Matroid Theory on Graphs

Ella Morgan
Supervised by Aras Erzurumluoglu

1 Introduction

Matroid theory was introduced in 1935 by Hassler Whitney in a paper titled "On the Abstract Properties of Linear Dependence" as an attempt to classify and capture the fundamental properties of linear dependence. Matroid theory applies the concept of linear dependence to areas outside of vector spaces. There are many different ways we can define the idea of dependence, resulting in many different types of matroids. For instance we can apply this concept to graphs through graphic matroids, where we consider a set of edges in a graph to be dependent if they form a cycle in the graph. We can then represent graph related problems such as finding minimum spanning trees, bipartite and non-bipartite matchings, or Hamilton cycles (allowing us to solve the travelling salesman problem) in graphs. This paper explores matroid theory with an emphasis on how its applicable to problems in graph theory.

Finding a minimum spanning tree in a graph is an example of a greedy algorithm, which attempts to find an optimal solution by greedily picking the next best option at each stage. If we can set up a problem we are trying to solve as a matroid, then the greedy algorithm will always provide an optimal solution. Bipartite matchings and finding Hamilton cycles in a graph are examples of problems which require the intersection of matroids, where we find all independent sets common between two or more matroids. Bipartite matching requires intersecting two matroids, and finding Hamilton cycles requires intersecting three matroids. A more complicated problem is non-bipartite matching, which requires the 'matroid parity problem'. In the matroid parity problem we pair off all our elements, and a set is only considered independent if for every element we include, we also include its paired element.

This paper is supplemented by examples coded in Python using SageMath [5] which are included in the appendix.

1.1 Definitions and Properties

A *matroid* is defined as an ordered pair $M = (E, I)$ where E is a finite set, referred to as the *ground set*, and I is a collection of *independent* subsets of E which satisfy the following properties [4]:

(I1) $\emptyset \in I$
(I2) If $I_1 \in I$ and $I_2 \subseteq I_1$, then $I_2 \in I$
(I3) If I_1 and I_2 are in I and $|I_1| < |I_2|$, then there is an element e of $I_2 \setminus I_1$ such that $I_1 \cup \{e\} \in I$.

M is a matroid if and only if its independent set I follows the properties listed above.

A *base* or *basis* of a matroid is an independent set of maximal size. We can uniquely define a matroid by its bases, and the other elements in I will be all subsets of the bases.

Theorem: All bases of a matroid have the same cardinality.

Proof: Assume there are two bases B_1 and B_2 of matroid M with different cardinalities. Without loss of generality assume $|B_1| < |B_2|$, then by (I3) there exists $e \in B_2 \setminus B_1$ such that $B_1 \cup e \in I$, contradicting B_1 having a maximal size, and thus B_1 cannot be a basis of M.

If B_1 is a basis of a matroid $M = (E, I)$, and $I_1 \in I$ has the same cardinality as B_1, then I_1 is clearly a maximal set of I and thus is also a basis of M.

M is a matroid if and only if its set of bases B satisfy the following properties [4]:

(B1) B is non-empty
(B2) If $B_1, B_2 \in B$ and $x \in B_1 \setminus B_2$, then $\exists y \in B_2 \setminus B_1$ such that $(B_1 \setminus x) \cup y \in B$

The *rank* of a matroid is the size of the largest linearly independent set defined on the matroid.

A *circuit* of a matroid M is a minimally dependent subset of M, where all proper subsets of a *circuit* are linearly independent. A matroid can also be uniquely defined by its circuits. Given a ground set E of a matroid M and a set of all circuits C, then the independent set I consists of all sets that are combinations of elements in the ground set which do not have any subsets that are circuits. M is a matroid if and only if its set of circuits C satisfy the following properties [4]:

(C1) $\emptyset \notin C$
(C2) If $C_1, C_2 \in C$ and $C_1 \subseteq C_2$, then $C_1 = C_2$
(C3) If C_1 and C_2 are distinct members of C and $e \in C_1 \cap C_2$, then there exists $C_3 \in C$ such that $C_3 \subseteq (C_1 \cup C_2) \setminus e$

Theorem: If I_1 is an independent set of a matroid $M = (E, I)$ and $e \in E$ is an element of M such that $I_1 \cup e$ is dependent, then $I_1 \cup e$ contains a unique circuit C_{e,I_1} which contains e.

Proof: There exists a circuit which must contain e, since I_1 is independent and e makes I_1 dependent. In order to prove that C_{e,I_1} is unique, let $C_1 = C_{e,I_1}$ and assume there is some other circuit $C_2 = C'_{e,I_1}$ distinct from C_1. By property (C3), since C_1 and C_2 are distinct members of C and $e \in C_1 \cap C_2$, then there must exist some $C_3 \in C$ such that $C_3 \subseteq (C_1 \cup C_2) \setminus e$. This causes a contradiction, as clearly $(C_1 \cup C_2) \setminus e$ is an independent set as $(C_1 \cup C_2) \setminus e \in I_1$.

1.2 Types of Matroids

Matric: A *matric matroid* $M = (E, I)$ is the matroid of a matrix A. The ground set E consists of the set of columns of A, and I is the set of all linearly independent subsets of E. We consider a set of vectors to be linearly dependent if one of the vectors can be expressed as a linear combination of the others.

Example: Below is a matrix that we will demonstrate a matric matroid on.

$$\begin{matrix} a & b & c & d & e \end{matrix}$$
$$\begin{bmatrix} 1 & 1 & 2 & 0 & 0 \\ 0 & 4 & 0 & 0 & 1 \end{bmatrix}$$

Let $M = (E, I)$ be the matric matroid corresponding to this matrix. The ground set is $E = \{a, b, c, d, e\}$, the set of independent sets is $I = \{\emptyset, \{a\}, \{b\}, \{c\}, \{e\}, \{a, b\}, \{a, c\}, \{a, e\}, \{b, c\}, \{b, e\}, \{c, e\}\}$. Since the size of the largest independent set is 2, this matroid has a rank of 2. Thus every independent set with cardinality 2 is a basis. There is a single circuit consisting of elements $C = \{a, b, e\}$.

Graphic: A graphic matroid $M = (E, I)$ has ground set E consisting of edges of a graph $G = (V, E)$ and independent sets consisting of all 'forests' of G, where a forest is a set of acyclic edges. M may also be referred to as the cycle matroid of G, and will frequently be denoted M_G.

Example: Let M_{G_1} be the graphic matroid of G_1, shown in Figure 1. Some independent sets of M_{G_1} include $\{f, b, d, g\}$, $\{c, f, g, e\}$, $\{a, b, f, g\}$ as these sets of edges are acyclic in G_1, and some dependent sets include $\{c, b, f\}$, $\{a, c, d, f\}$, $\{e, d, g, b\}$ as these sets contain cycles in G_1.

Some bases of M_{G_1} include $\{f, b, d, g\}$, $\{a, c, f, g\}$, $\{b, e, f, g\}$. Any other independent set with 4 elements is also a base, and since all bases have 4 elements the rank of M_{G_1} is 4.

All circuits of M_{G_1} can be easily listed, and they are:

$$C = \{\{a, c, f, g, e\}, \{a, c, f, d\}, \{a, b, g, e\}, \{c, b, f\}, \{a, b, d\}, \{e, d, g\}\}.$$

These sets all form cycles in G_1, and all proper subsets of them are independent.

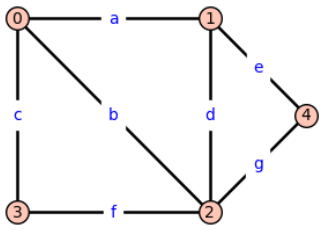


Figure 1: Graph G_1

Partition: Given a ground set E and a partition of E into disjoint blocks $P_1, P_2, ..., P_m$, we define a *partition matroid* to not have more than one element from each block P_i. Our independent set is the following:

$$I = \{I' \mid I' \subseteq E,\ |I' \cap P_i| \leq 1,\ i = 1, 2, ..., m\}$$

Example: Define a ground set $E = \{a, b, c, d, e, f, g\}$ with partition $P_1 = \{a, d, e\}$, $P_2 = \{c, g\}$, $P_3 = \{b, f\}$. The partition matroid on E then has the following bases:

$$B = \{\{a, c, b\}, \{a, c, f\}, \{a, g, b\}, \{a, g, f\}, \{d, c, b\}, \{d, c, f\}, \{d, g, b\},$$
$$\{d, g, f\}, \{e, c, b\}, \{e, c, f\}, \{e, g, b\}, \{e, g, f\}\}.$$

Where each basis has exactly one element from each block in the partition.

Transversal: Given a ground set $E = \{e_j;\ j = 1, 2, ..., n\}$ and a set of not necessarily disjoint subsets $Q = \{q_i;\ i = 1, 2, ..., m\}$ of E, a set of elements $T = \{e_{j(1)}, ..., e_{j(t)}\}$, $0 \leq t \leq n$ is a partial transveral of Q if the elements of T are distinct elements of E, and there are distinct integers $i(1), ..., i(t)$ such that $e_{j(k)} \in q_{i(k)}$ for $k = 1, 2, ..., t$. If $t = m$ then T is called a *transversal* of Q, and T is a system of distinct representatives of Q [1].

A transversal can be represented as a bipartite graph, where one group of vertices represent $Q = \{q_1, ..., q_m\}$ and the other group is the ground set $E = \{e_1, ..., e_n\}$. There is an edge between two vertices q_i and e_j if $e_j \in q_i$. Transversals are represented by maximum matchings in this graph.

A matroid $M = (E, I)$ is a *transversal matroid* if its set of independent sets I consists of all partial transversals and transversals of Q.

Example: Given a ground set $E = \{a, b, c, d, e, f, g, h\}$ and subsets $Q = \{q_1, q_2, q_3\}$ where $q_1 = \{a, b, d, e, h\}$, $q_2 = \{b, c, f\}$, $q_3 = \{d, f, g, h\}$. Figure 2 shows the bipartite graph G_2 which represents transversals of Q, where valid transversals are represented by maximum matchings in G_2.

The matching (q_1, a), (q_2, f) gives a partial transversal $T_1 = \{a, f\}$. Although f is also in q_3 this is only a partial transversal as $|T_1| < |Q|$.

Some transversals include $\{a, c, f\}$ which corresponds to the matching (q_1, a), (q_2, c), (q_3, f) and $\{b, d, h\}$ which corresponds to the matching (q_1, b), (q_2, d), (q_3, h).

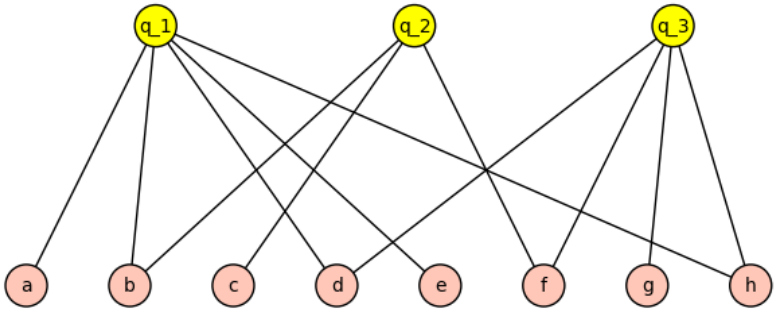


Figure 2: Transversals represented by a bipartite graph G_2

2 Greedy Algorithms

A greedy algorithm attempts to find an optimal solution by making the next best decision at each step without considering any future consequences. Matroid theory has a strong connection to greedy algorithms, as greedy algorithms provide an optimal solution for a problem if we can set up that problem as a matroid [3]. The greedy algorithm as shown in Algorithm 1 accepts a matroid $M = (E, I)$ and a weight function $w(\cdot)$ which assigns a non-negative weight to every element of E, and returns a maximum weighted set X_G of maximal size.

Algorithm 1 Greedy Algorithm

Input: A matroid $M = (E, I)$ with weight function $w(\cdot)$
Output: An independent set of maximum weight X_G

$X_0 = \emptyset$ and $j = 0$ // *initialization*

while $\exists e \in E \setminus X_j$ such that $X_j \cup e \in I$ **do**
  Choose an element e_{j+1} with maximum weight, such that $e_{j+1} \in E \setminus X_j$ and $X_j \cup e \in I$

  $X_{j+1} = X_j \cup e_{j+1}$
  $j = j + 1$
end while

$X_G = X_j$
return X_G

Theorem: Given a matroid $M = (E, I)$ with non-negative weight function $w(\cdot)$, the greedy algorithm (Algorithm 1) will always provide an optimal solution X_G.

Proof: Given any basis $B = \{f_1, f_2, ..., f_k\}$ indexed in a way such that $w(f_1) \geq w(f_2) \geq ... \geq w(f_k)$ we will show that the result of the greedy algorithm $X_G = \{e_1, e_2, ..., e_k\}$ (indexed such that $w(e_1) \geq w(e_2) \geq ... \geq w(e_k)$) will always have a total weight greater than or equal to the weight of B. In order to prove this, we will show that for any $1 \leq i \leq k$, $w(e_i) \geq w(f_i)$. Assume this to not be true, and let j be the smallest index such that $w(e_j) < w(f_j)$. Then let $I_1 = \{e_1, e_2, ..., e_{j-1}\}$ and $I_2 = \{f_1, f_2, ..., f_j\}$, but by (I3) since $|I_1| < |I_2|$ there is some $f_t \in I_2$ such that $I_1 \cup f_t \in I$. Because $w(e_j) < w(f_j)$ and $w(f_t) \geq w(f_j)$, the greedy algorithm would have selected f_t at some stage before selecting e_j since $w(f_t) > w(e_t)$, causing a contradiction.

By the last theorem, we can see that all matroids have the following property [4]:

(G) For all non-negative weight functions w: $E \to \mathbb{R}$, the greedy algorithm produces a maximal member of I of maximum weight.

Theorem: $M = (E, I)$ is a matroid if and only if it satisfies the properties (I1) and (I2) from Section 1.1, and property (G).

Proof: If M is a matroid then we know it must satisfy (I1), (I2), and (G).

Assume that M satisfies (I1), (I2), and (G) but M is not a matroid, thus M does not satisfy (I3). That is, there exists some $I_1, I_2 \in I$ where $|I_1| < |I_2|$ and for all $e \in I_2 \setminus I_1$, $I_1 \cup e \notin I$.

$I_1 \setminus I_2$ is non-empty, since otherwise by (I2) there would be an element in $I_2 \setminus I_1$ which we could add to it that would make an independent set. Since $|I_1 \setminus I_2| < |I_2 \setminus I_1|$ we can find some positive $\epsilon \in \mathbb{R}$ such that $0 < (1+\epsilon)|I_1 \setminus I_2| < |I_2 \setminus I_1|$.

Define the weight function w: $E \to \mathbb{R}$ as the following:

$$w(e) = \begin{cases} 2 & \text{if } e \in I_1 \cap I_2, \\ 1 \ / \ |I_1 \setminus I_2| & \text{if } e \in I_1 \setminus I_2, \\ (1+\epsilon) \ / \ |I_2 \setminus I_1| & \text{if } e \in I_2 \setminus I_1, \\ 0 & \text{otherwise.} \end{cases}$$

The greedy algorithm will first select elements of $I_1 \cap I_2$, and then by the assumption that adding any elements of $I_2 \setminus I_1$ will make the set dependent, the algorithm will add all elements from $I_1 \setminus I_2$. Then, as many elements from $E \setminus (I_1 \cup I_2)$ will be added as necessary (which have a weight of 0) to provide a maximal independent set. Thus our maximum weight set X_G has weight

$$w(X_G) = 2|I_1 \cap I_2| + |I_1 \setminus I_2|(1/|I_1 \setminus I_2|) = 2|I_1 \cap I_2| + 1.$$

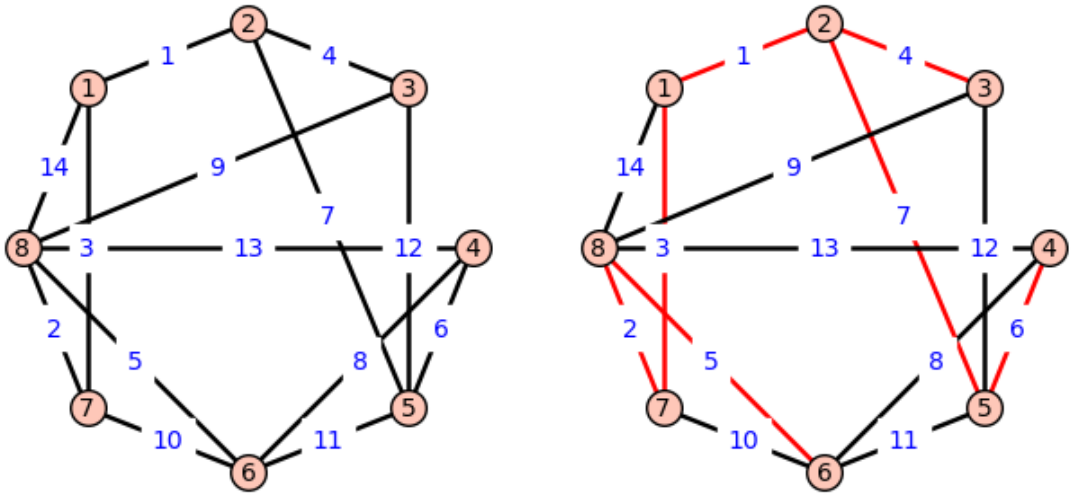
Consider I_2, by (I2) I_2 must be contained in a maximal set of I, thus there exists some basis I_2' containing I_2 with weight function

$$w(I_2') \geq w(I_2) = 2|I_1 \cap I_2| + |I_2 \setminus I_1|((1+\epsilon)/|I_2 \setminus I_1|) = 2|I_1 \cap I_2| + 1 + \epsilon.$$

But by (G) it cannot be that case that $w(I_2') > w(X_G)$, as the greedy algorithm will always produce a maximal member of I of maximum weight, resulting in a contradiction. Hence if M satisfies (I1), (I2), and (G) then M must satisfy (I3) as well, proving that M is a matroid.

Minimum Spanning Tree: Finding a minimum spanning tree is the most frequently used example of the greedy algorithm for matroids. A *spanning tree* of a graph G is a set of edges in G which connect all vertices in G and contain no cycles. When finding a minimum spanning tree in a weighted graph G we wish to find a set of edges which form a spanning tree of minimum total weight. In matroid theory, this corresponds to finding a minimum weight basis of the graphic matroid of G. This can be accomplished through the greedy algorithm with the modification that we seek a minimum weighted base instead of a maximum one.

Example: Figure 3(a) displays a weighted graph G_3 that we wish to find a minimum spanning tree in. Note that the weight corresponding to each edge is the number in the middle of that edge, in the cases where the number overlaps two edges. This is the input graph for appendix code A.2 along with a list of weights associated with each edge. First the function finds the graphic matroid M_{G_3}, then it orders the edges by smallest to largest weight. It begins by adding the edge with the smallest weight to a set X_{G_3}. It keeps adding the next smallest weight to X_{G_3} that can be added without causing the set to become dependent. The function returns the list of edges highlighted in red in Figure 3(b).



(a) Graph G_3 to find a minimum spanning tree in

(b) G_3 with minimum spanning tree highlighted in red

Figure 3: Finding a spanning tree in a graph using the greedy algorithm

Greedy Job Assignment: A job assignment problem where there is a pool of applicants each qualified for a set of jobs, and a pool of jobs ranked in the priority at which they need to be filled. This problem can be solved optimally by the greedy algorithm. At each step we take the job that needs to be filled with the highest priority, and assign that job to the applicant who is most qualified.

3 Operations on Matroids

In this section we go over different operations on matroids which form new matroids. These operations include finding duals, deletion and contraction to form matroid minors, and parallel-series expansion of matroids.

3.1 Duals of Matroids

The *dual* of a matroid $M = (E, I)$ is denoted $M^* = (E, I^*)$. All independent sets of M^* are disjoint from at least one basis of M. Let $B = \{B_1, B_2, ..., B_n\}$ be the set of bases of M. Then the set of bases B^* of M^* is $B^* = \{E \setminus B_1, E \setminus B_2, ..., E \setminus B_n\}$.

Theorem: The dual M^* of a matroid M is a matroid.

Proof: Let M^* be the dual of matroid M, where B, B^* are the set of bases of M, M^*, respectively. We will show that M^* satisfies conditions (B1) and (B2) from Section 1.1, from which it follows that M^* is a matroid.

(B1) Since B is non-empty, it follows that B^* is non-empty.

(B2) Suppose $B_1^*, B_2^* \in B^*$ and let $x \in B_1^* \setminus B_2^*$. We must show that $\exists y \in B_2^* \setminus B_1^*$ such that $(B_1^* \setminus x) \cup y \in B^*$.

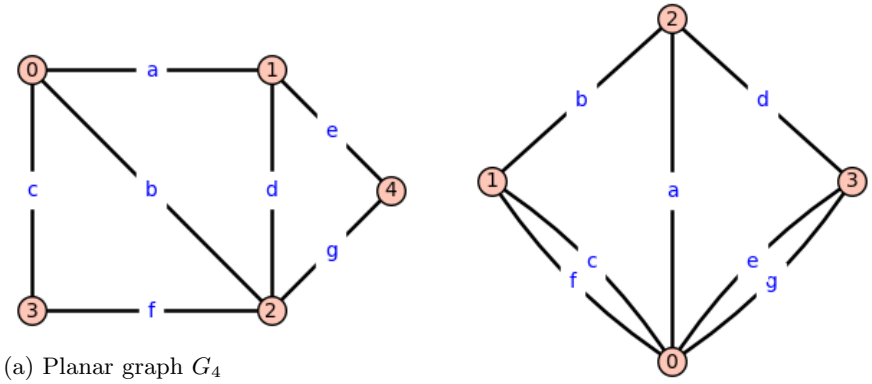
By definition $B_1^* = E \setminus B_1$ and $B_2^* = E \setminus B_2$, then $B_1^* \setminus B_2^* = B_2 \setminus B_1$ where $B_1, B_2 \in B$. It follows that $x \in B_2 \setminus B_1$ and from Section 1.1, there is a unique circuit C_{x,B_1} in $B_1 \cup x$ which will contain x, where C_{x,B_1} is minimally dependent so all of its subsets are independent sets.

Since C_{x,B_1} is dependent and B_2 is independent, $C_{x,B_1} \setminus B_2$ is non-empty. Let $y \in C_{x,B_1} \setminus B_2$. Then because $(B_1 \setminus y) \cup x$ will not contain C_{x,B_1}, and C_{x,B_1} was a unique circuit, it follows that $(B_1 \setminus y) \cup x$ is independent. Because $(B_1 \setminus y) \cup x$ is independent and $|(B_1 \setminus y) \cup x| = |B_1|$, it follows that $(B_1 \setminus y) \cup x \in B$.

Since $y \in B_1 \setminus B_2$, we have that $y \in B_2^* \setminus B_1^*$ and $(B_1 \setminus y) \cup x \in B$ implies that $E \setminus ((B_1 \setminus y) \cup x) \in B^*$. Since $E \setminus ((B_1 \setminus y) \cup x) = ((E \setminus B_1) \setminus x) \cup y = (B_1^* \setminus x) \cup y$, condition (B2) is satisfied.

Dual of a planar graph: A planar graph G is a graph that can be drawn on the plane in a way where none of its edges intersect. The *geometric dual* G^* of planar graph G has a vertex for every face of G, and there is an edge between two vertices in G^* if that edge touches both faces in G. Let M_G be the graphic matroid of a planar graph G, then the dual M_G^* is a graphic matroid. Furthermore, if G^* is the geometric dual of G then the graphic matroid M_{G^*} is isomorphic to M_G^*. Proofs for these will not be presented here, but can be found in [4].

Example: Figure 4(a) shows a planar graph G_4 with 4 faces, and Figure 4(b) shows its planar dual G_4^*, which has a vertex for each face of G_4. The face contained by edges $\{c, b, f\}$ in G_4 corresponds to vertex 1 in G_4^*, and similarly the face contained by edges $\{a, b, d\}$ corresponds to vertex 2, $\{e, d, g\}$ corresponds to vertex 3, and vertex 0 is the area outside of graph G, known as the 'unbounded face'. Edges between vertices in G_4^* represent the edges in G_4 touching both planes. For example edge a in G_4 touches face 2 and the outer face, thus vertices 2 and 0 in G_4^* are connected by a. Edge b in G_4 touches faces 1 and 2, thus b connects vertices 1 and 2 in G_4^*. The dual of the graphic matroid M_{G_4} is then the graphic matroid of G_4^*, $M_{G_4^*}$.



(a) Planar graph G_4

(b) Geometric dual G_4^*

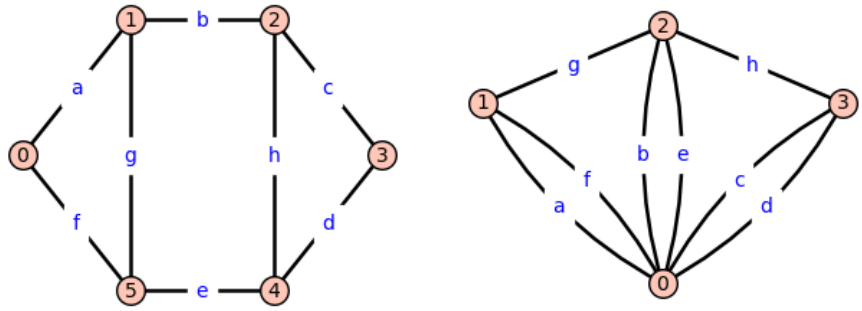
Figure 4: Geometric dual of a planar graph

3.2 Minors

Matroid *minors* are formed by two operations: *restriction* and *contraction*. Let $M = (E, I)$ and take some $Y \subseteq E$. Then $M' = (Y, I')$ is a matroid, where $I' = \{y \mid y \subseteq Y,\ y \in I\}$. We call this the restriction of M to Y, and denote it $M|_Y$. If $T = E \setminus Y$ then we denote M' as $M \setminus T$ and refer to it as the *deletion* of T. Clearly $M|_Y = M \setminus (E \setminus Y)$. Contraction is the dual of restriction, and it is denoted M/T which we refer to as the *contraction of T from M*. We define contraction in terms of the dual and restriction operations, where $M/T = (M^* \setminus T)^*$.

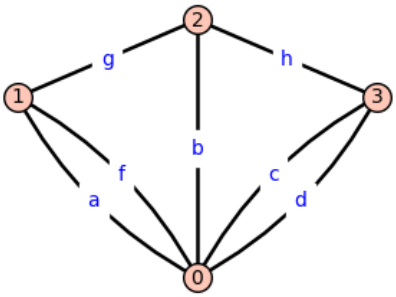
Example: This example shows how if M_{G_5} is the graphic matroid of a planar graph G_5 then we can represent the changes being made to M_{G_5} through how G_5 changes. Figure 5(a) shows a planar graph G_5 and Figure 5(b) shows G_5's geometric dual G_5^*, where $M_{G_5}^*$ is the graphic matroid of

G_5^*. Figure 5(c) shows $G_5^* \setminus e$ which is the deletion of edge e from G_5^* and is obtained from simply removing that edge from the graph without altering the other edges or vertices. The graphic matroid of $G_5^* \setminus e$ is equivalent to $M_{G_5}^* \setminus e$. Figure 5(d) shows $(G_5^* \setminus e)^*$, which is the geometric dual of the graph in Figure 5(c) which is equivalent to finding G_5/e, the contraction of e from the original graph G_5. The graphic matroid of G_5/e gives the contraction of e from M_{G_5}, or M_{G_5}/e.

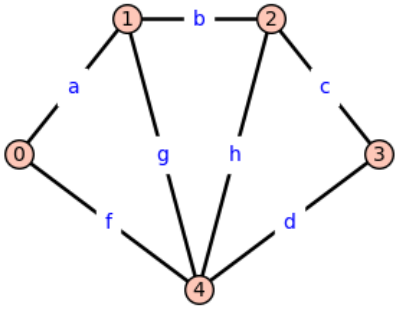


(a) Planar graph G_5

(b) Geometric dual G_5^*



(c) G_5^*/e



(d) $(G_5^*/e)^*$

Figure 5: Dual and minors of a planar graph

3.3 Series-Parallel Extension

Starting with a graph G we can add edges in *series* or *parallel* to other edges in G. If we can form a graph by starting from a single edge and building the graph through only adding edges in series or parallel, then we refer to the graph as a *series-parallel graph* [6].

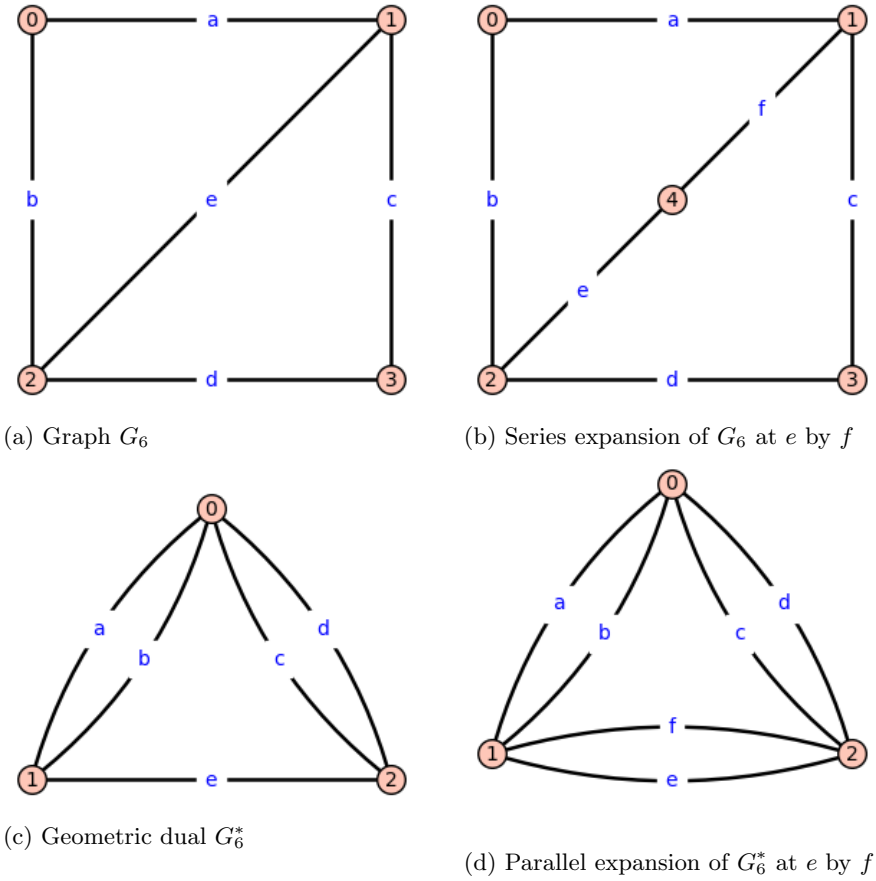
In *series expansion* an edge e of G is replaced by two edges e_1, e_2 in series. For a matroid M, expanding M at x by y in series results in the bases B of M becoming one of the following: $B \cup y$ for B a base of M, or $B \cup x$ for B a base of M and $x \notin B$. We denote the series expansion of M at x by y as $sM(x, y)$.

In *parallel expansion* an edge e of G is replaced by two edges e_1, e_2 in parallel. For a matroid M, expanding M at x by y in parallel results in the bases of M are all one of the following: B for B a base of M, or $(B \setminus x) \cup y$ for B a base of M and $x \in B$. We denote the parallel expansion of M at x by y as $pM(x, y)$.

Finding the dual of the series expansion of M at x by y is equivalent to finding the parallel expansion of M^* at x by y: $(sM(x, y))^* = pM^*(x, y)$.

Example: This example shows series and parallel expansion as an operation on the underlying graph of a graphic matroid. Figure 6(a) displays a graph G_6, and Figure 6(b) demonstrates the

series expansion of G_6 at e by f. The graphic matroid of this graph is $sM_{G_6}(x, y)$. The geometric dual of G_6 is shown in Figure 6(c), and the parallel expansion of G_6^* at e by f is shown in Figure 6(d). The graphic matroid of this graph is $pM^*_{G_6}(e, f)$. It is evident from these graphs that the geometric dual of Figure 6(b) is the graph of Figure 6(d), displaying that $(sM_{G_6}(e, f))^* = pM^*_{G_6}(e, f)$.



(a) Graph G_6

(b) Series expansion of G_6 at e by f

(c) Geometric dual G_6^*

(d) Parallel expansion of G_6^* at e by f

Figure 6: Series and parallel expansion of a graph

4 Operations Combining Matroids

4.1 Matroid Intersection

We define matroid *intersection* as the intersection of the independent sets of two matroids $M_1 = (E, I_1)$, $M_2 = (E, I_2)$ defined on the same ground set E. Their intersection $M_1 \cap M_2$ is the following:

$$M_1 \cap M_2 = (E, I_1 \cap I_2).$$

The result of $M_1 \cap M_2$ is not necessarily a matroid. With weighted matroid intersection we find the independent set with the largest weight common to both matroids.

4.2 Matroid Union

Given two matroids $M_1 = (E_1, I_1)$ and $M_2 = (E_2, I_2)$ we define their *union* $M_1 \cup M_2$ as the following:

$$M_1 \cup M_2 = (E_1 \cup E_2, I)$$

Where I is defined as follows:

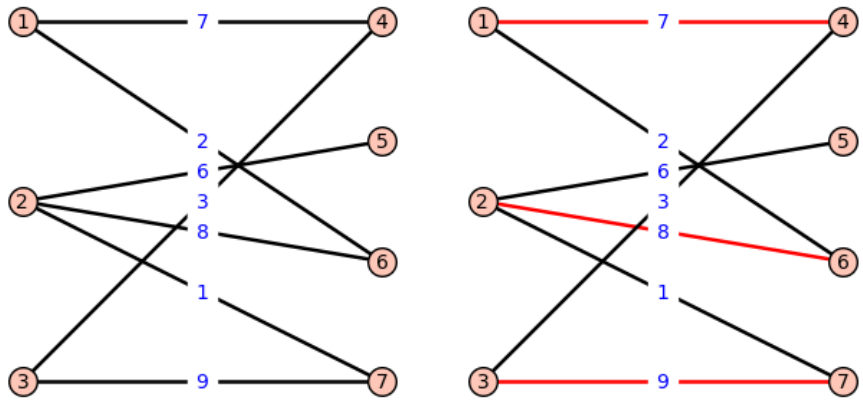
$$I = \{I_1' \cup I_2' \mid I_1' \in I_1, I_2' \in I_2\}$$

The union of matroids results in a matroid.

4.3 Applications

Bipartite Matching: A maximal matching of a bipartite graph G with parts X and Y can be found by intersecting two partition matroids of G. Take the first matroid M_1 to be the partition matroid where elements of the ground set are placed in the same block if they are adjacent to the same vertex in X, and likewise let M_2 be the partition matroid where elements are placed in the same block if they are adjacent to the same vertex in Y. The intersection of M_1 and M_2 then gives all feasible matchings, and if the edges of G are weighted then an optimal solution can be found by determining which feasible matching has maximum weight.

Example: Figure 7(a) shows a weighted bipartite graph which we seek a maximum weighted matching on. This graph is the input to appendix code A.3 along with a set of its weights. In the function the first matroid M_1 corresponds to the partition matroid induced by edges incident to vertices 1, 2, and 3. Thus in each independent set of M_1 there is only at most one edge incident to each of these vertices. Likewise, partition matroid M_2 is induced by edges incident to vertices 4, 5, 6, and 7, thus in its independent sets there is only at most one edge incident to each of those vertices. The intersection of these two matroids returns all feasible matchings. The edges highlighted in red in Figure 7(b) display the largest matching returned by the function.



(a) A weighted bipartite graph G_7 to find a maximum matching on

(b) The graph G_7 with its optimal matching highlighted in red

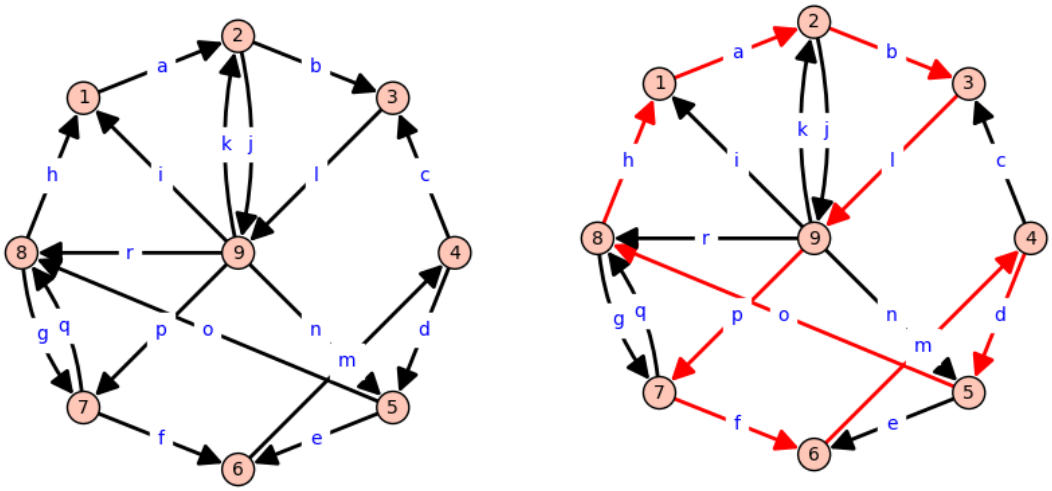
Figure 7: Finding a matching in a bipartite graph through matroid intersection

Hamilton Cycle: A *Hamilton cycle* is a path in a graph G on n vertices which starts and ends on the same vertex, and visits each vertex of G in between the start and end vertex exactly once. This problem can be formulated as the intersection of three matroids. Before creating our matroids we arbitrarily choose one vertex e to split into two vertices, one where we will begin at and one we will end at, creating a new graph G' on $n+1$ vertices. All outgoing edges of e shall leave from our 'starting' vertex e_1 and all incoming edges of e shall return to our 'ending' vertex e_2. Our first matroid M_1 will be a graphic matroid of the undirected graph of G'. The other two matroids M_2 and M_3 will be partition matroids. In M_2 we place to edges in the same block if they are directed into the same vertex, and likewise in M_3 we place edges in the same block if they direct out of the

same vertex. We then intersect M_1, M_2, M_3, and we only consider a solution feasible if it consists of n edges.

The *travelling salesman problem* is the problem of finding a minimum weighted Hamilton cycle in a graph G. In the case that we have more than one feasible solution, we take the optimal solution to be the solution with the smallest total weight.

Example: Figure 8(a) shows a directed graph that we seek to find a Hamilton cycle in. This graph is the input to appendix code A.4, which created three matroids based on this graph and finds their intersection. The first matroid is the graphic matroid of the undirected graph of G_7, the first partition matroid is based off the partition $P_1 = \{\{h,i\}, \{a,k\}, \{b,c\}, \{m\}, \{d,n\}, \{e,f\}, \{g,p\}, \{r,o,q\}, \{j,l\}\}$ and the second partition matroid is based off the partition $P_2 = \{\{a\}, \{b,j\}, \{l\}, \{c,d\}, \{e,o\}, \{m\}, \{f,q\}, \{g,h\}, \{i,k,n,p,r\}\}$. The resulting Hamilton cycle obtained by intersecting these three matroids is highlighted in red in Figure 8(b).



(a) Directed graph G_8 to find a Hamilton cycle in

(b) Result of finding Hamilton cycle in G_8 highlighted in red

Figure 8: Finding a Hamilton cycle in a graph

5 Matroid Parity Problem

The matroid parity problem generalizes matroid intersection and non-bipartite matching problems [2]. We pair off the elements in the ground set E of a matroid M, then for any $e \in E$ and its 'paired' element $\bar{e} \in E$, if we add e to a parity set S we are required to add $\bar{e}$ to the parity set as well.

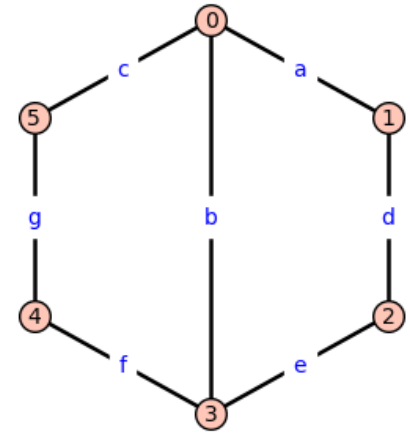
Matroid Intersection: The intersection of two matroids $M_1 = (E, I_1)$ and $M_2 = (E, I_2)$ can be generalized by the matroid parity problem. Create a new matroid $\bar{M}_2 = (\bar{E}, \bar{I}_2)$ isomorphic to M_2 where E and $\bar{E}$ are disjoint. Then the intersection of M_1 and M_2 becomes the problem of solving the parity problem on $M_1 \cup \bar{M}_2$ where if $e \in E$ is included in an independent set, then $\bar{e} \in \bar{E}$ must also be in the independent set.

Non-bipartite Matching: To find a matching in a graph $G = (V, E)$ which may or may not be non-bipartite, start by subdividing all edges of G so that each edge in G is replaced by two edges e and $\bar{e}$ (adding a new vertex connecting those two edges), and call this new subdivided graph

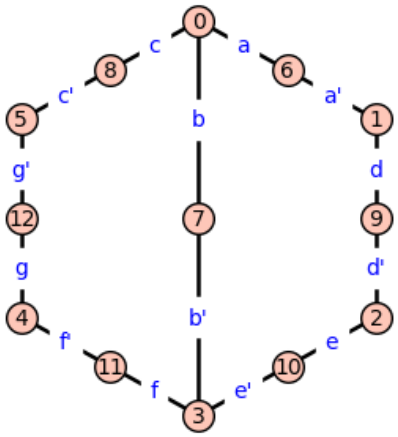
$G' = (V', E')$. Then we define a partition matroid M where edges of E' are placed in the same block if they are incident to the same vertex V from the original graph G. We then find feasible matchings by solving the matroid parity problem, where each edge e and its pair $\bar{e}$ must be added to an independent set together.

Example: Figure 9(a) shows the unweighted graph we wish to find a matching on. The first step is subdividing all the edges and renaming them, as shown in Figure 9(b). For example in G_9 edge a is adjacent to vertices 0 and 1, and after subdividing a into a and a' a new vertex is added (vertex 6) so that a is now incident to vertices 0 and 6, and a' is incident to vertices 1 and 6.

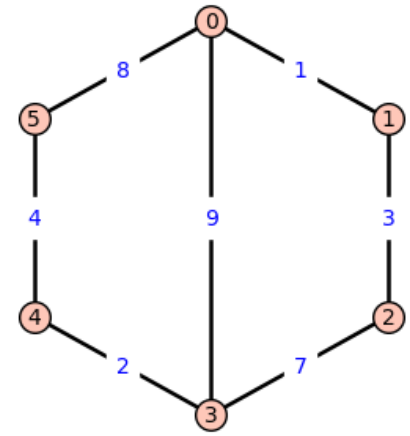
Next the partition matroid is formed with blocks $P = \{\{a, b, c\}, \{a', d\}, \{d', e\}, \{b', e', f\}, \{f', g\}, \{c', g'\}\}$, then we solve the matroid parity problem on this partition matroid, where we only select independent sets where each edge is included with its 'pair', e.g. if a is in an independent set then a' must be as well. The result of this is a set of all feasible solutions, then we choose the optimal solution to be the feasible solution with largest weight. The code for this is given in appendix code A.6. The weights of the graph are displayed in Figure 9(c), and the largest weight matching is highlighted in red in Figure 9(d).



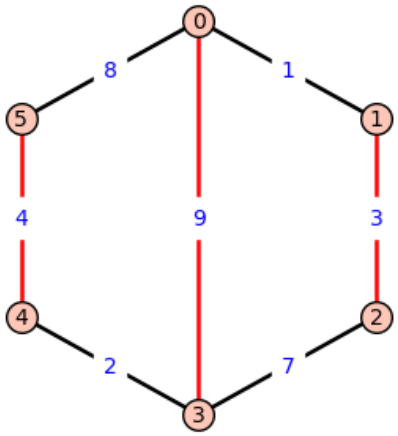
(a) Input graph G_9



(b) Input graph after subdividing edges, creating G'_9



(c) Weighted graph G_9



(d) Result of maximum matching

Figure 9: Subdividing edges for non-bipartite matching

References

[1] Jack Edmonds and Delbert Ray Fulkerson. Transversals and matroid partition. Technical report, Rand Corp Santa Monica CA, 1965.

[2] Eugene L. Lawler. Combinatorial optimization: Networks and matroids. 1976.

[3] USR Murty and Adrian Bondy. Graph theory (graduate texts in mathematics 244), 2008.

[4] J.G. Oxley. *Matroid Theory*. Oxford graduate texts in mathematics. Oxford University Press, 2006.

[5] The Sage Developers. *SageMath, the Sage Mathematics Software System (Version 9.0)*, 2020. `https://www.sagemath.org`.

[6] Dominic JA Welsh. *Matroid theory*. Courier Corporation, 2010.

A Code

A.1 Partition Matroid

Input: Accepts a partition of elements as a list of lists, where the inner lists are the blocks of the partition.

Output: Returns all bases of the partition matroid defined by the partition passed to the function.

```
def PartitionBase(edge_list):

    total_elements = 1

    for i in map(len, edge_list):
        total_elements *= i

    base = [set() for x in range(total_elements)]
    elements = total_elements

    for edges in edge_list:

        n = elements / len(edges)
        elements /= len(edges)
        rep = total_elements / (n * len(edges))

        for i, edge in enumerate(edges):
            for j in range(rep):
                for k in range(n):
                    base[i*n + j*(total_elements/rep) + k].add(edge)

    return base
```

A.2 Minimum Spanning Tree

Input: A SageMath graph object and a dictionary of weights indexed by the edge labels of the graph.

Output: A list of edges which provide a minimum weight spanning tree in the input graph.

```
def SolveGreedy(graph, weights):

    greedy_set = set()

    M = Matroid(graph)
    sorted_weights = sorted(weights.items(), key=operator.itemgetter(1))
    independent_sets = list(map(set, M.independent_sets()))

    for weight in sorted_weights:

        greedy_set.add(weight[0])

        if not (greedy_set in independent_sets):
            greedy_set.remove(weight[0])

    return list(greedy_set)
```

A.3 Bipartite Matching

Input: A SageMath bipartite graph object and a dictionary of weights indexed by the edge labels of the graph.

Output: A list of edges which provide a maximum weighted matching in the input graph.

```
def SolveBipartiteMatching(bipartite_graph, weights):

    temp_list = []
    edge_list = []

    for i in list(bipartite_graph.left):

        for j in bipartite_graph.edge_iterator([i]):
            temp_list.append(list(j)[2])

        edge_list.append(temp_list)
        temp_list = []

    M1 = Matroid(bases=PartitionBase(edge_list))

    temp_list = []
    edge_list = []

    for i in list(bipartite_graph.right):

        for j in bipartite_graph.edge_iterator([i]):
            temp_list.append(list(j)[2])

        edge_list.append(temp_list)
        temp_list = []

    M2 = Matroid(bases=PartitionBase(edge_list))

    intersect = M1.intersection(M2, weights)
    return(list(intersect))
```

A.4 Hamilton Cycle

Input: A directed graph as a SageMath digraph object, and the name of a vertex s to split into two vertices (the choice of this vertex is arbitrary and doesn't affect the results).

Output: Returns a list of edges which give a Hamilton cycle in the input graph.

```
def SolveHamiltonCycle(digraph, s):

    new_vertex = digraph.order() + 1
    digraph.add_vertex(new_vertex)

    for e in digraph.incoming_edge_iterator(s):
        digraph.delete_edge(e)
        digraph.add_edge(e[0], new_vertex, e[2])

    M = Matroid(digraph.to_undirected())
```

```
    p_out = []
    p_in = []

    for v in digraph.vertices():

        temp_out = []
        temp_in = []

        for e in digraph.outgoing_edge_iterator(v):
            temp_out.append(e[2])

        if len(temp_out) > 0:
            p_out.append(temp_out)

        for e in digraph.incoming_edge_iterator(v):
            temp_in.append(e[2])

        if len(temp_in) > 0:
            p_in.append(temp_in)

    for e in digraph.incoming_edge_iterator(new_vertex):
        digraph.delete_edge(e)
        digraph.add_edge(e[0], s, e[2])

    digraph.delete_vertex(new_vertex)

    p = ListIntersection(list(map(set, PartitionBase(p_out))),
    list(map(set, PartitionBase(p_in))))
    return list(ListIntersection(list(map(set, M.bases())), p)[0])
```

A.5 Matroid Parity Problem

Input: A matroid and a parity set represented as a list of sets, where each inner set contains two elements that are paired off together.

Output: A list of independent sets of the input matroid which are a valid parity set.

```
def SolveParityProblem(matroid, parity_set):

    result = set()
    independent_sets = list(map(set, matroid.independent_sets()))

    for L in range(1, len(parity_set)+1):
        for subset in itertools.combinations(parity_set, L):
            s = sum(set(subset), ())
            if set(s) in independent_sets:
                result.add(s)

    return result
```

A.6 Non-bipartite Matching

Input: A SageMath graph object to find feasible matchings on.

Output: List of all feasible matchings.

```
def SolveMatching(graph):

    parity_set = set()
    partition = []
    matchings = []

    edges = graph.edges()[:]
    vertices = graph.vertices()[:]
    s = graph.order()

    for i, e in enumerate(edges):
        graph.subdivide_edge(e, 1)
        graph.set_edge_label(e[0], (s + i), str(e[2] + '1'))
        graph.set_edge_label(e[1], (s + i), str(e[2] + '2'))
        parity_set.add((str(e[2] + '1'), str(e[2] + '2')))

    for v in vertices:
        partition.append(list(list(zip(*graph.edges_incident(v)))[2]))

    M = Matroid(bases=PartitionBase(partition))

    for match in SolveParityProblem(M, parity_set):
        temp_list = []

        for e in match:
            if e[1] == '1':
                temp_list.append(e[0])
        if(len(temp_list) > 1):
            matchings.append(temp_list)

    return matchings
```