5-28-2014

Chain Match: An Algorithm for Finding a Perfect Matching of a Regular Bipartite Multigraph

Stefanie L. Ostrowski
Dartmouth College

Dartmouth Computer Science Technical Report TR2014-753

Chain Match: An Algorithm for Finding a Perfect Matching of a Regular Bipartite Multigraph

Stefanie Ostrowski

May 28, 2014

Abstract

We consider the problem of performing an edge coloring of a d-regular bipartite multigraph $G = (V, E)$. While an edge coloring can be found by repeatedly performing Euler partitions on G, doing so requires that the degree of G be a power of 2. One way to allow the Euler partitioning method to continue in cases where d is not a power of 2 is to remove a perfect matching from the graph after any partition that results in a graph with an odd degree. If this perfect matching can be identified in $O(E)$ time, we can maintain the best case runtime for this coloring of $O(E \lg d)$. This paper presents Chain Match, an algorithm that finds a perfect matching in a d-regular bipartite multigraph. While we have proven that Chain Match will always terminate with a perfect matching, we have not been able to implement it within our goal runtime of $O(E)$.

1 Introduction

A graph $G = (V, E)$ is bipartite if V can be divided into two disjoint subsets V_L and V_R such that vertices in V_L have edges to only vertices in V_R

and vice versa. A graph is *d-regular* if each vertex has exactly d incident edges. This graph is a *multigraph* if we allow multiple occurences of an edge (u, v) in E. A *perfect matching* of a bipartite graph $G = (V, E)$ is a set of edges $E' \subseteq E$ such that each vertex in V is an endpoint of exactly one edge in E'.

An *edge coloring* of a graph $G = (V, E)$ assigns a color to each edge $(u, v) \in E$ such that no vertex has more than one incident edge of a given color. Hall's Theorem implies that you can find d disjoint perfect matchings in any d-regular bipartite graph [Hal35]. Therefore, an edge coloring of a d-regular bipartite multigraph G can be found by finding these d disjoint perfect matchings and coloring the edges in each matching the same color. This solution is the optimum solution for the edge-coloring problem of G; that is, it finds the edge coloring of G that uses the minimum number of colors. Therefore, we can reduce the problem of edge coloring a d-regular bipartite multigraph to the problem of finding d disjoint perfect matchings.

If the degree of G is a power of 2, we can find these disjoint perfect matchings by performing $\lg d$ Euler partitions of G. An *Euler partition*, which can be performed on a d-regular graph in which all vertices have even degree, traces out disjoint cycles in G such that every edge is contained in a exactly one cycle. It then partitions these edges into two subgraphs by placing all the edges traversed from a vertex in V_L to a vertex in V_R in one subgraph and all the remaining edges (those traversed from a vertex in V_R to a vertex in V_L) in the other subgraph. Since a cycle, by definition, starts and ends on the same vertex, it contains the same number of left-to-right edges as right-to-left edges, and thus the subgraphs will contain an equal number of edges. Moreover, each of the subgraphs will have a new degree equal to $d/2$. Since the degree of G is a power of 2, we can continue performing Euler partitions on these subgraphs until we arrive at a set of subgraphs that each has $d = 1$, that is, a set of d disjoint perfect matchings.

When the degree is not a power of 2, however, we will necessarily arrive at a situation in which the degree of a subgraph is odd and an Euler partition cannot be performed. In such a case, removing a perfect matching from the subgraph would reduce the degree of each vertex by one, creating a subgraph with an even degree and allowing the Euler partioning method to continue. This paper presents Chain Match, an algorithm that finds a single perfect matching in a bipartite graph.

Chain Match begins by forming a set of chains C such that each chain $c \in C$ is a sequence of adjacent vertices in V. We build these chains by

performing a depth-first search (DFS) through G, adding each visited vertex to the current chain. Whenever the DFS would have created a branch in the DFS tree (upon reaching a vertex all of whose adjacent vertices have already been visited), we mark the current chain as complete and create a new chain, starting at the next vertex to be visited by DFS. When all the vertices have been visited we terminate our DFS. At this point, all vertices will be contained in exactly one chain. Each chain will either have an even or odd number of vertices. If all the chains have even length, we can find a perfect matching by pairing each vertex with its neighbor, i.e., pairing the first vertex with the second, the third with the fourth, etc., and putting the edges between these pairs into the matching. If the chains are not all even, we can recombine vertices by using other edges contained in E (those not represented by the connections implicit in current chains) until we have a set of only even chains, at which point we can form a perfect matching using the pairing method. In the rest of this paper, we will discuss how to rearrange vertices to form these even chains.

Before moving on to a more in-depth discussion of this algorithm, let us walk through an example. Consider the bipartite multigraph presented in Figure 1. After performing a DFS of the form described previously, we have four chains of odd length (7, 9, 17, and 18) and one chain of even length ($0-13-1-10-5-11-6-12-3-19-2-16-4-14-8-15$). We can recombine these chains to form all even-length chains in two steps. First, as Figure 2 shows, we use the edges $(7, 19)$ and $(17, 0)$ to eliminate two odd-length chains by breaking our even chain after vertex 19, connecting 19 to 7, and connecting 17 to 0. As Figure 3 shows, this recombination leaves us with the even-length chains $7-19-3-12-6-11-5-10-1-13-0-17$ and $2-16-4-14-8-15$ and the odd-length chains 9 and 18. Next, we can take advantage of the edges $(9, 14)$, $(18, 8)$, and $(4, 15)$ to eliminate our final two odd-length chains; we break the chain $2-16-4-14-8-15$ after vertex 16 and again after 14. We then connect 9 to 14, 4 to 15, and 8 to 18, forming the even-length chain $9-14-4-15-8-18$. At this point, we have three even-length chains ($7-19-3-12-6-11-5-10-1-13-0-17$, $2-16$, and $9-14-4-15-8-18$) and no odd-length chains. To form a perfect matching, we pair every other vertex with the vertex after it, resulting in the perfect matching shown in Figure 4.

0 : 13, 17, 18
1 : 10, 13, 15
2 : 16, 17, 19
3 : 12, 13, 19
4 : 14, 15, 16
5 : 10, 11, 18
6 : 11, 12, 17
7 : 11, 12, 19
8 : 14, 15, 18
9 : 10, 14, 16

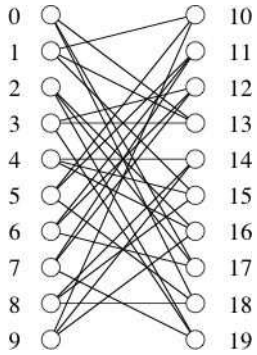

Figure 1: A bipartite multigraph with $N = 10$ and $d = 3$.

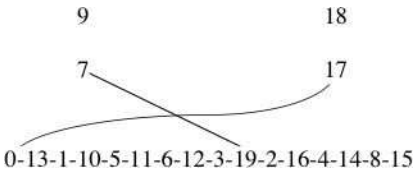

Figure 2: Step 1 of the algorithm. We use the edges $(7, 19)$ and $(17, 0)$ to eliminate odd chains 7 and 17.

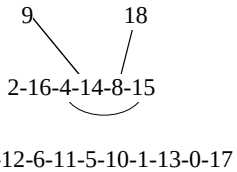

Figure 3: Step 2 of the algorithm. We use the edges $(9, 14)$, $(18, 8)$, and $(4, 15)$ to eliminate the remaining odd chains, 9 and 18.

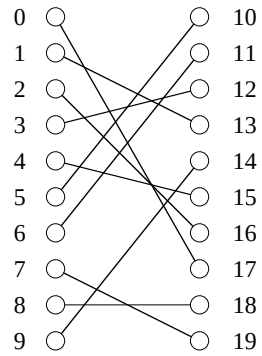

Figure 4: The perfect matching found using Chain Match.

2 Notation and Definitions

In order to analyze Chain Match, let us lay out the notation and terms we will use throughout our discussion.

A *left vertex* is a vertex from the left side V_L of a bipartite graph and a *right vertex* is a vertex from the right side V_R. Recall that a *chain* is a sequence of adjacent vertices of a bipartite graph. A chain containing an odd number of vertices is an *odd chain*, and a chain containing an even number of vertices is an *even chain*. An odd chain both begins and ends with either a left vertex or a right vertex; we will call the former an *l-chain* and the latter an *r-chain*. The *connections* between an l-chain L and some other chain B refer only to the set of edges between left vertices in L and right vertices in B. Similarly, the *connections* between an r-chain R and some other chain B refer only to the set of edges between right vertices in R and left vertices in B.

We use l and r to denote generic left and right vertices, respectively. We use L to denote an even chain of length 0 or greater beginning on an l and ending on an r. Similarly, we use R to denote an even chain of length 0 or greater beginning on an r and ending on an l. Subscripts denote specific instances of vertices and chains. The same subscript on an L and R denotes that they are reversals of each other; for example, L_1 and R_1 refer to the same chain but with the vertices in reversed order.

Figure 5: An odd connection. An edge connects left vertex l_2 in the l-chain to right vertex r_5 in the r-chain.

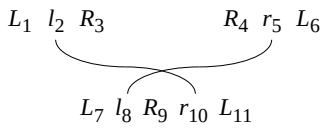

Figure 6: A crossed connection. An edge connects left vertex l_2 in the l-chain to right vertex r_{10} in the even chain. Another edge connects right vertex r_5 in the r-chain to left vertex l_8 in the even chain. This connection is *crossed* because, as this figure makes clear, the edge between the l-chain and the even chain crosses the edge between the r-chain and the even chain (when the even chain is oriented such that it begins on an l).

3 Proof of Possibility of Progress

As discussed previously, the success of Chain Match hinges on the ability to always arrive at a set of only even chains. In what follows, we will prove that, if there are any odd chains, then regardless of how many there are, it is always possible to perform a sequence of operations that will eliminate two odd chains (an l-chain and an r-chain) by taking advantage of the additional edges in G not contained in the chains at that moment. Thus, regardless of the initial arrangement of chains, it is always possible to arrive at a set of only even chains.

There are six ways in which an l-chain can stand in relation to an r-chain. We will refer to these relations as *connections*.

Case 1 As Figure 5 shows, an *odd connection* refers to a direct connection between an l-chain and an r-chain.

Case 2 A *shared even* refers to an even chain that is connected to both an l-chain and an r-chain. If an l-chain does not have an odd connection, it is either connected to a shared-even or it is not. As Figure 6 shows, a *crossed connection* refers to a situation in which an l-chain is connected to a shared even, such that the r in the even chain connected to the

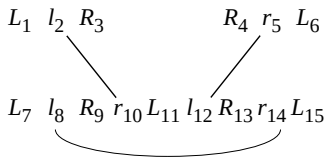

Figure 7: A shared bridge connection. An edge connects left vertex l_2 in the l-chain to right vertex r_{10} in the even chain. Another edge connects right vertex r_5 in the r-chain to left vertex l_{12} in the same even chain. There is a bridge that connects left vertex l_8 in the even chain to right vertex r_{14}, also in the even chain. Since the two spanned connections connect to the same even chain, this is a *shared* bridge connection.

l-chain occurs after the l in the even chain connected to the r-chain when the even chain is oriented such that it starts on an l.

Case 3 A *bridge* is an edge that connects a vertex to the left of a connection between an even chain and an l-chain to a vertex to the right of a connection between an even chain and r-chain when the even chain(s) is oriented such that it begins on an l. This bridge can be internal to a single even chain (i.e., the chain is connected to both an l-chain and r-chain) or it can connect two even chains (i.e., one even chain is connected to an l-chain and the other is connected to an r-chain). Figure 7 demonstrates a *shared bridge connection*, which is when the two connections being spanned connect to the same even chain.

Case 4 If an l-chain is not connected to an r-chain or a shared even it must be connected to an *unshared even*: an even-chain whose connections to odd-chains are either all l-chains or r-chains. Figure 8 shows an *unshared bridge connection*, which is when an l-chain is connected to an unshared even that has a bridge to another even chain that is connected to an r-chain.

Case 5 Figure 9 illustrates a *shared no-bridge connection* which occurs when none of the cases 1–4 apply and an l-chain is connected to a shared-even. This connection is like a shared bridge connection, but without a bridge.

Case 6 Figure 10 illustrates an *unshared no-bridge connection*, which occurs

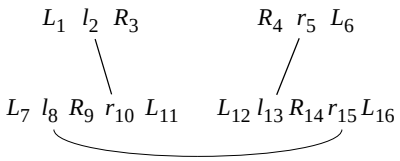

Figure 8: An unshared bridge connection. An edge connects left vertex l_2 in the l-chain to right vertex r_{10} in one of the even chains. Another edge connects right vertex r_5 in the r-chain to left vertex l_{13} in the other even chain. There is an edge from left vertex l_8 in one of the even chains to right vertex r_{15} in the other. This edge is considered a bridge because, when both even chains are oriented such that they begin with an l and the even chain connected to the l-chain is drawn to the left of the even chain connected to the r-chain, the bridge spans the other two connections.

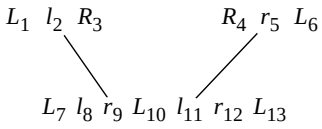

Figure 9: A shared no-bridge connection. An edge connects left vertex l_2 in the l-chain to right vertex r_9 in the even chain. Another edge connects right vertex r_5 in the r-chain to left vertex l_{11} in the even chain. There is no edge that spans these two connections, i.e., there is no edge from a left vertex in $L_7 l_8$ to a right vertex in $r_{12} L_{13}$.

> if none of cases 1–5 apply. This connection is like an unshared bridge connection, but without a bridge.

Because this list is exhaustive, we know that an l-chain always stands in one of the above six relations to an r-chain. Since we know that one of these cases will always apply, if we can prove that it is possible to reduce the number of odd chains in each situation, we will have shown that we can always reduce the number of odd chains and thus can ultimately arrive at a situation in which we have only even chains.

Lemma 3.1. If an odd connection exists, the number of odd chains can be reduced by two.

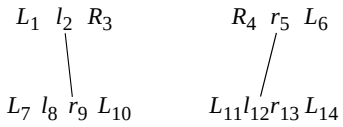

Figure 10: An unshared no-bridge connection. An edge connects left vertex l_2 in the l-chain to right vertex r_9 in one of the even chains. Another edge connects right vertex r_5 in the r-chain to left vertex l_{12} in the other even chain. There is no edge that spans these two connections, i.e., there is no edge from a left vertex in $L_7 l_8$ to a right vertex in $r_{13} L_{14}$.

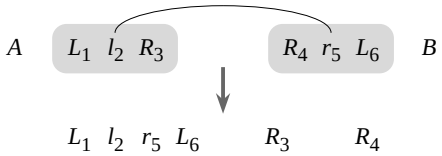

Figure 11: Processing an odd connection. The edge connecting the left vertex l_2 in A to the right vertex r_5 in B is an odd connection between chains A and B. By breaking A after l_2, breaking B before r_5, and connecting $L_1 l_2$ to $r_5 L_6$ via this edge, we have successfully taken two odd chains and produced three even chains, reducing the number of odd chains by two.

Proof. Suppose that, as in Figure 11, we have the l-chain $A = L_1 l_2 R_3$ and the r-chain $B = R_4 r_5 L_6$. Assume that there is an edge (l_2, r_5). By breaking chain A after l_2 and chain B before r_5, we can form the chains $L_1 l_2 r_5 L_6$, R_3, and R_4, all of which are even. ∎

Lemma 3.2. If a crossed connection exists, the number of odd chains can be reduced by two.

Proof. Suppose that, as in Figure 12, we have the l-chain $A = L_1 l_2 R_3$, the r-chain $B = R_4 r_5 L_6$, and the shared-even chain $C = L_7 l_8 R_9 r_{10} L_{11}$. Assume that the edges (l_2, r_{10}) and (r_5, l_8) exist. By breaking A after l_2, B before r_5, and C before l_8 and after r_{10}, and by reversing the subchain $l_8 R_9 r_{10}$ to form the chain $r_{10} L_8 l_8$, we can form the chains $L_1 l_2 r_{10} L_9 l_8 r_5 L_6$, R_3, R_4, L_7, and L_{11}, all of which are even. ∎

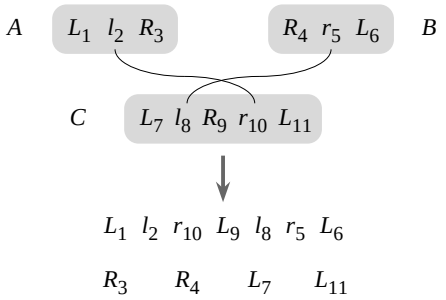

Figure 12: Processing a crossed connection. The edge connecting l_2 in A to r_{10} in C and the edge connecting r_5 in B to l_8 in C forms a crossed connection. We begin by breaking A after l_2, breaking B before r_5, and breaking C before l_8 and after r_{10}. We then reverse $l_8 R_9 r_{10}$ to form $r_{10} L_9 l_8$, which we connect to $L_1 l_2$ through the (l_2, r_{10}) edge, forming the chain $L_1 l_2 r_{10} L_9 l_8$. This chain is then connected to $r_5 L_6$ through the (r_5, l_8) edge, forming the even chain $L_1 l_2 r_{10} L_9 l_8 r_5 L_6$. We now have five even chains in the place of two odd chains, and have thus reduced the number of odd chains by two.

Lemma 3.3. If a shared bridge connection exists, the number of odd chains can be reduced by two.

Proof. Suppose that, as in Figure 13, we have the l-chain $A = L_1 l_2 R_3$, the r-chain $B = R_4 r_5 L_6$, and the shared-even chain $C = L_7 l_8 R_9 r_{10} L_{11} l_{12} R_{13} r_{14} L_{15}$. Assume that the edges (l_2, r_{10}) and (r_5, l_{12}) exist, as does the bridge (l_8, r_{14}). By breaking A after l_2, B before r_5, and C before l_8, after r_{10}, before l_{12}, and after r_{14}, and by reversing the subchains $l_8 L_9 r_{10}$ and $l_{12} R_{13} r_{14}$, we can form the chains $L_1 l_2 r_{10} L_9 l_8 r_{14} L_{13} l_{12} r_5 L_6$, R_3, R_4, L_7, L_{11}, and L_{15}, all of which are even. ∎

Lemma 3.4. If an unshared bridge connection exists, the number of odd chains can be reduced by two.

Proof. Suppose that, as in Figure 14, we have the l-chain $A = L_1 l_2 R_3$, the r-chain $B = R_4 r_5 L_6$, the unshared-even chain $C = L_7 l_8 R_9 r_{10} L_{11}$, and the unshared-even chain $D = L_{12} l_{13} R_{14} r_{15} L_{16}$. Assume that the edges (l_2, r_{10}) and (r_5, l_{13}) exist, as does the bridge (l_8, r_{15}). By breaking A after l_2, B before r_5, C before l_8 and after r_{10}, and D before l_{13} and after r_{15},

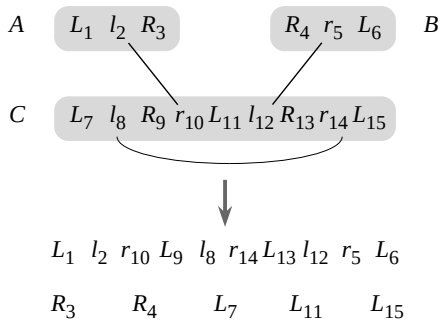

Figure 13: Processing a shared bridge connection. The edges connecting l_2 in A to r_{10} in C and connecting r_5 in B to l_{12} in C, together with the bridge from l_8 to r_{14} that spans these connections, form a shared bridge connection. To process this connection, we break A after l_2, B before r_5, and C before l_8, after r_{10}, before l_{12}, and after r_{14}. We then reverse $l_8R_9r_{10}$ to form $r_{10}L_9l_8$ and connect L_1l_2 to $r_{10}L_9l_8$ through the edge (l_2, r_{10}), forming $L_1l_2r_{10}L_9l_8$. Next, we reverse $l_{12}R_{13}r_{14}$ to form $r_{14}L_{13}l_{12}$ and connect $L_1l_2r_{10}L_9l_8$ to $l_{12}R_{13}r_{14}$ through the bridge (l_8, r_{14}), forming the chain $L_1l_2r_{10}L_9l_8r_{14}L_{13}l_{12}$. Finally, we connect this chain to r_5L_6 through the edge (r_5, l_{12}), forming $L_1l_2r_{10}L_9l_8r_{14}L_{13}l_{12}r_5L_6$. We are left with a set of only even chains instead of the original set which contained two odd chains, thereby reducing the number of odd chains by two.

and by reversing subchains $l_8R_9r_{10}$ and $l_{13}R_{14}r_{15}$, we can form the chains $L_1l_2r_{10}L_9l_8r_{15}L_{14}l_{13}r_5L_6$, R_3, L_7, L_{11}, L_{12}, L_{16}, and R_4, all of which are even. ∎

In order to prove that progress can be made when encountering a shared, no-bridge connection or an unshared, no-bridge connection, some additional lemmas will be useful.

Lemma 3.5. If we partition the chains formed from a connected component of a d-regular bipartite multigraph into two sets α and β, where α contains at least one l-chain, no r-chains, and zero or more even chains and β contains at least one r-chain, no l-chains, and zero or more even chains, then there must be an edge from an l in α to an r in β.

Proof. Suppose there are α_E even chains and α_L l-chains in α. Let us define the total number of l and r vertices in α as α_l and α_r, respectively. We know

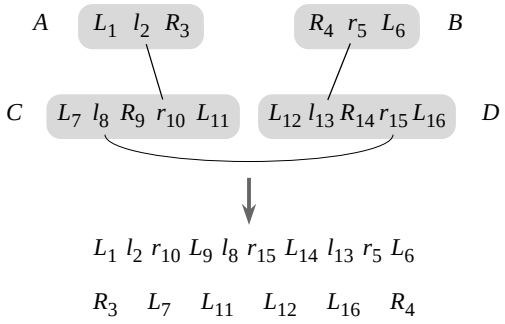

Figure 14: Processing an unshared bridge connection. The edges connecting l_2 in A to r_{10} in C and connecting r_5 in B to l_{13} in D, together with the bridge from l_8 in C to r_{15} in D, form an unshared bridge connection. To process this connection, we break A after l_2, B before r_5, C before l_8 and after r_{10}, and D before l_{13} and after r_{15}. We then reverse $l_8R_9r_{10}$ and $l_{13}R_{14}r_{15}$ and connect L_1l_2 to $r_{10}L_9l_8$ through the edge (l_2, r_{10}). Next, we connect the resulting chain, $L_1l_2r_{10}L_9l_8$, to $r_{15}L_{14}l_{13}$ through the bridge (l_8, r_{15}), forming the chain $L_1l_2r_{10}L_9l_8r_{15}L_{14}l_{13}$. Finally, we connect this chain to r_5L_6 through the edge (r_5, l_{13}), forming $L_1l_2r_{10}L_9l_8r_{15}L_{14}l_{13}r_5L_6$. We are left with a set of only even chains instead of the original set which contained two odd chains, thereby reducing the number of odd chains by two.

that an even chain has the same number of l and r vertices and an l-chain has one more l vertex than r vertex. From this, we know that

$$\alpha_l = \alpha_r + \alpha_L \ . \tag{1}$$

Since the graph is d-regular, there must be $\alpha_l d$ edges connected to ls in α or, by equation (1), there must be $(\alpha_r + \alpha_L)d$ edges connected to ls in α. Similarly, there must be $\alpha_r d$ edges connected to rs in α. Let us call the edges that do not leave α (i.e., edges for which both the l and r endpoints are contained in chains in α) *internal edges*, and the edges that do leave α *external edges*. Consider the case with the minimum number of external edges (the case where as few edges leave α as possible). This scenario would arise if the bipartite graph were such that every r in α was connected to an l also in α. This case is possible since there are fewer r vertices in α than l vertices. These edges account for $d\alpha_r$ of the $d(\alpha_r + \alpha_L)$ edges connected to ls in α. Thus, in the case with the minimum number of external edges, there must

be $d\alpha_L$ external edges connected to ls in α. Therefore, when we partition the chains of a connected component of a d-regular bipartite multigraph into two sets as described, there must be an edge from an l in one set to an r in the other. ∎

The concept of *elongating* a chain will also help with the proofs for the remaining two concepts. Consider the l-chain $A = L_1 l_2 R_3$ and the even chain $C = L_4 l_5 r_6 L_7$. If there are no connections between A and C, there is nothing to be done. If there are one or more edges between an l in A and an r in C, however, the edge (l_2, r_6) must exist such that there are no connections from ls in A to rs in L_7. To elongate A by C, we would break A after l_2, break C after r_6, reverse the subchain $L_4 l_5 r_6$ (resulting in $r_6 l_5 R_4$) and connect this chain to l_2. We are now left with the odd chain $A' = L_1 l_2 r_6 l_5 R_4$ and the even chains R_3 and $C' = L_7$. At this point, we have successfully elongated A by C. To *fully elongate* A by C, we would repeat this process, now with A' and C', until change no longer occurs. A full elongation of A by C results in the chains $\tilde{A}$ and $\tilde{C}$. For purposes of clarity in the upcoming proof, we will refer to the subchains of the initial odd chain that are broken off during the elongation process as the *tails* of A, such as R_3 in this case. Note that these tails will all be of even length and will not have any ls connected to rs in C. Elongating an r-chain by an even chain is an analogous process. In this case, we begin with the r-chain $B = R_1 r_2 L_3$ and the even chain $C = L_4 l_5 r_6 L_7$. Again, if there are no connections between B and C, we are done. If there are connections, we elongate using the edge (r_2, l_5), such that there are no connections between rs in B and ls in L_4, giving us the chains $B' = R_1 r_2 l_5 r_6 L_7$, $C' = L_4$, and the tail L_3. To fully elongate B by C, we repeat this process until changes no longer occur. Once again, the tails of B, the subchains broken off of B at each step of elongation, will all be of even length and there will not be any edges from rs in these tails to ls in C.

Lemma 3.6. After elongating A by B, there are no connections between $\tilde{A}$ and $\tilde{B}$.

Proof. Without loss of generality, let us assume that A is an l-chain. Then we have $A = L_1 l_2 R_3$. Let B be the even chain $L_4 l_5 r_6 L_7$. Assume that there exists the connection (l_2, r_6) such that r_6 is the rightmost connection within B to A, that is, there are no connections between A and L_7. The first step in the elongation of A by B results in the chains $A' = L_1 l_2 r_6 l_5 R_4$, $B' = L_7$, and R_3. Since there were no connections between an l in A and an r in L_7, the

only connections between ls in A' and rs in B' can be between l_5R_4 and B'. If there are no connections between these chains, elongation is done. Otherwise, repeat the process of elongation while changes occur. When changes no longer occur, there will be no connections between $\tilde{A}$ and $\tilde{B}$. Since elongation always decreases the length of B, this process necessarily terminates. Thus, after elongating A by B, there will be no connections between $\tilde{A}$ and $\tilde{B}$. ∎

Lemma 3.7. If a shared, no-bridge connection exists, the number of odd chains can be reduced by two.

Proof. Suppose that, as in Figure 15, we have the l-chain $A = L_1l_2R_3$, the r-chain $B = R_4r_5L_6$, and the shared even chain $C = L_7l_8r_9L_{10}l_{11}r_{12}L_{13}$. Assume that the edges (l_2, r_9) and (r_5, l_{11}) exist. Also assume that there is no bridge in C spanning these edges, that is, there is no edge from an l in L_7l_8 to an r in $r_{12}L_{13}$. Let D be the set of remaining even chains in this connected component of the graph. Assume that A and B are not connected by an odd connection, a shared even connection, a shared bridge connection, or an unshared bridge connection, that is, there is no chain in D that would create such a connection. Additionally, assume that there are no edges from ls in A to rs in L_{10} or from rs in B to ls in L_{10}. Let us perform the step shown in Figure 15, breaking A after l_2, B before r_5, and C before and after L_{10}. Using the edges (l_2, r_9) and (r_5, l_{11}), we are left with l-chain $A' = L_1l_2r_9l_8R_7$, r-chain $B' = R_6r_5l_{11}r_{12}L_{13}$, and even chains R_3, L_{10}, R_4, as well as the set of even chains D.

From our assumptions, we know that there are no edges from ls in the set $\{L_1, l_2, R_3, L_7, l_8\}$ to rs in the set $\{R_4, r_5, L_6, r_{12}, L_{13}\}$. Therefore, there are no connections between ls in R_3 and rs in B', between rs in R_4 and ls in A', or between ls in A' and rs in B'. There could, however, be edges between A' and L_{10} or between B' and L_{10}. To deal with this complication, let us fully elongate A' by L_{10} and place the tails of A' (the subchains broken off of A' during the elongation process) into a set F. As was discussed previously, after elongating A' by L_{10}, there will not be any connections between $\tilde{A'}$ and $\tilde{L_{10}}$. Similarly, to deal with potential connections between B' and $\tilde{L_{10}}$, let us elongate B' by the updated $\tilde{L_{10}}$ and place any tails of B' into a set H. Since A' and B' are modified through the elongation process, it is possible that after these elongations, there will be an odd connection between $\tilde{A'}$ and $\tilde{B'}$. If this occurs, we can process this connection as described in Lemma 3.1, and are thus finished.

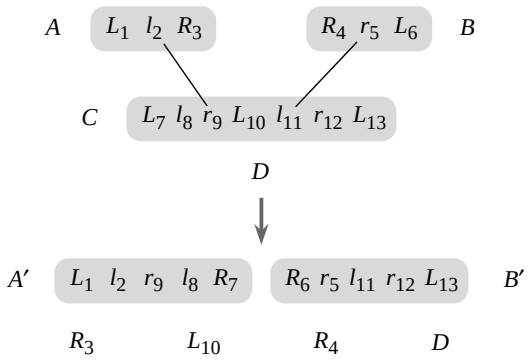

Figure 15: The first step in processing a shared, no-bridge connection. We break chain A after l_2, B before r_5, and C before and after L_{10}. We then reverse the subchain $L_7 l_8 r_9$ and connect it to $L_1 l_2$, forming A', $L_1 l_2 r_9 l_8 R_7$. We also connect $r_5 L_6$ to $l_{11} r_{12} L_{13}$ forming B', $R_6 r_5 l_{11} r_{12} L_{13}$.

If such a connection does not exist, we can divide our chains into two sets $\alpha = \{\tilde{A}', R_3, F\}$ and $\beta = \{\tilde{B}', \tilde{L_{10}}, R_4, D, H\}$, such that α contains all of our l-chains with some even chains and β contains all of the r-chains as well as the rest of the even chains.

By Lemma 3.5, we know that there must be an l in α connected to an r in β. As we have previously seen, there are no connections between ls in α and rs in $\tilde{B}'$, $\tilde{L_{10}}$, R_4, or H. Thus, there must be a connection between some l_α in α and some r_D in D (and hence, D cannot be the empty set). If l_α is a vertex in the chain $\tilde{A}'$, then $\tilde{A}'$ is directly connected to a chain in D. Alternatively, if l_α is in R_3 or a chain in F, we can elongate $\tilde{A}'$ by R_3 or the relevant chain in F (or both) until $\tilde{A}'$ contains l_α, directly connecting $\tilde{A}'$ to a chain in D and effectively undoing some of the elongation steps we performed earlier. Such a rearranging is possible because of the edge between l_2 and R_3 and the fact that F contains the tails of A'.

Let us now move the set of even chains D from β to α, so that we now have the partitions $\alpha' = \alpha \cup \{D\}$ and $\beta' = \beta - \{D\}$. Since $\tilde{B}'$, $\tilde{L_{10}}$, R_4, and H are not connected to $\tilde{A}'$, F, or R_3, using the same logic as the previous partition shows us that there must be an edge from B' to some chain in D (perhaps after elongating $\tilde{B}'$ by R_4 or H).

Since we showed that D is not the empty set, we know that D either contains one chain or more than one chain.

- If D contains only one chain, we know that this chain must be connected to both $\tilde{A}'$ and $\tilde{B}'$. Let us call this chain d_1. This connection is analogous to the case we had at the beginning of this proof; A has now been replaced by $\tilde{A}'$, B has been replaced by $\tilde{B}'$, C has been replaced by d_1, and the set of remaining even chains is now empty. However, we previously established that it is impossible to have a shared, no-bridge connection where the set of remaining even chains is empty. Thus, this situation is impossible, and the connection between A', B', and d_1 must either be a crossed connection or a shared, bridge connection, both of which can be processed.

- If D contains more than one chain, we know that $\tilde{A}'$ and $\tilde{B}'$ must either connect to the same chain in D or different chains in D. If this connection is a crossed connection, shared bridge connection, unshared bridge connection, or unshared, no-bridge connection we can proceed by processing the connection as outlined in previous or upcoming lemmas to reduce the number of odd chains by two, and are thus done. If, however, this connection is a shared, no-bridge connection, we once again have a situation analogous to how we started this proof: A has now been replaced by $\tilde{A}'$, B has been replaced by $\tilde{B}'$, C has been replaced by the chain in D that $\tilde{A}'$ and $\tilde{B}'$ are connected to (let us call it d_1), and the size of the set containing any remaining chains has been reduced by one (since we have removed d_1). Since we know it is impossible to have an unshared, no-bridge connection or shared, no-bridge connection where $D = \emptyset$, repeating this process will eventually lead to an odd connection, crossed connection, shared bridge connection, or unshared bridge connection, allowing us to decrease the number of odd chains by two by following the relevent steps outlined in previous lemmas.

Therefore, if a shared, no-bridge connection exists, we can decrease the number of odd chains by two. ∎

Lemma 3.8. If an unshared, no-bridge connection exists, the number of odd chains can be reduced by two.

Proof. Suppose that, as in Figure 16, we have the l-chain $A = L_1 l_2 R_3$, the r-chain $B = R_4 r_5 L_6$, and the even chains $C = L_7 l_8 r_9 L_{10}$ and $D = L_{11} l_{12} r_{13} L_{14}$. Assume that the edges (l_2, r_9) and (r_5, l_{12}) exist. Also assume

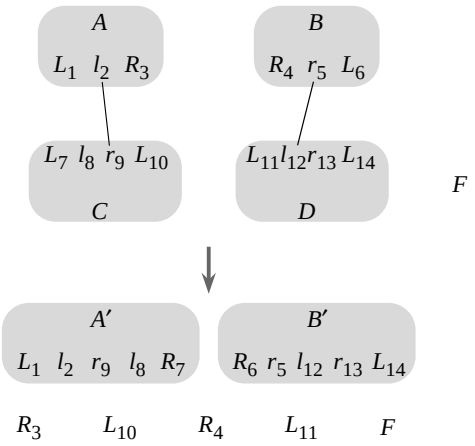

Figure 16: The first step in processing an unshared, no-bridge connection. We break chain A after l_2, B before r_5, C after r_9, and D before l_{12}. We then reverse subchain $L_7 l_8 r_9$ and connect it to $L_1 l_2$, forming $A' = L_1 l_2 r_9 l_8 R_7$. Additionally, we reverse subchain $r_5 L_6$ and connect it to $l_{12} r_{13} L_{14}$, forming $B' = R_6 r_5 l_{12} r_{13} L_{14}$.

that there is no bridge spanning these edges, that is, there is no edge from an l in $L_7 l_8$ to an r in $r_{13} L_{14}$. Let F be the set of remaining even chains. Assume that A and B are not connected by an odd connection, a shared even connection, a shared bridge connection, or an unshared bridge connection, that is, there is no chain in F that would create such a connection. Additionally, assume that there are no edges from ls in A to rs in L_{10} or from rs in B to ls in L_{11}. Let us perform the step shown in Figure 16, breaking A after l_2, B before r_5, C after r_9, and D before l_{12}. Using the edges (l_2, r_9) and (r_5, l_{12}), we are left with l-chain $A' = L_1 l_2 r_9 l_8 R_7$, r-chain $B' = R_6 r_5 l_{12} r_{13} L_{14}$, and even chains R_3, L_{10}, R_4, L_{11} as well as the set of even chains F.

If there exists an l in L_{10} that is connected to an r in L_{11}, we can use this edge to connect C and D, thus creating a shared, no-bridge connection which we can process, reducing the number of odd chains by two.

Otherwise, we have an almost identical situation to that of Lemma 3.7, but with a few slight differences. Once again, we know from our assumptions that there are no connections between ls in R_3 and rs in B', between rs in R_4 and ls in A', or between ls in A' and rs in B'. There could, however, be edges between ls in A' and rs in L_{11} or between rs in B' and ls in L_{10}. To handle this complication, let us fully elongate A' by L_{11} and B' by L_{10}, putting the

tails in sets H and I, respectively. Once again, we can partition these chains into two sets, where $\alpha = \{\tilde{A}', R_3, L_{10}, H\}$ and $\beta = \{\tilde{B}', R_4, L_{11}, I\}$. Using identical logic to the proof of Lemma 3.7, we can show that, after potentially performing a few elongations of $\tilde{A}'$ or $\tilde{B}'$, the chains $\tilde{\tilde{A}}'$ and $\tilde{\tilde{B}}'$ are either connected to the same chain in F or different chains in F. Let us call these chains f_1 and f_2, noting that f_1 and f_2 may refer to the same chain. If the connection between $\tilde{A}'$, $\tilde{B}'$, f_1, and f_2 is a crossed connection, shared bridge connection, unshared bridge connection, or shared, no-bridge connection, we can process it by the procedures described in previous lemmas, and are thus done. If it is an unshared, no-bridge connection, we can process it using the method described in this lemma. This process will necessarily terminate because, as was demonstrated in Lemma 3.7, the set containing the remaining even chains from a given shared, no-bridge or unshared, no-bridge connection cannot be empty, and decreases in size with each iteration of this processing. Therefore, if an unshared, no-bridge connection exists, we can decrease the number of odd chains by two.

■

4 Discussion of Runtime

Chain Match can naturally be broken into two parts: (1) building the initial chains with DFS and (2) rearranging the chains to arrive at a set of only even chains. For some regular, bipartite multigraph $G = (V, E)$ with N vertices on each side and a degree of d, we can build the initial chains in our goal runtime of $O(E)$, because we never consider an edge more than once in our DFS. Processing these chains, however, is significantly more difficult. The number of odd chains that we could have after the initial chain formation is $O(N)$. Therefore, in order to get rid of all the odd chains within our goal runtime of $O(E)$, we must both detect and process each of these connections in either a direct or amortized runtime of $O(d)$.

For now, let us imagine that we can detect connections in constant time. Connections of cases 1–4 can each be processed in a constant number of steps as was shown in our proof of progress. Case 5 and case 6 connections, which we will refer to as no-bridge connections, pose more of a problem. As our proofs demonstrated, processing no-bridge connections involves repeatedly recombining chains until we arrive at an odd, crossed, shared bridge, or unshared bridge connection. In between our initial no-bridge connection and

our end goal of forming a case 1–4 connection, we may form different no-bridge connections. Figure 17 shows the average number of consecutive no-bridge connections found when processing no-bridge connections for graphs with $d = 3$. For example, if we encounter a case 5 connection and then, in its processing, recombine it to form a case 6 connection, then a case 5 connection, and finally a case 2 connection, this would contribute three to the consective no-bridge connection count. As is demonstrated in Figure 17, while the number of consecutive no-bridge connections is very low compared with N, it does not appear to be bound by a constant number. Moreover, we cannot think of any theoretical reason why such a constant bound would exist. This means that, as long as some of our odd chains are connected in either of these ways, we cannot assume that the number of connections we will need to process is $O(N)$. Hence, even if we have a way to process connection types 1–4 in constant time, it does not seem as though we can achieve an overall runtime of $O(E)$.

Furthermore, we currently do not have a constant-time implementation of the processing steps for odd, crossed, shared bridge, or unshared bridge connections. While the actual breaking and recombining of chains can be done in constant time using linked lists, we have not found a way to update the information that is necessary for detecting future connections, such as which chain contains a certain vertex and whether a given chain is an l-chain, r-chain, or even chain. For example, after finding and processing the initial connection, we have no way to efficiently update the information about affected vertices in order to allow for the detection of future connections. These updates would not be a problem if vertices switched chains a constant number of times; however, this does not seem to be the case. Figure 18 shows the maximum number of times, M_{EO}, a vertex goes from an even chain to an odd chain over 50 trials at each N for a graph of varying N with $d = 3$. Once again, while the number of times this happens is very low when compared with N, it does not appear to be constant, nor do we have a theoretical justification for why it would be so. Thus, we would have to update the information pertaining to whether a vertex is in an even chain or odd chain more than a constant number of times, which appears to make our goal runtime unlikely for this algorithm.

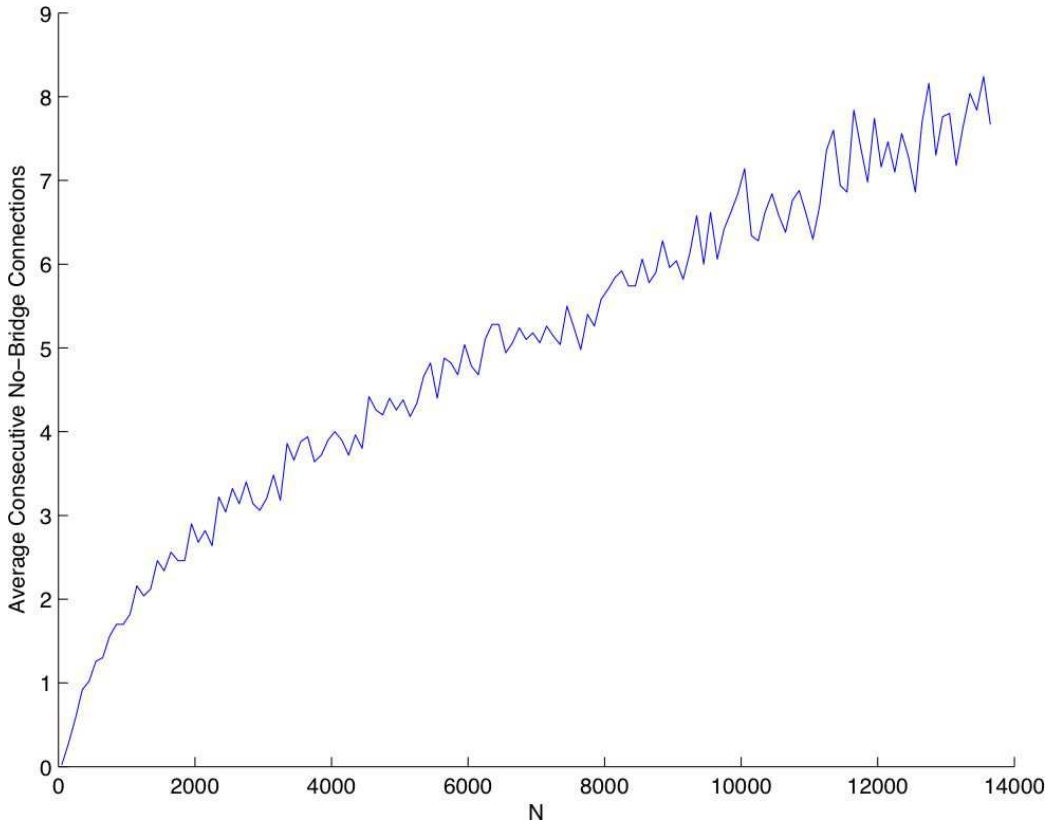

Figure 17: The average number of consecutive no-bridge connections for graphs of increasing N. While this count is small when compared with N, it does not appear to be bounded by a constant number, nor do we have a theoretical justification for why it would be.

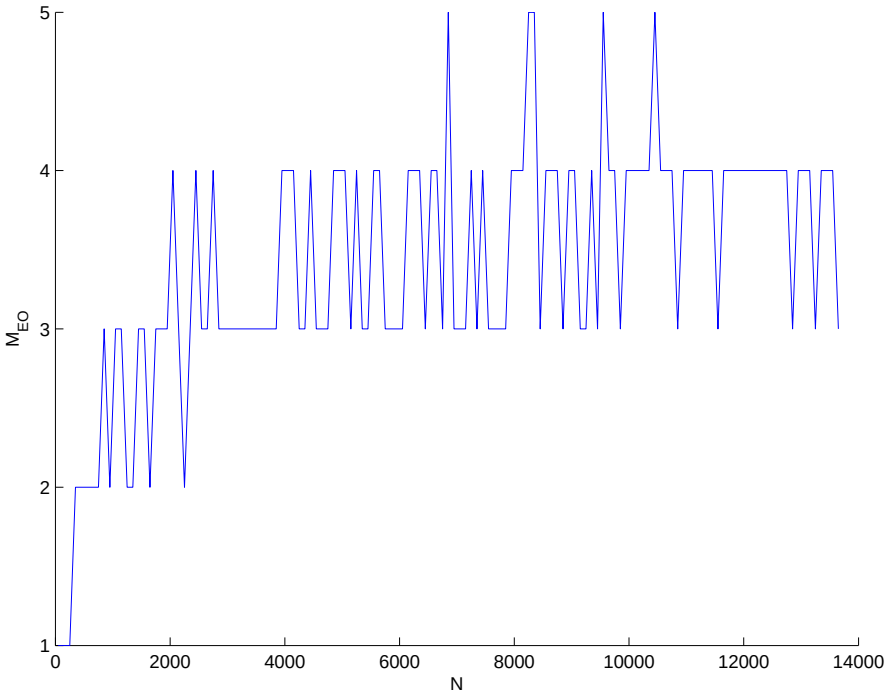

Figure 18: The maximum number of times a vertex goes from an even chain to an odd chain over 50 trials for a given N. While vertices move from even chains to odd chains a small number of times compared to N, this amount does not appear to be bounded by a constant number, nor do we have a theoretical justification for why it would be.

5 Conclusion

In this paper we presented Chain Match, an algorithm for finding a perfect matching of a d-regular bipartite multigraph, and proved that this algorithm always terminates with a perfect matching. However, we have not been able to implement this algorithm in such a way that would allow it to compete with other known perfect matching algorithms, nor does it seem as though such an implementation is possible.

6 Acknowledgments

Finally, I would like to thank Andrew Hannigan, Ellie Levey, Patricia Neckowicz, and Lauren Tran for their ideas and inspiration on this problem. Most importantly, I would like to thank Professor Tom Cormen, without whom this paper would not have been possible, for his unwavering support, guidance, and optimism throughout the past four years.

References

[Hal35] P. Hall. On representatives of subsets. *Journal of the London Mathematical Society*, 10(1):26–30, 1935.