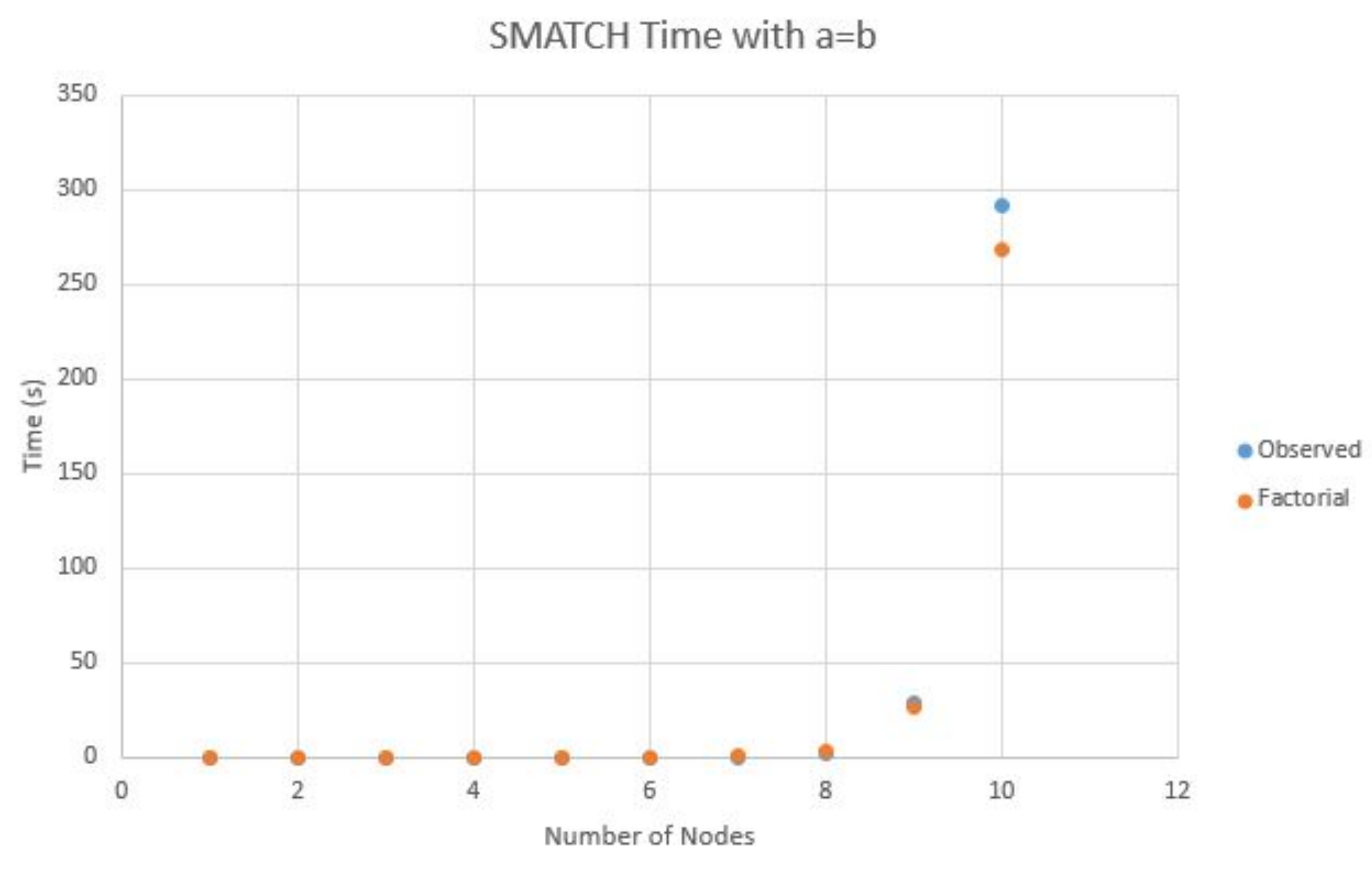
Timing Results



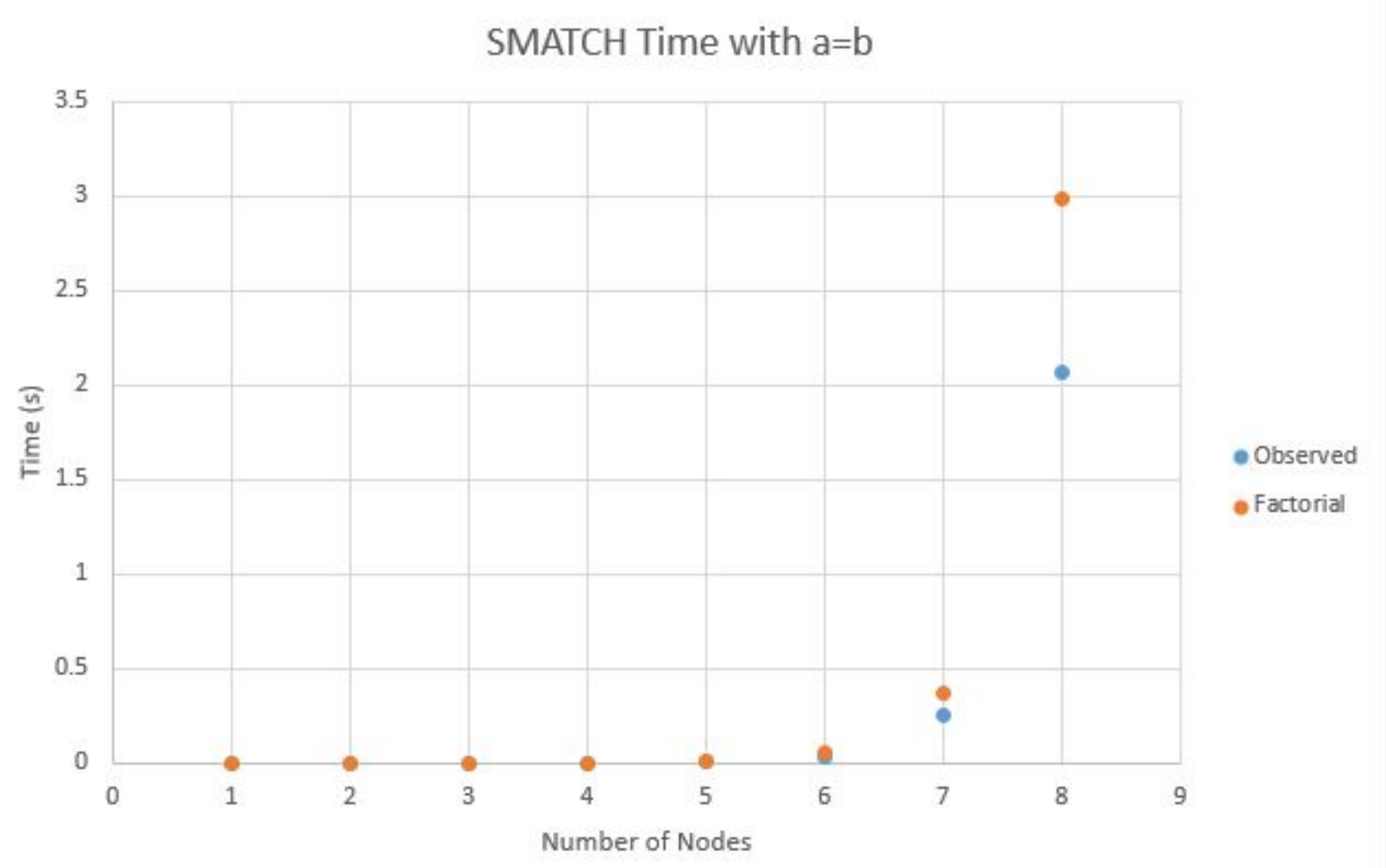
Timing Results



Timing Results



Timing Results



Other Results

- Memory consumption is high
  - At graph sizes of 11 nodes, 10 edges each memory consumption approaches 20GB
  - Memory scales similar to runtime
- SMATCH score returned is correct (optimal)
  - In some cases this is better than popular heuristic
  - Will compare against heuristic more with enhanced algorithm

Plans for Improvements

- Combine mapping and scoring
  - Score nodes as they are matched
  - Avoids recomputing
- Send subgraphs to worker machines for parallelism
- Score likely alignments first, use as cutoff
  - Denominator does not change (N+E)
  - Can avoid unnecessary computation

Try SNAP

- Interface looks very similar to networkX
- They claim it is an order of magnitude faster