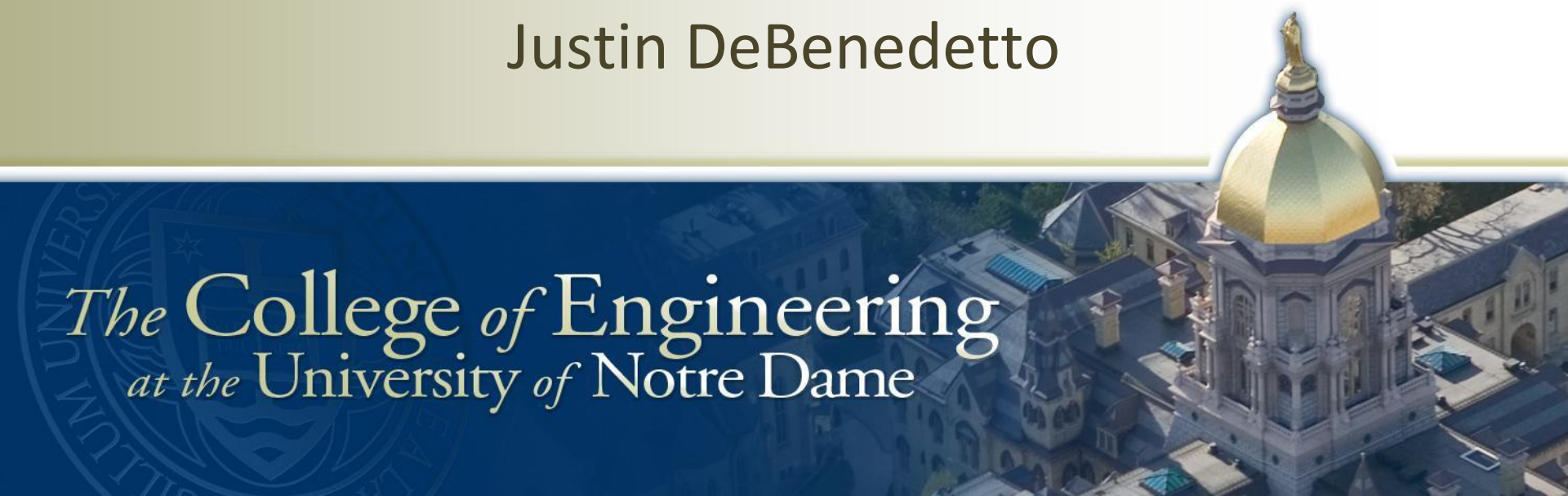
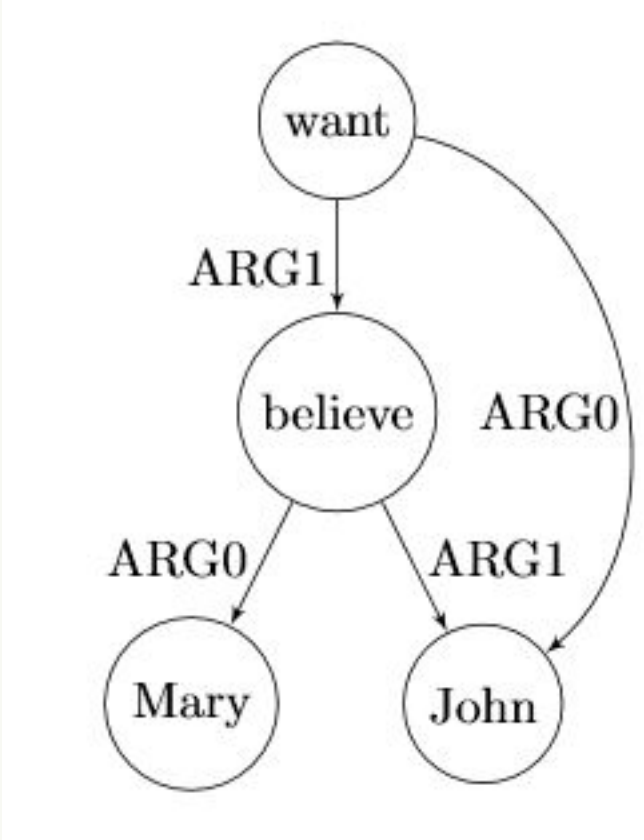
Graph Similarity Scoring
Applied to
Abstract Meaning Representation

Justin DeBenedetto



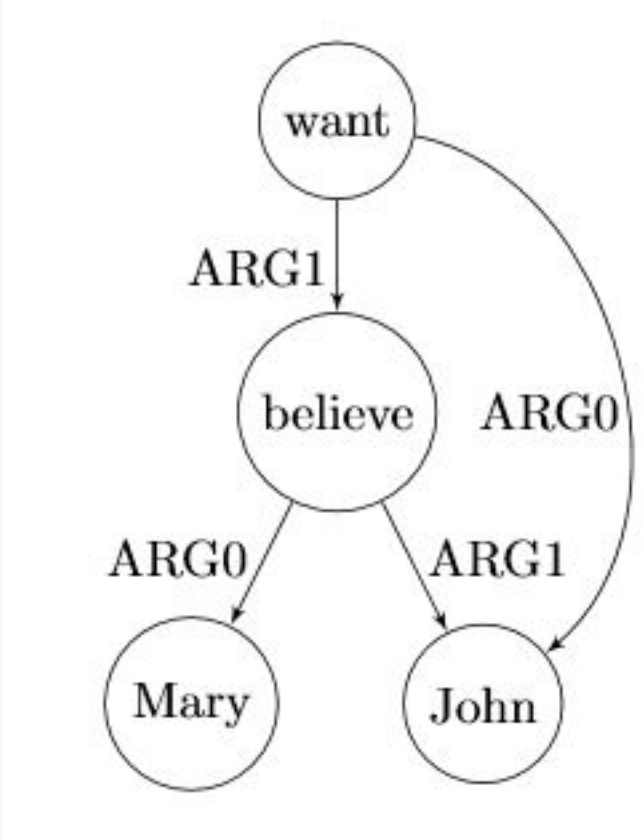
Abstract Meaning Representation (AMR)

- 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

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

Basic Implementation Pseudocode

Algorithm 1 Basic SMATCH pseudocode

```
1:  procedure GETSMATCH(A,B)
2:      maxF1 ← 0
3:      for mapping in nodeMapping(a,b) do
4:          correct ← 0
5:          for alignedPair in mapping do
6:              if labels match then
7:                  correct ← correct + 1
8:          for edges in a do
9:              replace end-points with aligned nodes from b
10:             if new edge exists in b then
11:                 correct ← correct + 1
12:         precisionDenominator ← number of triples in b
13:         recallDenominator ← number of triples in a
14:         precision ← correct/precisionDenominator
15:         recall ← correct/recallDenominator
16:         f1 ← (recall + precision)/2
17:         if f1 > maxF1 then
18:             maxF1 ← f1
19:     return maxF1
```

```
20: procedure NODEMAPPING(A,B)
21:     allAlignments ← empty
22:     Select node_a in a
23:     for node_b in b do
24:         newAlignments ← align node_a to node_b
25:         newA ← a − node_a
26:         newB ← b − node_b
27:         newAlignments ← nodeMapping(newA, newB)
28:         append newAlignments to allAlignments
29:     return allAlignments
```

Basic Implementation Pseudocode

Algorithm 1 Basic SMATCH pseudocode

```
1:  procedure GETSMATCH(A,B)
2:      maxF1 ← 0
3:      for mapping in nodeMapping(a,b) do
4:          correct ← 0
5:          for alignedPair in mapping do
6:              if labels match then
7:                  correct ← correct + 1
8:          for edges in a do
9:              replace end-points with aligned nodes from b
10:             if new edge exists in b then
11:                 correct ← correct + 1
12:         precisionDenominator ← number of triples in b
13:         recallDenominator ← number of triples in a
14:         precision ← correct/precisionDenominator
15:         recall ← correct/recallDenominator
16:         f1 ← (recall + precision)/2
17:         if f1 > maxF1 then
18:             maxF1 ← f1
19:     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
```

Find all ways to match nodes in A with nodes in B

Basic Implementation Pseudocode

Algorithm 1 Basic SMATCH pseudocode

```
1:  procedure GETSMATCH(A,B)
2:      maxF1 ← 0
3:      for mapping in nodeMapping(a,b) do
4:          correct ← 0
5:          for alignedPair in mapping do
6:              if labels match then
7:                  correct ← correct + 1
8:          for edges in a do
9:              replace end-points with aligned nodes from b
10:             if new edge exists in b then
11:                 correct ← correct + 1
12:         precisionDenominator ← number of triples in b
13:         recallDenominator ← number of triples in a
14:         precision ← correct/precisionDenominator
15:         recall ← correct/recallDenominator
16:         f1 ← (recall + precision)/2
17:         if f1 > maxF1 then
18:             maxF1 ← f1
19:     return maxF1
```

```
20: procedure NODEMAPPING(A,B)
21:     allAlignments ← empty
22:     Select node_a in a
23:     for node_b in b do
24:         newAlignments ← align node_a to node_b
25:         newA ← a − node_a
26:         newB ← b − node_b
27:         newAlignments ← nodeMapping(newA, newB)
28:         append newAlignments to allAlignments
29:     return allAlignments
```

Check if node labels match and if edge labels match

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

Improvements

- Combine mapping and scoring
  - Score nodes as they are matched
  - Avoids recomputing
- Score likely alignment first, use as cutoff
  - Number incorrect is cutoff threshold
  - Can avoid unnecessary computation
- Send subgraphs to worker processes for parallelism

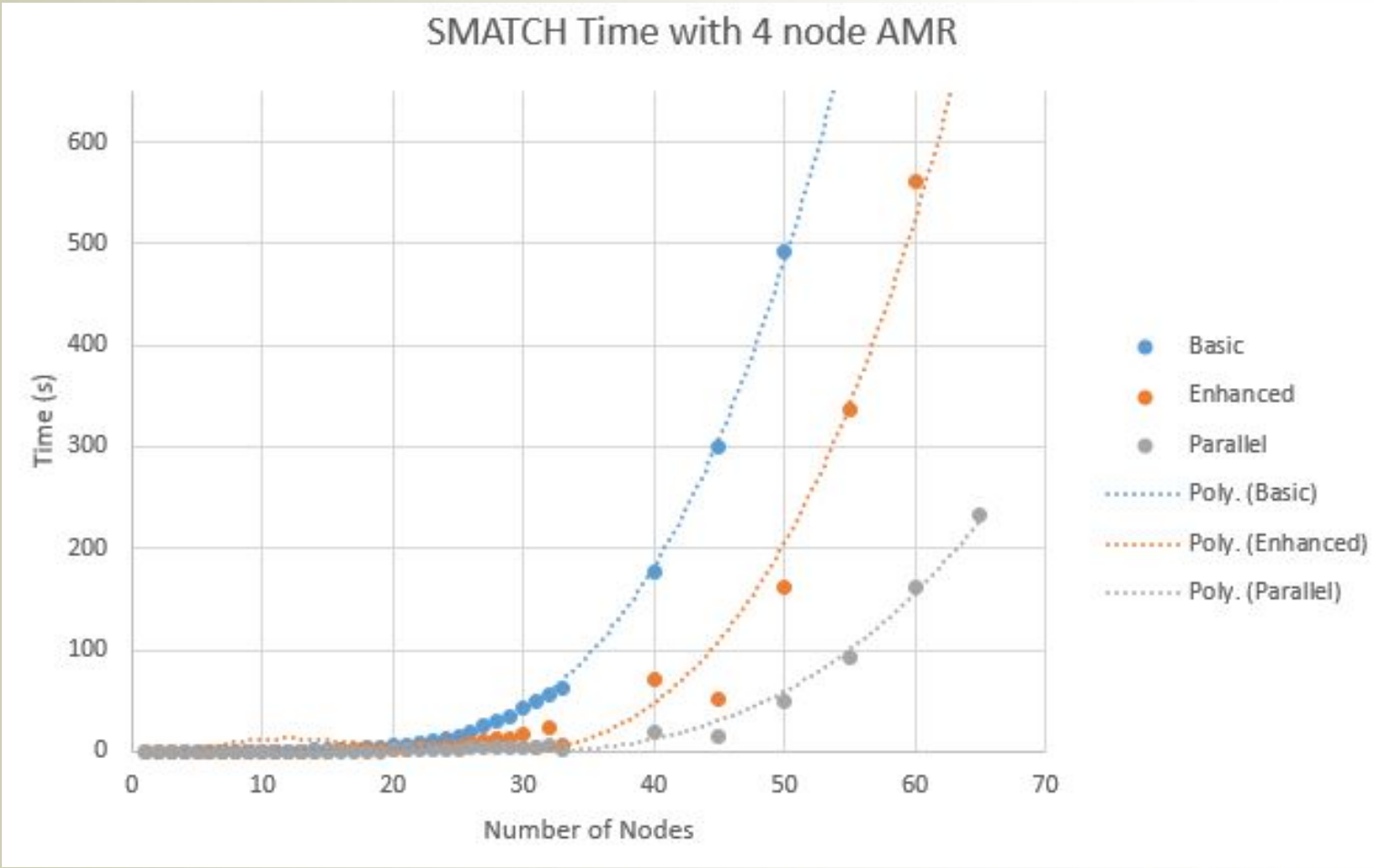
New Complexity

- Previously ~O(N!/(N-M)!*|M+E|)
- Now ~O(N!/(N-M)!*|E|) in worst case, ~O(N!/(N-M)!) in average case
- Worst case complexity not improved greatly
- Better in practice using cutoff to eliminate parts of search space
- Effectiveness increases as SMATCH increases

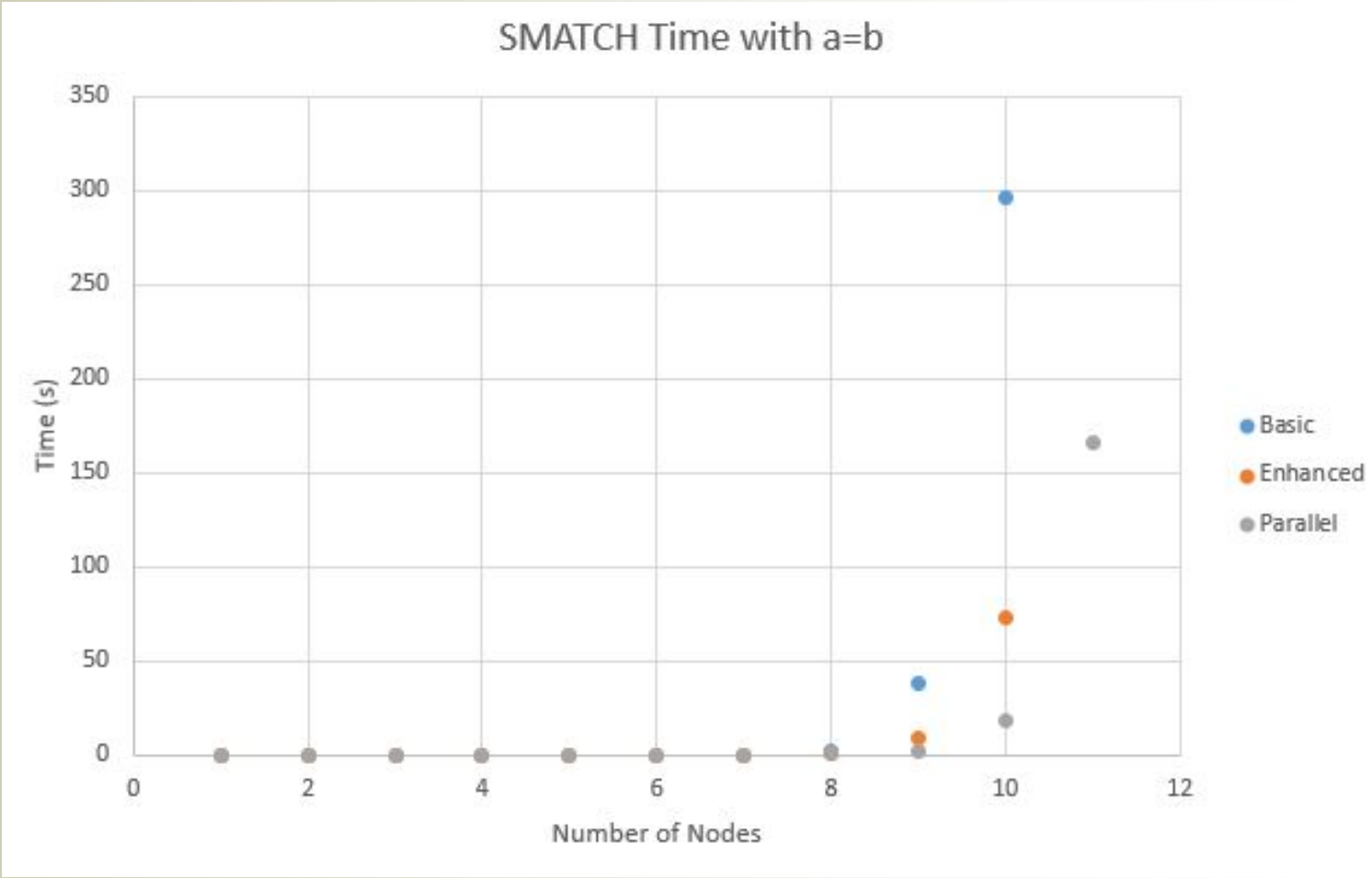
Implementation

- Implemented in Python 2.7
- Uses NetworkX
- Uses Multiprocessing library
- ~500 lines of code (separate functions for basic vs enhanced vs parallel, so some repetition)

Timing Results



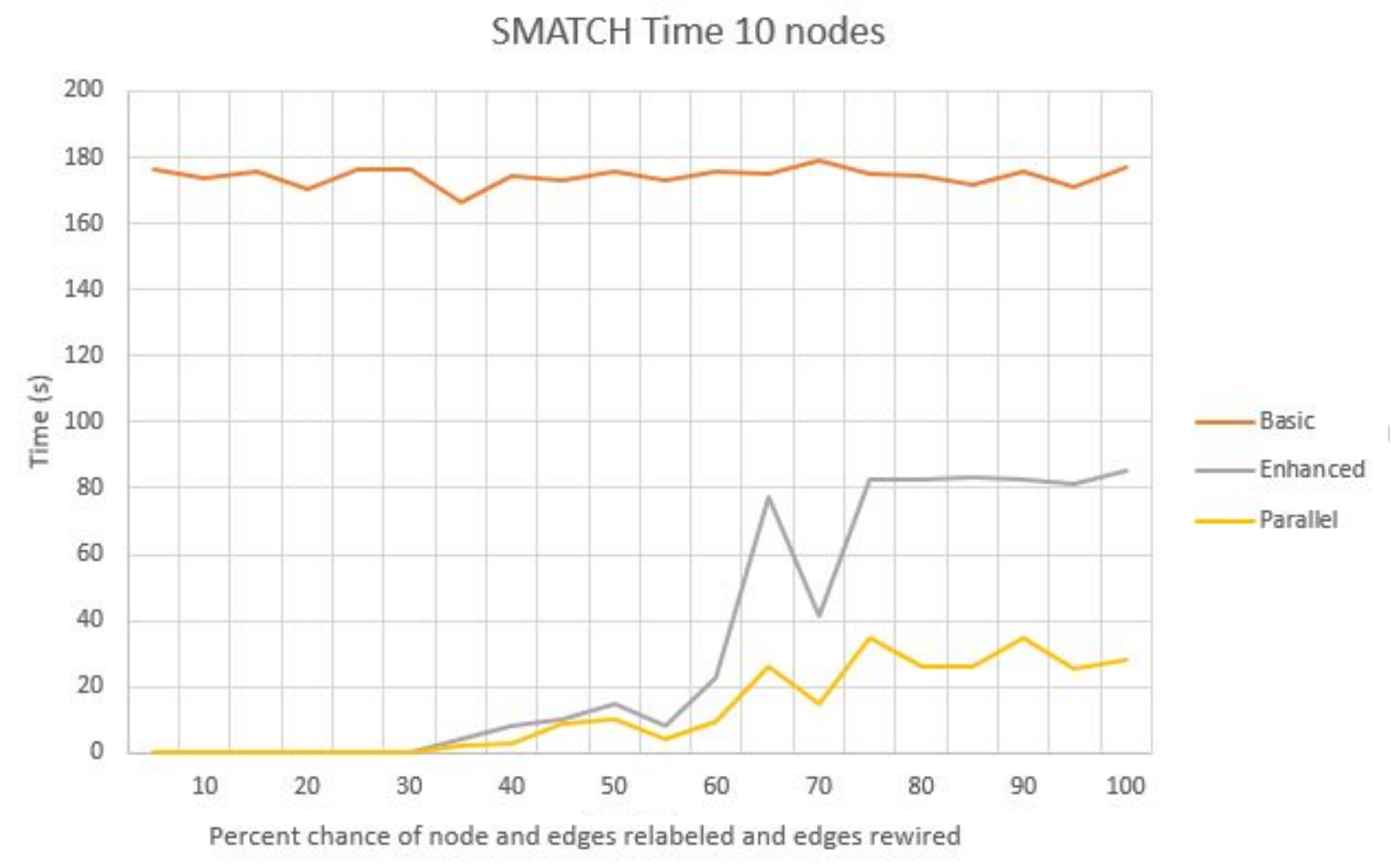
Timing Results



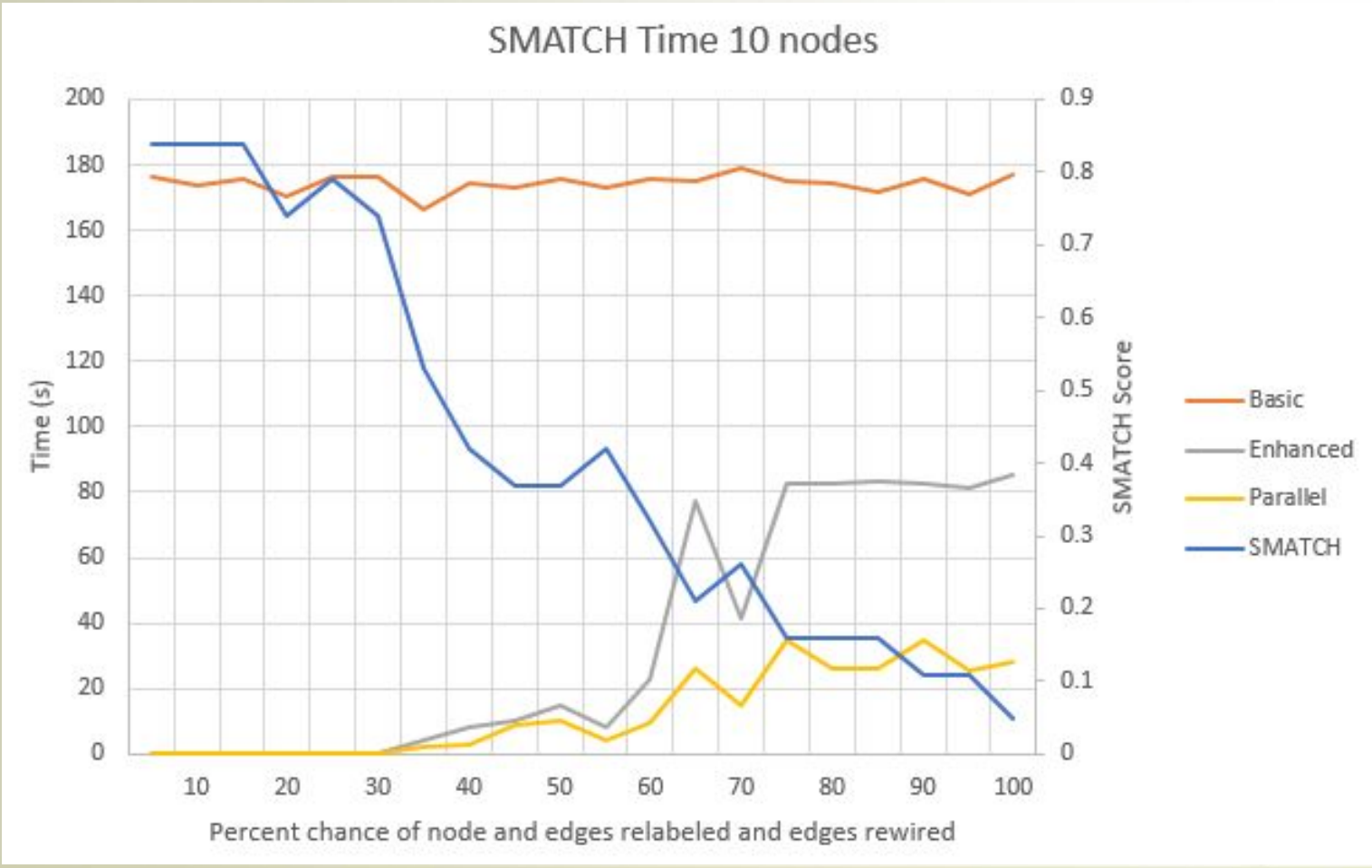
Generating Candidate AMRs

- In practice we use SMATCH on AMRs that are similar
- Imitate this by randomly rewiring edges, relabeling edges, and relabeling nodes

Timing Results



Timing Results



Conclusions

- While the worst case complexity remains bad, typical use can be made much better
- Prune search space by not pursuing bad subgraphs
- Parallelize subgraph search
- Most effective when candidate is close to correct (high SMATCH)