Practical Heuristics for Inexact Subgraph Isomorphism

Jon Berry
Scalable Algorithms Department
Sandia National Laboratories

July 12, 2010
SIAM Annual Meeting

Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company,
for the United States Department of Energy's National Nuclear Security Administration
under contract DE-AC04-94AL85000.

“Subgraph Isomorphism” -- Loosened

- **Objective: find exact or inexact matches of a small pattern graph within a large semantic graph**

- **Complexity of exact isomorphism problem: NP-complete**
  - *however, having vertex and edge types helps in practice if not in theory*

- **Our contribution**
  - We describe a generic heuristic method to find inexact matches in semantic graphs
  - We demonstrate this method on a specific test case: finding chordless 5-cycles in large graphs
  - This is available in open-source C++ code

Subgraph Isomorphism Heuristic: Input

The Target Graph:

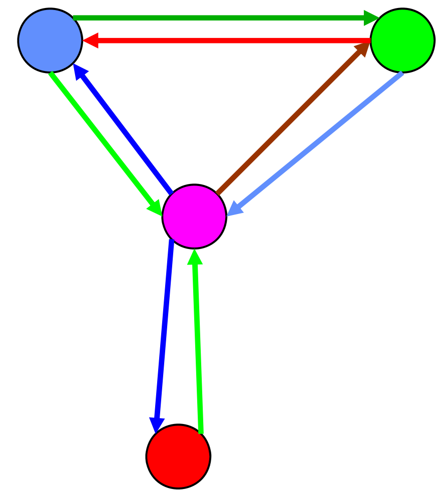
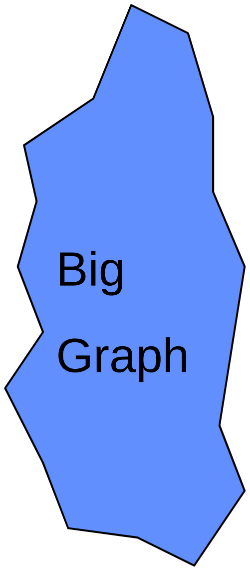


Table of Type and Auxiliary Information:

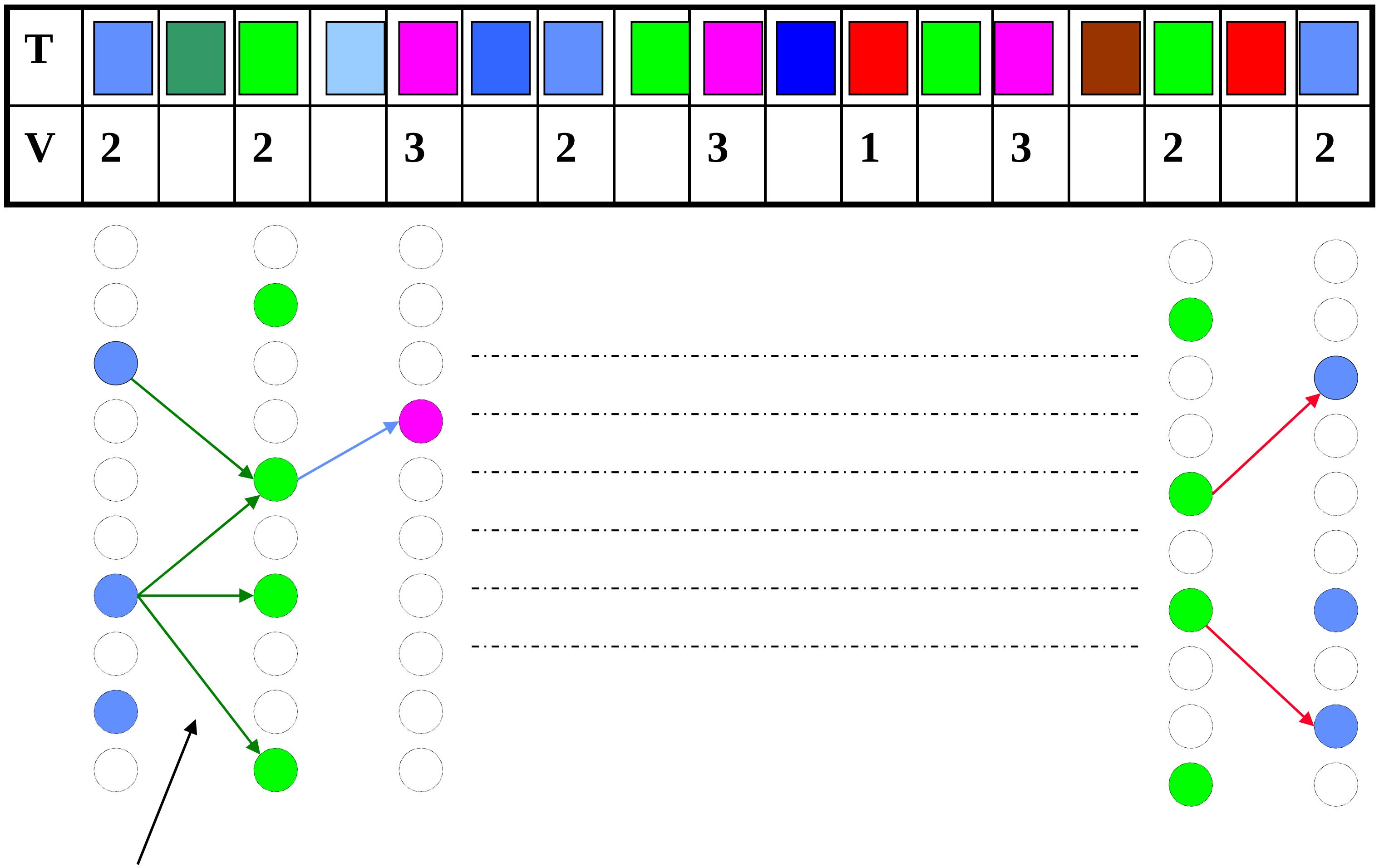
T																	
V	**2**		**2**		**3**		**2**		**3**		**1**		**3**		**2**		**2**

Walk description: e.g.: Euler Tour (or concatenations of these)

Encapsulate Matches in a Directed Bipartite Graph (B)



k times:
visit each edge

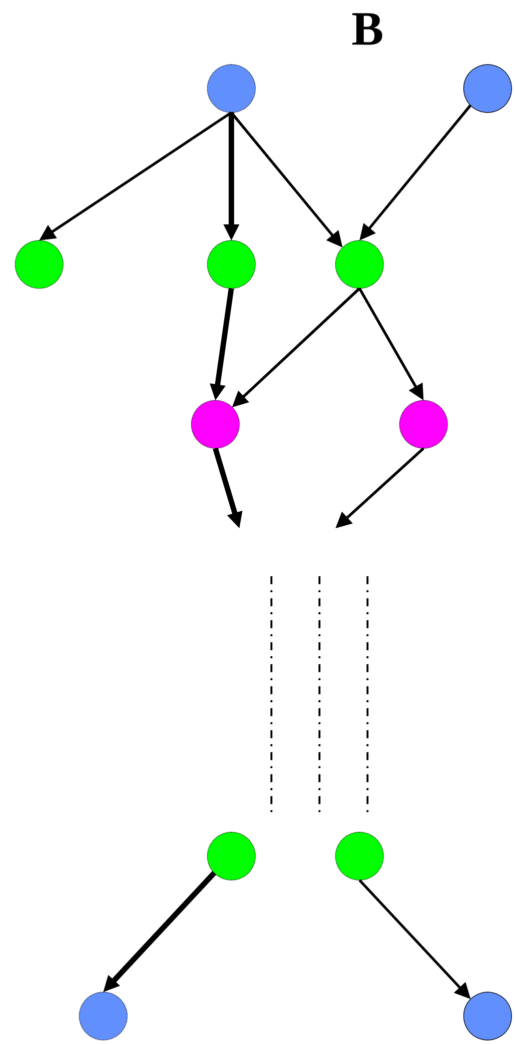


A user-provided comparator determines whether edges match

(the default is exact vertex and edge type matching)

The Bipartite Graph Representation

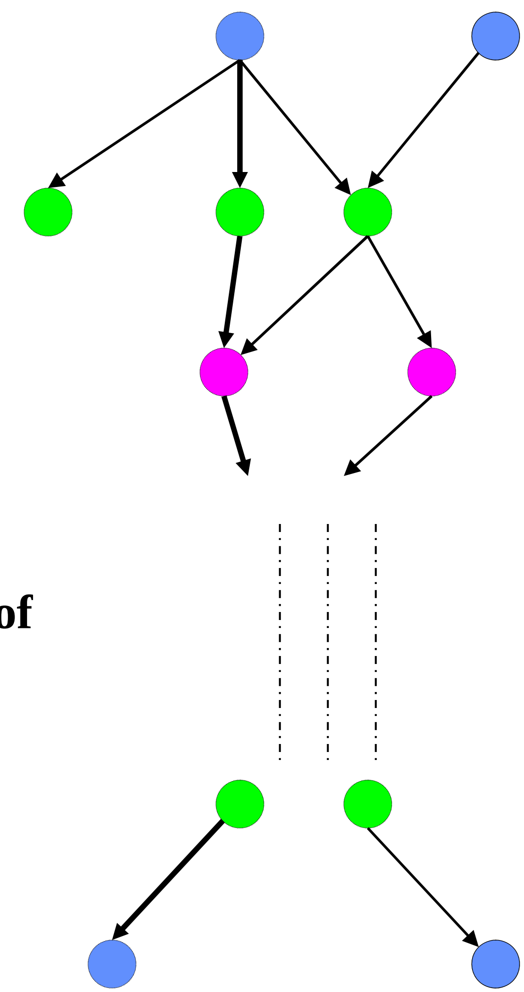
- **Any path from the top to the bottom represents a *type-isomorphic walk* through the original graph**
- **Any possible exact match is represented**
- **Many inexact matches are also represented**
- **The number of candidate matches can be large; this depends on the vertex and edge attributes**
  - *Richness*: is the set of types large? Well distributed?
  - *Strength*: does leveraging the types significantly reduce the size of the bipartite graph and/or the number of type-isomorphic paths?



How do We Explore the Set of Paths?

- ***Idea:* branch and bound**
  - *Issue:* what's the bound? – need graph distance between subgraph and graph
- ***Idea:* find connected components of B**
  - *Issue*: giant component phenomenon still applies
- ***Idea:* use augmenting paths in maximum flow**
  - *Issue:* need formalism to constrain augmenting paths

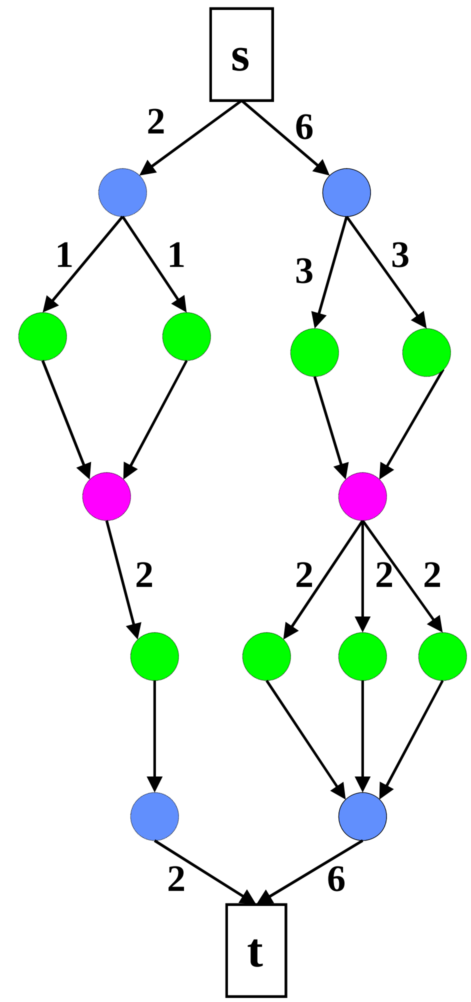
- **Our current approach (inspired by randomized rounding of linear programming solutions):**
  - Compute a *special betweenness centrality* in O(n+m) time
  - Take guided random walks
  - "Visit" each walk with a user evaluation function



Sandia National Laboratories

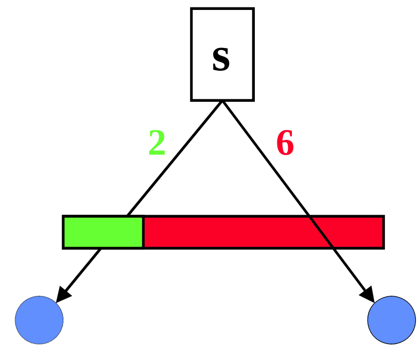
Weighting the Edges of B

- **Normal "betweenness centrality" is super-quadratic**
  - For each vertex *v* (or edge *e*):
    - For all pairs of vertices (s,t), *s* and *t* are not *v*:
      - Compute the proportion of (s,t) shortest paths that go through *v* (or *e*)
- **Simplification:**
  - We care about only one (s,t) pair: a super-source and a super-sink
  - One forward BFS and one backward BFS will yield this simplified betweenness centrality
  - Each edge will be weighted with the number of type-isomorphic (s,t) walks through it



Embarrassingly Parallel Random Walks

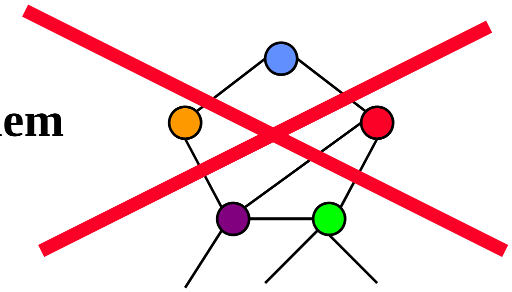
- **Random (s,t) walk**
  - At vertex *v* of out-degree *k*, consider a weighted, k-sided die
    - The weights are the betweenness values of *v*'s out-edges
    - Roll the biased die to choose a downward path
- **Let *w_s* be the sum of the weights of the out-edges of *s***
  - Take *w_s* walks to visit a good proportion of the candidates
  - Take these walks independently, and in parallel
  - Each walk defines a candidate match: *apply a user-provided evaluation function to decide acceptance*



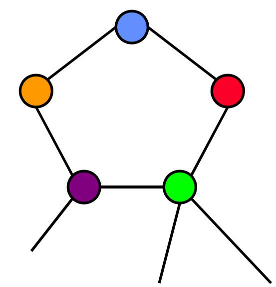
"I refuse to be embarrassed by embarrassing parallelism" – C. Phillips, 2009

Case Study: Chordless 5-cycles

- **Our heuristic would work best on semantic graph problems with a rich set of types (e.g. RDF data)**



- **However, we'll show how to use it on a more basic problem**
  - Find all 5-cycles such that
    - Each of 5 vertex types is represented (in a given order)
    - There are no chords

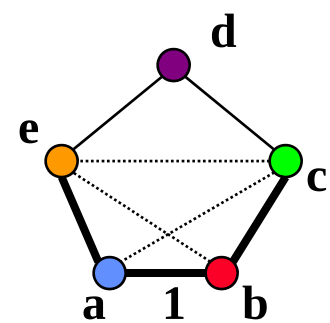


- **The brute force algorithm runs in O(n^5) time**
- **Our heuristic runs in O(n + mD^2 + E[#paths in B])**
  - D is the maximum degree of a cycle vertex we wish to check
  - That last term will be heavily dependent on our strategy to define edge types

The Key to Success

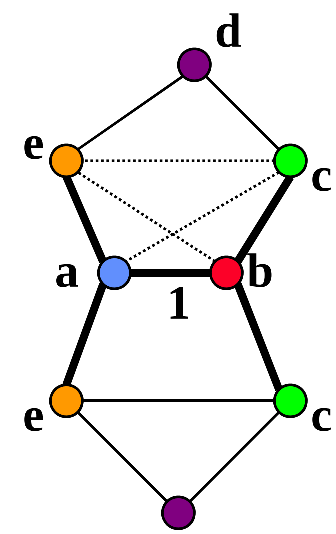
- ***If the vertex and edge attributes aren't strong enough to cut down the search space, define and use derived features/attributes***

- **In our example, we'll define the "bowl-ness" of an edge (idea credit: Cynthia Phillips)**
  - Consider the "bowls" of an edge (e.g. (a.b))
  - Make sure that one exists
  - Make sure that (b,e), (a,c), (c,e) don't exist
  - Make sure that type sequence is feasible
  - If there is an ok bowl, edge type is 1; else 0
  - This is a local one-neighborhood operation: O(D^2)



Evaluating Candidates

- **Each candidate match is evaluated as follows, via a "visitor" method**
  - We know that the type sequence will be correct, but did we touch exactly 5 vertices?
  - Are there no chords? A cycle of 1-edges still might have them, so we double check.



The Experiment

- **Coded using the MultiThreaded Graph Library (MTGL) subgraph isomorphism framework**
  - Open-source: https://software.sandia.gov/trac/mtgl
  - There are tutorials on the basics and the subgraph isomorphism code
  - Implementation, tutorial credit:
    - Greg Mackey, lead developer of the MTGL
- **Tested on "R-MAT" graphs**
  - Chakrabarti, Zhan, Faloutsos, 2004
  - Parameters (0.57, 0.19, 0.19, 0.05) – simulate a power-law degree distribution
- **Run on a Linux workstation**
  - Same code could run on multicore or the Cray XMT
  - We select D=20, ignoring 5-cycles involving high-degree vertices

The Results

n	m	#(s,t) paths in B	mD^2	Brute force operations	Chordless 5-cycles found
4096	31917	3940	~12M	~10^15	1
16384	129302	97929	~48M	~10^20	13
65536	521108	428373	~200M	> 10^20	25

Conclusions

- **We've taken a heuristic intended for semantic queries and applied it to a more fundamental problem – with success**
- **The running time is dominated by precomputation of edge weights and by taking random walks through B – both parallelize**
- **With tuning, the code can run on the Cray XMT, but there wasn't time to do this**
- **When the problem is more semantic (there are more types), the heuristic is stronger**
- **User codes the comparator for building B and the visitor for evaluating candidates**
- **Sometimes it's necessary to exploit structure in the target graph**