Solution Sketches
BGL Problems

Andreas Bärtschi
based on slides by Rasťo Šrámek

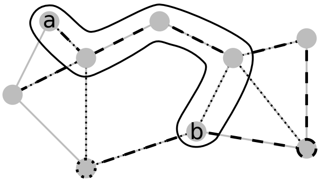
ETH Zürich

December 16, 2015

Ant Challenge

Approach:

- Compute minimum spanning trees for each species.
  Prim or **Kruskal** work: Edge weights are pairwise distinct, hence MST is unique.
- Keep track of the minimum weight for each edge (optional).
- Dijkstra's shortest path on graph with minimal edge-weights (or on graph with multi-edges with weights defined by each species's MST).




```
// s = number of species
// t = number of trees
// e = number of edges
vector<pair<int, int> > edges(e);
vector<vector<int> > weights(s, vector<int>(e));
// create Graph to compute Species i's MST
Graph G(edges.begin(), edges.end(),
        weights[i].begin(), t);
```

Important Bridges

Problem:

- Find bridges in a graph.
- Solution 1: **Biconnected Components**
  Look for components of size 1: Those contain the bridges.
- Solution 2: Implement pre-order DFS, number vertices based by exploration time, keep track of the lowest ID a child found.
  (Tarjan's bridge-finding algorithm)

Caveats:

- Don't overcomplicate things.
  (Don't use unnecessary library parts, e.g. BGL's articulation points.
  If you do, make sure your algorithm also solves bordercases correctly.)
- Output the bridges correctly sorted!

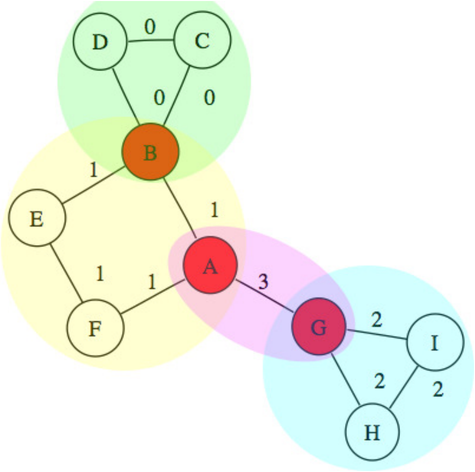
Important Bridges – biconnected components

Carefully read the **BGL manual**:

biconnected_components(g, component)

“A connected graph is biconnected
if the removal of any single vertex
(and all edges incident on that vertex)
can not disconnect the graph.”

Here there is exactly one bridge.



Buddy Selection

Problem

n students (n even), each has c hobbies. Check if we can match them in pairs so that each pair shares at least f hobbies.

Model

Straightforward (if you see it): **perfect matching** in an *unweighted* undirected graph. Make an edge between two students, if they share at least f hobbies.

Matching is easy to compute once the graph is constructed, edmonds_maximum_cardinality_matching is $O(mn\alpha(m, n))$.

Note that it is not necessarily a bipartite matching.

Buddy Selection – graph construction

"Reasonable" brute force:

```
foreach student-pair {u, v}
   count = 0
   foreach u-hobby: a
      foreach v-hobby: b
         if a == b
            ++count
   if count >= f
      add_edge(u, v)
```

$O(n^2 c^2)$, still fast enough

Faster:

```
foreach student u
   sort hobbies of u
foreach student-pair {u, v}
   S = intersection of u- and v-hobbies
   // S computed by lineartime sweep.
   if |S| >= f
      add_edge(u, v)
```

$O(n(c \log c) + n^2 c) = O(n^2 c)$

Coin Tossing

Problem

Input:
1) Sequence of m games, some with known result, some without.
2) $s_1, \ldots, s_n$ – number of won games for each player.

Output:
Check if possible to assign outcomes to unknown games and get the standings.

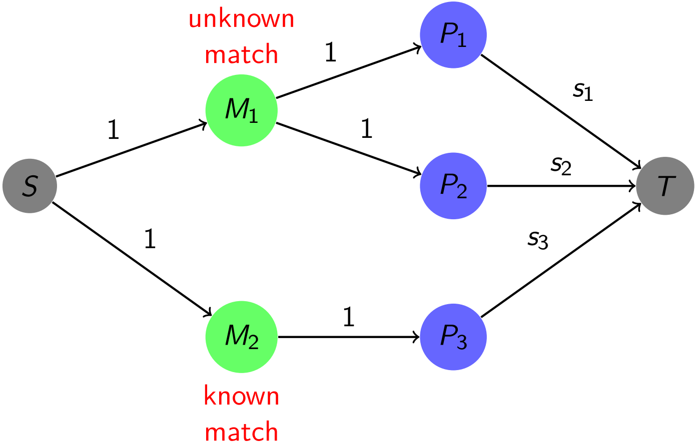
Solution model

Graph flow, vertices: source, m game-vertices, n player-vertices, sink.

Coin Tossing – model

Create game-vertices as in the tutorial problem *Soccer Prediction*.

$1-1$ correspondence between (integral) flows of value m and allowed solutions.



Satellites

Problem

Straightforward from the statement:
Output **Minimum Vertex Cover** of an unweighted bipartite graph.

Model

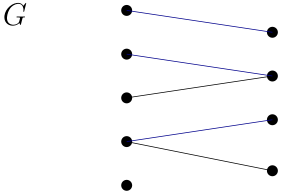
König's Theorem: $|\text{Minimum Vertex Cover}| = |\text{Maximum matching}|$.
Modified BFS to find the vertex cover.

Downfalls

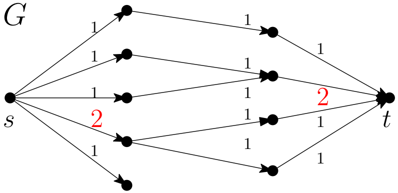
Don't write your own L-R switching BFS! (If you do: matching edges oriented backwards, non-matching edges oriented forwards.)

Model the bipartite matching as a flow problem.
Run the simple BFS on the residual graph which we provided on Moodle.
Create the correct flow graph!

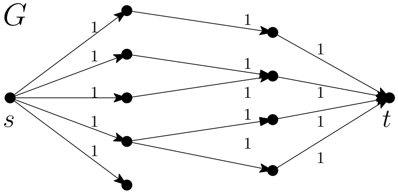
Satellites – flow graph



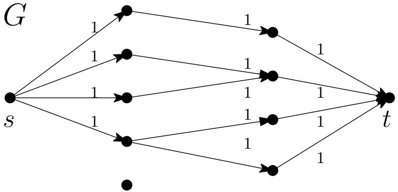
Matching problem.



This doesn't work.

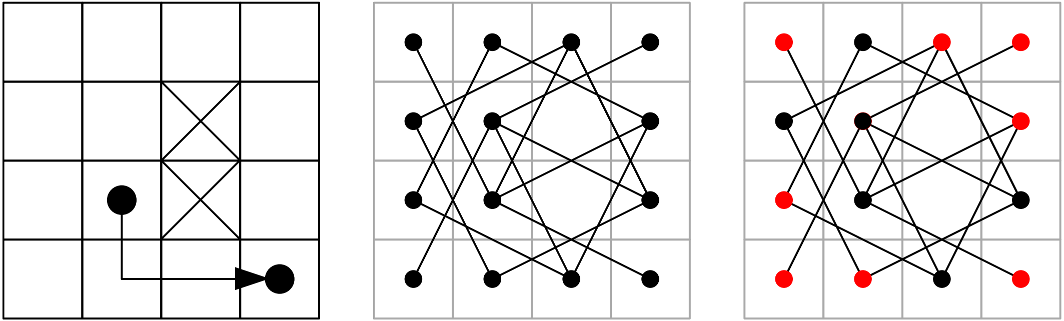


This is the correct flow graph.



This doesn't work.

Placing Knights



Connect all *present* chessboard fields with an edge. The maximum number of non-threatening knights corresponds to the **Maximum Independent Set** of this graph. But MaxIS is NP-hard in general!

Placing Knights – bipartite graph

$|\textbf{Maximum Independent Set}| \;=\; n - |\textbf{Minimum Vertex Cover}|.$ [BGL week 3]

König's Theorem: In bipartite graphs, the size of a minimum vertex cover equals the size of a maximum matching.

Use chessboard coloring to see that the graph is indeed bipartite!