Two algorithms for finding edge colorings in regular bipartite multigraphs

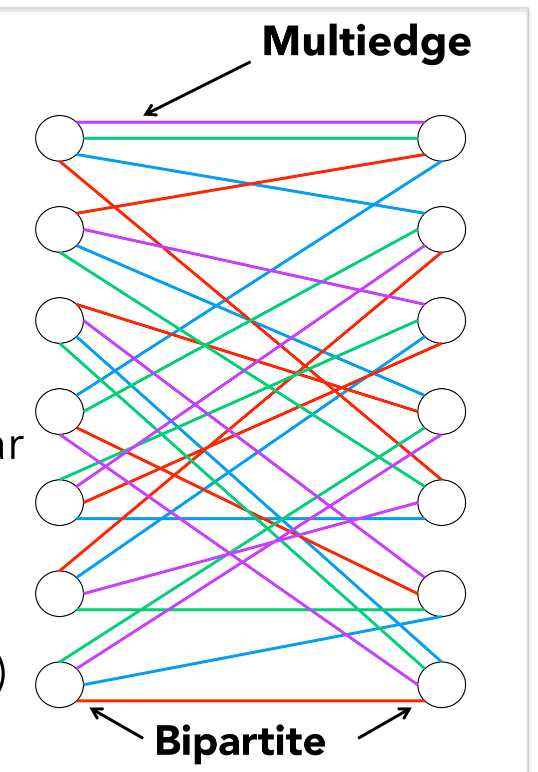
Patricia Neckowicz, Thomas Cormen
Department of Computer Science, Dartmouth College

Definitions

A **regular bipartite multigraph** G with degree 4 (d=4).

Edge coloring: The edges of G are colored with d colors such that each vertex has exactly one incident edge of each color. An edge coloring of a regular bipartite multigraph is equivalent to finding d perfect matchings.

Goal: Find an edge coloring in $O(E \lg d)$ time. E is number of edges.

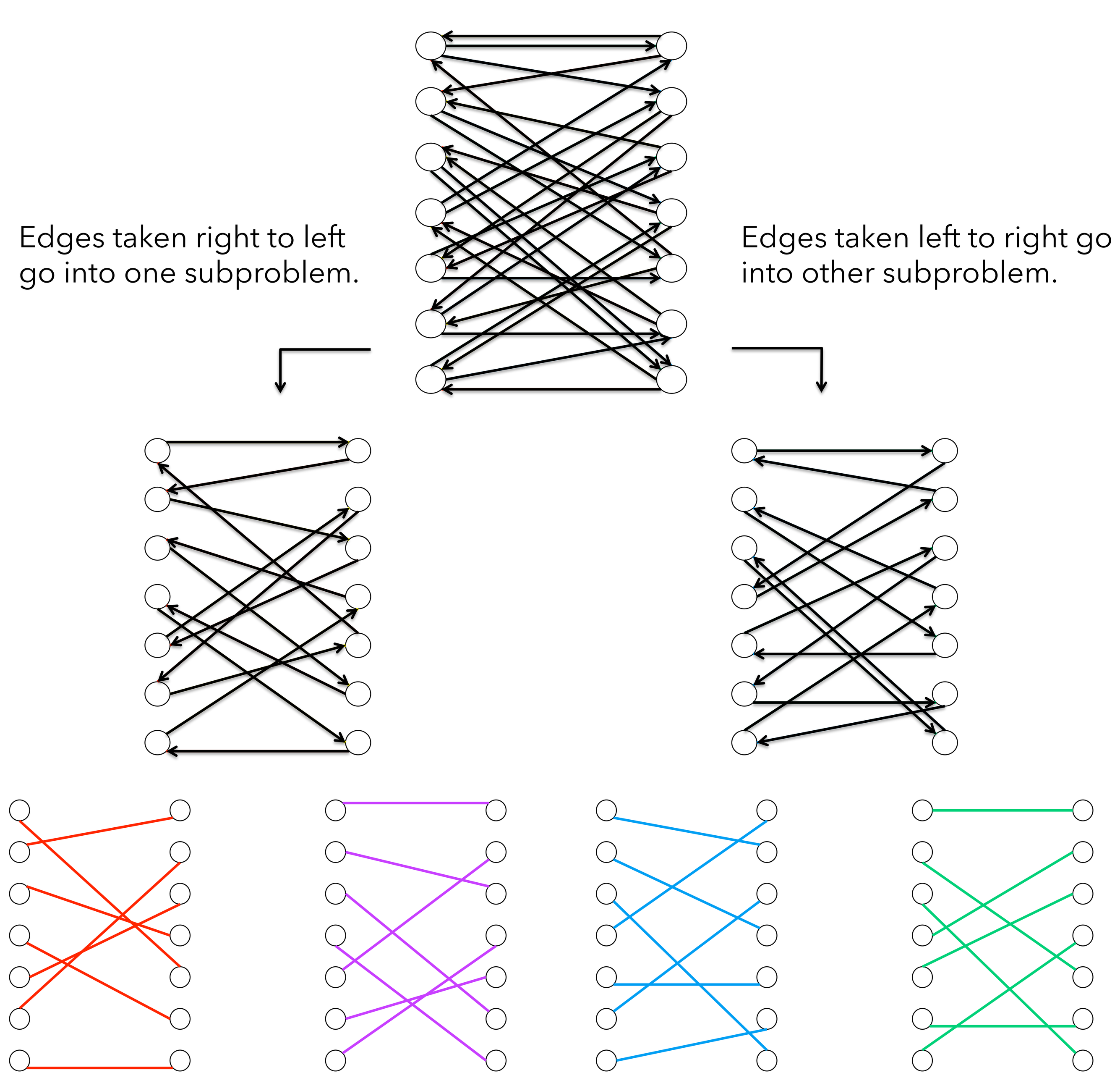


Divide-and-conquer

If degree d is even, split into two d/2-regular subgraphs.

- Trace out cycles.
- Edges taken from left to right go into one subgraph.
- Edges taken from right to left go into other subgraph.
- Once d = 1, assign the same color to each edge in the subgraph.

We can keep applying this idea only when d is a power of 2.



Our approach

When d is not a power of 2, we cannot easily perform the edge coloring using the divide-and-conquer approach. Want to make degree even in $O(E)$ time.

- Previous approach: Find a perfect matching and remove.
- Our approach: *Add* a perfect matching to the graph (we call these edges **dummy edges**) to make d even.

Additional constraint:
When tracing cycles, all dummy edges must be traversed in the same direction.

We add dummy edges in two ways:

- **Static:** Add dummy edges and then trace cycles.
- **Dynamic**: Trace paths from left to right and add dummy edges to close paths.

Static dummy edges

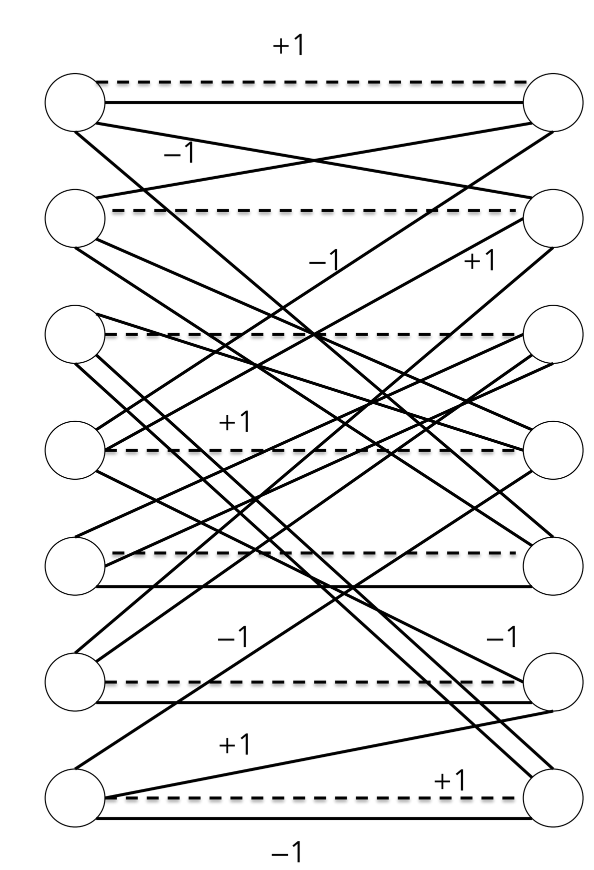
Main idea:

- Add dummy edges straight across. Require that they are taken from left to right.
- While there is still a dummy edge $e = (l, r)$ that has not been taken:
  - Take e from left to right
  - Search for a path from r back to l. Can "untake" edges that have already been assigned directions.
  - Commit the edges in this path to a cycle.

The figure shows one iteration of the while loop. Dashed lines represent dummy edges. Every edge is assigned a number: –1, 0, or +1.

- –1 means been taken right to left.
- +1 means been taken left to right.

Edge directions can be untaken, resulting in 0 edges. Once a dummy edge has been taken, it cannot be taken or untaken. Once each dummy edge has a +1, put –1 edges into one subproblem and +1 edges into the other. Each vertex has even degree in remaining graph of 0 edges, so easily trace cycles.



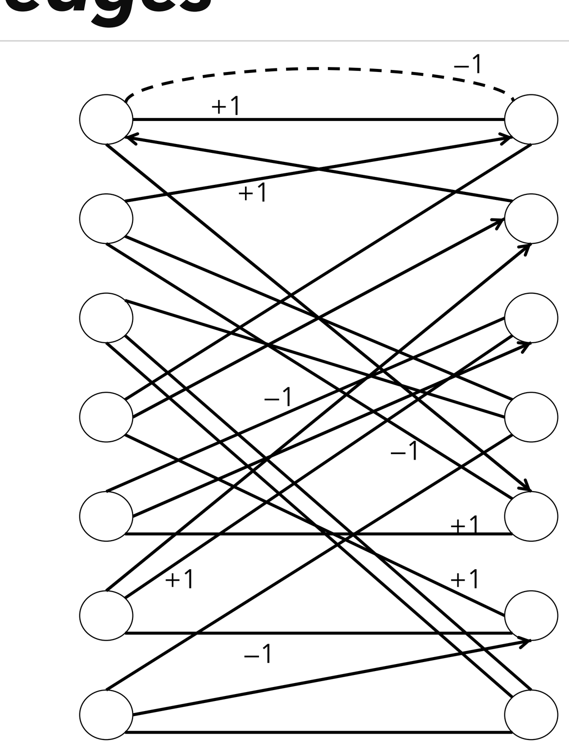
Dynamic dummy edges

Main idea:

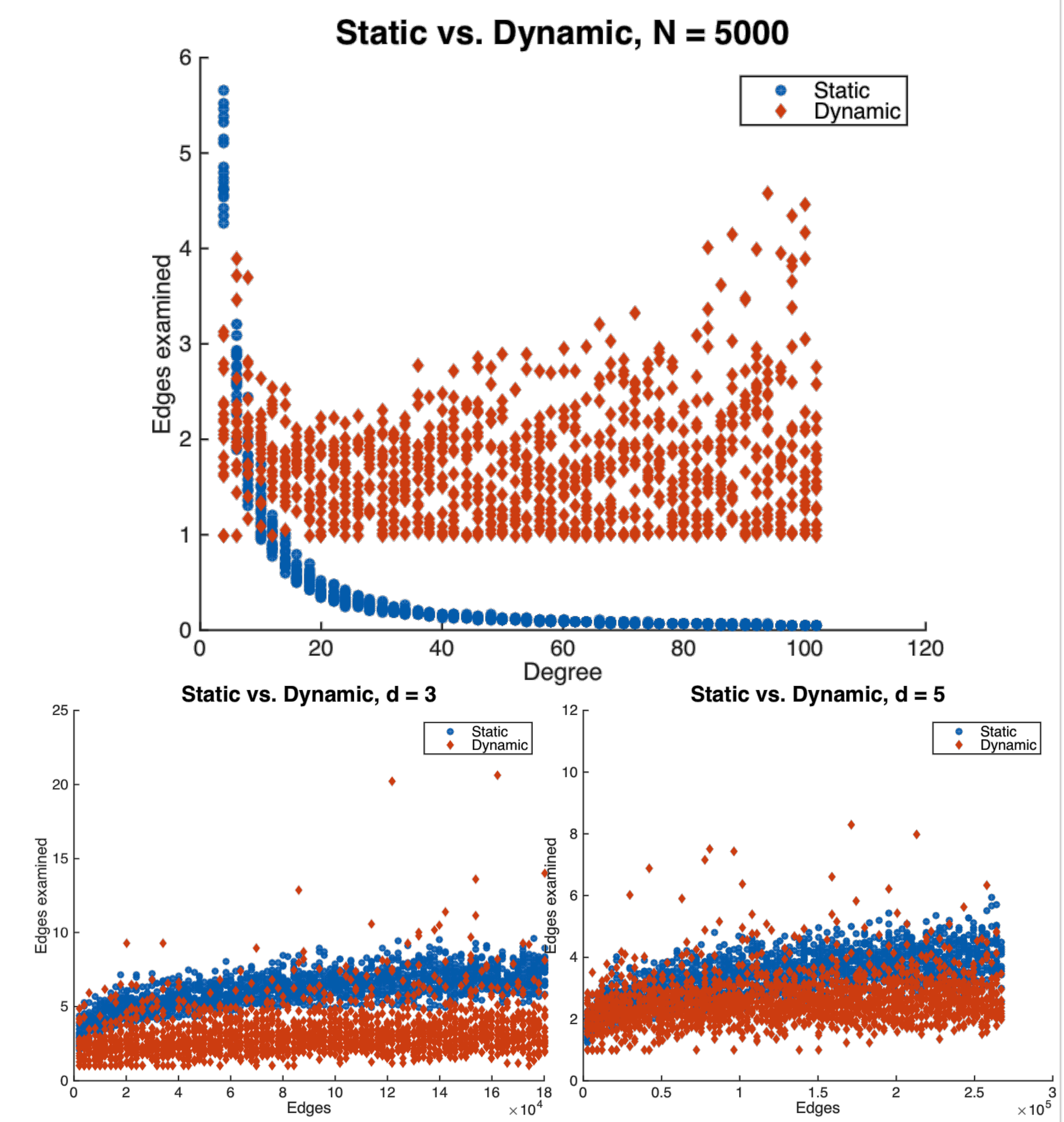
- While there is a left vertex with no incident dummy edge
  - Start from such a left vertex l and **walk** from l to a right vertex r. We determine r by "getting stuck": getting to a right vertex that we cannot leave because all incident edges have been taken.
  - Add a dummy edge e=(l,r). Take e from right to left to close a cycle.

The figure shows one iteration of the while loop, using the same notation as above.

Additional heuristic: Direct edges such that each vertex has $\lfloor d/2 \rfloor$ edges going from right to left and $\lceil d/2 \rceil$ edges going from left to right.



Results



The dynamic strategy visits fewer edges for graphs with degree 3. For higher odd-degree graphs, the static strategy visits fewer edges.

Conclusions and acknowledgments

Previous attempts to solve the edge-coloring problem have failed to prove the running time bound of $O(E \lg d)$ or have yet to be implemented. Although we do have an implementation with promising results, we do not yet have a proof that the algorithm runs in $O(E \lg d)$.

I would like to thank my advisor Prof. Tom Cormen for his ideas and guidance over the past year.