An Algorithm for Finding Edge Colorings in Regular Bipartite Multigraphs

Patricia Neckowicz

Consider the problem of performing an edge coloring of a regular bipartite multigraph. For a d-regular multigraph $G = (L, R, E)$, an edge coloring is equivalent to a decomposition of E into d perfect matchings. One algorithm for identifying these d perfect matchings uses a divide-and-conquer approach. The edge set E is split into two sets E_1 and E_2 such that the multigraphs $G_1 = (L, R, E_1)$ and $G_2 = (L, R, E_2)$ are $d/2$-regular. This partition, called an *Euler partition*, is formed by tracing out cycles using all edges in E and splitting those taken left to right from those taken right to left. Because the resulting multigraphs are now smaller instances of our original problem, we can recurse. Our base case is $d = 1$, where the remaining edges form a perfect matching.

If the degree of the multigraph is a power of 2, we can repeatedly perform Euler partitions to find all d of these matchings. However, when d is not a power of 2, eventually a partition will result in a multigraph of odd degree, and we are no longer guaranteed the decomposition into cycles. If a perfect matching is removed from the edge set in $O(E)$ time (giving us even degree), we can achieve the best-case runtime of $O(E \log d)$ for this problem. Many attempts at identifying this perfect matching in $O(E)$ time have been overly complicated or have failed to prove the runtime bound. I would like to consider a different approach.

Instead of removing a perfect matching to achieve an even degree, we can add a perfect matching. Let us call the added edges *dummy edges*. Let us assume for now that all dummy edges are included in the same subproblem after an Euler partition is formed; we will see why this property is important. Each of the two subproblem graphs will fall into one of four cases after the partition: (1) an even-degree graph with no dummy edges, (2) an odd-degree graph with dummy edges, (3) an odd-degree graph with no dummy edges, and (4) an even-degree graph with dummy edges. In case (1), we can perform Euler partitions with no constraint. In (2), we can remove the dummy edges to get case (1). In (3), we are left with our original program, so we add dummy edges to achieve even degree, arriving at case (4). We see that in case (2), we can remove the dummy edges because they were included in the same subproblem of the Euler partition; if we had not required this property, the subproblems with some of the dummy edges would be meaningless, as they could not be decomposed into some original (non-dummy) perfect matchings. Therefore, we will require that dummy edges are placed in the same subproblem and enforce this via our Euler partition, by traversing all of the dummy edges in the same direction (without loss of generality, we can choose left to right). If we can satisfy this constraint and perform the partition in $O(E)$ time, we will achieve the best-case runtime. One strategy is presented below.

while *there is a dummy edge that has not been taken* **do**
    take a dummy edge (s, e) from left to right;
    perform a search from e to s to close the cycle, without taking a dummy from right to left in the process;
end
form cycles from the remaining edges;
split the edge set into edges that were taken left to right and edges taken right to left;

During the search step, we can apply various heuristics and strategies in an attempt to achieve the desired runtime of $O(E)$. These heuristics include, but are not limited to, (1) when on the left, taking a dummy edge if possible, and (2) when on the right, taking edges that bring you to unused dummy edges. However, these strategies alone are not sufficient to solve this problem, as we will not always be able to complete a cycle. It is possible to get stuck on the right, where the only available edge from right to left is a dummy edge. I expect that the majority of my work will be experimenting with various solutions to the specific problem of getting stuck. In the process, I hope to prove some useful results concerning the runtime of these approaches.