Perfect Matching

Marco Daresta, Chiara Langella

University of Verona

December, 17th 2018

Structure of the project

The work is organize as follows:

- Chapter 1: Introduction to problem of Perfect Matching
- Chapter 2: Algorithm

Definition

A graph is an ordered pair G = (V,E) of sets where the elements of V are the vertices (or nodes) of G and the elements of E are its edges. In particular $E \subseteq \binom{V}{2}$ is the set of all possible pairs of elements of V

|V(G)| we denote the order of G, i.e the number of vertices of G;
|E(G)| we denote the size of G, i.e the number of edges of G

Definition

A vertex $v \in V$ is incident to an edge $e \in E$ if $v \in e$.
Two edges $e_1, e_2 \in E$ are incident (or adjacent) if they are a common vertex.
Two vertices $v_1, v_2 \in V$ are adjacent if there exist an edge e $\in E$ such that $e = v_1v_2$. Let G = (V,E) and G' = (V',E') be two graphs. If $V' \subseteq V$ and $E' \subseteq E$, then G' is a subgraph of G. Moreover, if $G \neq G'$, then we say that G' is a proper subgraph of G.

Definition

Let $G = (V, E)$ be a graph and let $v \in V$ be a vertex. The degree of a vertex v is defined as

$$d_G(V) := |N_G(v)|$$

where $N_G(v) := \{w \in V : vw \in E\}$ is the neighbour of v. In other words, it is the number of edges incident to the vertex $v \in V$. Next, we define

$$\delta(G) := \min_{v \in V} d_G(v)$$

the minimum degree of G, i.e. the degree of the vertex of less incident edges and

$$\Delta(G) := \max_{v \in V} d_G(v)$$

Definition

A graph G =(V,E) is bipartite if its vertex set V can be partitioned into two disjoint subsets $V = A \cup B$ such that every edge $e \in E$ has the form $v_1 v_2$, with $v_1 \in A$ and $v_2 \in B$.

Definition

Given a graph G = (V,E), a matching M(G) in G is a set of pairwise non-adjacent edges; that is, no two edges share a common vertex. Or, in an equivalent form, if every vertex of G is incident to at most an edge in M, i.e. $deg(v) \leq 1 \quad \forall v \in G$. This means that, in a matching M(G), all vertices can have either degree 0 or 1. In particular

- se deg(v) = 1, then the vertex v is matched (or saturated) if it is an endpoint of one of the edges in the matching.
- se deg(v) = 0, then the vertex v is unmatched.

In a matching, two edges cannot be incident: if they were, then the vertex incident to these two edges would have degree 2, but this is not possible by definition.

Definition

Given G = (V,E), a maximal matching is a matching M of a graph G with the property that if any edge not in M is added to M, it is no longer a matching, that is, M is maximal if it is not a subset of any other matching in graph G

Definition

A maximum matching M(G) is a matching that contains the largest possible number of edges. There may be many maximum matchings and such number is called matching number. Note that every maximum matching is maximal, but not every maximal matching is a maximum matching. The following figure shows examples of maximum matchings in the same three graphs.

A matching M(G) of a graph G is said to be perfect if every vertex $v \in G$ is incident to exactly one edge $e \in M$, i.e. if $deg(v) = 1 \forall v \in V$

From the definition above, we deduce:

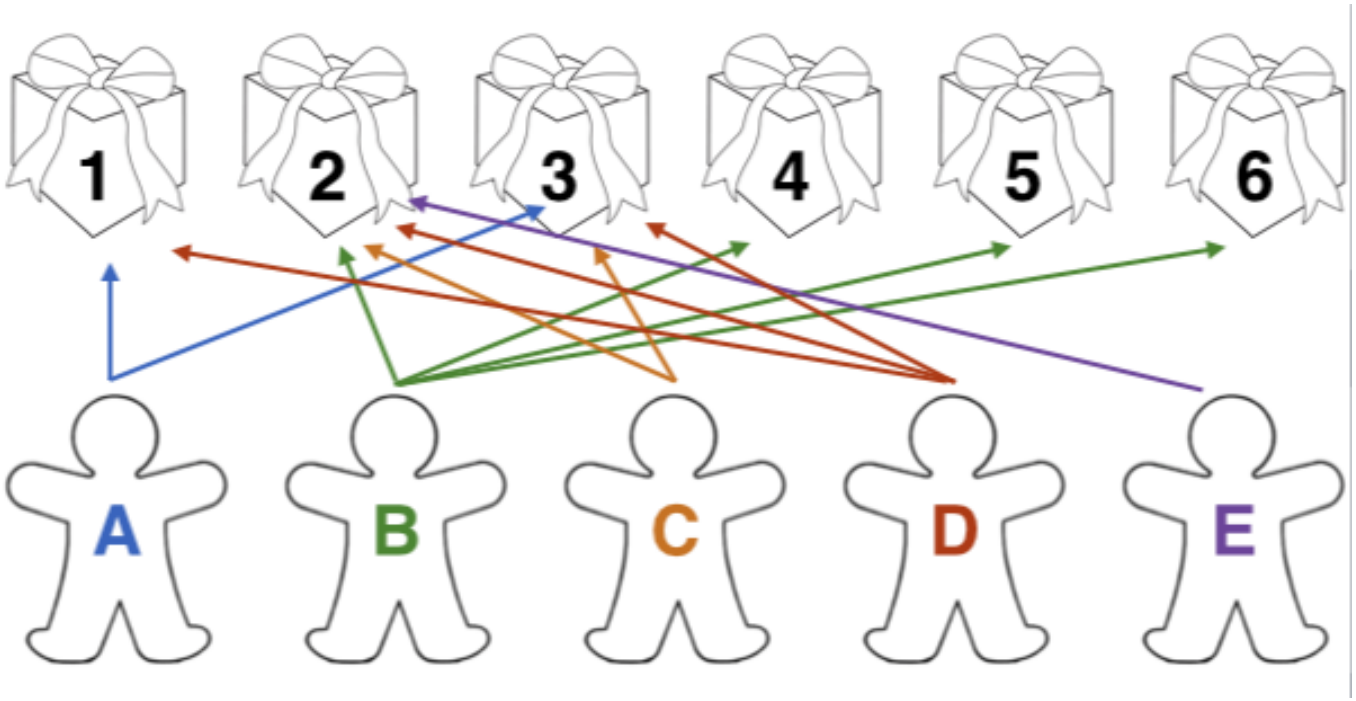
1. Every perfect matching of graph is also a maximum matching of graph, because there is no chance of adding one more edge in a perfect matching graph.. As a consequence, it is also maximal.
2. A maximum matching of graph is not necessary perfect.
3. If a graph G has a perfect matching, then the number of vertices |V(G)| is even. If it were odd, then the last vertex pairs with the other vertex, and finally there remains a single vertex which cannot be paired with any other vertex for which the degree is zero. It clearly violates the perfect matching principle.

Theorem (Hall, 1935)

A bipartite graph G admits a matching of A if and only if

$$|N(S)| \geq |S| \qquad \forall S \subseteq A$$

Suppose I have 6 gifts to give to 5 friends. Can I distribute one gift to each person so that everyone gets something they wish? We can model this situation as a graph partioned into two sets: the set of the gifts and the set of the friends



Let's check if it satisfies Hall's theorem: we consider a subset $X = \{A, C, D, E\}$, hence $|X| = 4$ and as a consequence $N(X) = \{1, 2, 3\}$, so $|N(X)| = 3$. From this, we deduce that $|X| \geq |N(X)|$, thus Hall's condition has been violated and so there no exists a matching. In other words, it is not possible to distribute to everyone the desired gift.

Definition

Let G = (V,E) be a graph. A vertex cover of G is a subset $U \subseteq V$ such that every edge $e \in E$ is incident to a vertex $v \in U$.

This means that every vertex in the graph is touching at least one edge. Vertex cover is a topic in graph theory that has applications in matching problems and optimization problems. A vertex cover might be a good approach to a problem where all of the edges in a graph need to be included in the solution. In particular, you ask to find minimum vertex cover.

The following result establishes a link between vertex cover and matching. In particular

Theorem (König, 1931)

In a bipartite graph G, the number of edges in a maximum matching is equal to the number of vertices in a minimum vertex cover.

Basic notions

Consider a graph $G = (V, E)$, where V set of *vertices* and E set of *edges*. Define:

- $\delta_G(S)$, $S \subseteq V$ is the set of those edges in E with precisely one end-vertex in S.
- A *graft* is the couple (G, T), where G is a connected graph in which an even number of vertices $T \subseteq V$ have been distinguished as *odd*.
- The *T-parity* of a set of vertices $S \subseteq V$ is the parity of $|S \cap T|$.

Hence S is *T-odd* then $\delta_G(S)$ is a *T-cut*.

Gomory-Hu Tree

Definition

Let $G = ((V_G, E_G), c)$ be an undirected graph with $c(u, v)$ being the capacity of the edge (u, v) respectively. Denote the minimum capacity of an $\{s, t\}$-cut by $\lambda_{s,t}$ for each $\{s, t\} \in V_G$. Let $T = (V_T, E_T)$ be a tree with $V_T = V_G$, denote the set of edges in an $\{s, t\}$ path by $P_{s,t}$ for each $\{s, t\} \in V_T$. Then T is said to be a **Gomory-Hu tree** of G if

$$\lambda_{s,t} = \min_{e \in P_{s,t}} c(S_e, T_e) \quad \text{for all} \quad \{s, t\} \in V_G,$$

where

1. S_e and T_e are the two connected components of $T \setminus \{e\}$ in the sense that (S_e, T_e) forms a s, t-cut in G.
2. $c(S_e, T_e)$ is the capacity cut in G.

The **Gomory-Hu tree** of an undirected graph, i.e. a graph without orientation, is a weighted tree that represents the minimum $\{s,t\}$-cuts for all $\{s,t\}$ pairs in the graph.
Given (G,T) a graft and $c : E \to \mathbb{R}_+$ be a *cost* function, a minimum T-cut for (G,T,c) is define as

$$c(\delta(X)) = \lambda_{G,T} = \min\{c(\delta(S)) : \delta(S) \quad \text{is a } T\text{-cut of} \quad (G,T)\}$$

where the cost $c(F)$ of a set F of edges is defined as $\sum_{e \in F} c(e)$.

Switching

Switching a set S means replacing S by $\bar{S}$. For example, if $S = X$, then after switching S we obtain that $S = \bar{X}$ and $\bar{S} = X$.

Observe that switching S does not change the T-parity of a S (nor $\delta(S)$), since $|S \cap T|$ and $|\bar{S} \cap T|$ have the same parity since $|T|$ is even.

Uncrossing

If $S \cap X \neq \emptyset$ for every possible switching of S and X, then S and X are said to *cross*.

Lemma

Let T_1, T_2 be even cardinality subsets of V. Let $\delta(S_1)$ be a minimum T_1-cut and assume that S_1 is T_2-even. Then there exists a minimum T_2-cut $\delta(S_2)$ such that S_1 and S_2 do not cross.

Consider a graft (G, T) and a T-even set $S \subseteq V$, denoted by G_S the graph obtained from G by identifying all nodes in S into a single node and letting $T_S := T \setminus S$. Note that (G_S, T_S) is a graft. When $S = \{s, t\} \subseteq T$ then we rely on a shorter notation $G_{s,t} = G_{\{s,t\}}$ and $T_{s,t} = T_{\{s,t\}}$.
Since node identification does not affect the edge set of a graph, a cost function c for G is also a cost function for G_S and $G_{s,t}$.

A simple algorithm

We show four steps to compute $\lambda_{G,T}$:
`MinT-cut(G,T,c)`

1. if $T = \emptyset$ then return ∞, that means (G, T) does not contains any T-cut;
2. let s and t be any two different nodes in T;
3. let $\delta(S)$ be a minimum $\{s, t\}$-cut;
4. if S is T-odd then return $\min(c(\delta(S)), \texttt{MinT-cut}(G_{s,t}, T_{s,t}, \texttt{c}))$; otherwise return $\min(\texttt{MinT-cut}(G_S, T_S, \texttt{c}); \texttt{MinT-cut}(G_{\bar{S}}, T_{\bar{S}}, \texttt{c}))$

T-pairing

Let (G, T) be a graft with cost function $c : E \longmapsto \mathbb{R}_+$. A *$T$-pairing* is a partition of T into pairs. The value of the T-pairing $\mathcal{P}$ is defined as:

$$val_G(\mathcal{P}) = \min_{\{u,v\} \in \mathcal{P}} \lambda_G(u, v)$$

where $\lambda_G(u, v)$ denotes the cost of a minimum $\{u, v\}$-cut, Let $\mathcal{P}$ be any T-pairing and $\delta(S)$ be any T-cut. Since $\delta(S)$ is T-odd, $\mathcal{P}$ contains a pair $\{u, v\}$ such that $\delta(S)$ is $\{u, v\}$-odd. Therefore, $c(\delta(S)) \geq \lambda_G(u, v) \geq val_G(\mathcal{P})$ and the value of $\mathcal{P}$ is a lower bound on $\lambda_{G,T}$.

MinT-cut finds a T-pairing

Let s and t be two odd nodes. Let $\delta(S)$ be a minimum $\{s,t\}$-cut.

Lemma (2.7)

Let $\delta(S)$ be a minimum $\{s,t\}$-cut in (G,c). Then we have:

$$\lambda_G(u,v) \geq \min(c(\delta(S)), \lambda_{G_{s,t}}(u,v)) \qquad \forall u,v \in V(G) \setminus \{s,t\}$$

Lemma (2.8)

Let $\delta(S)$ be a minimum $\{s,t\}$-cut in (G,c). Then we have:

$$\lambda_G(u,v) = \lambda_{G_s}(u,v) \qquad \forall u,v \in V(G) \setminus S$$

Each iteration of the algorithm contemplates two possibilities:

1. **Case 1**: $\delta(S)$ is T-odd. By Lemma 2.5, $\lambda_{G,T} = \min\{c(\delta(S)), \lambda_{G_{s,t},T_{s,t}}\}$. By Lemma 2.7, if $\mathcal{P}'$ is a $T_{s,t}$-pairing with $val_{G_{s,t}}(\mathcal{P}') = \lambda_{G_{s,t},T_{s,t}}$, then $\mathcal{P} = \mathcal{P}' \cup \{s,t\}$ is a T-pairing with $val_G(\mathcal{P}) \geq \min(c(\delta(S)), val(\mathcal{P}')) = \min(c(\delta(S)), \lambda_{G_{s,t},T_{s,t}}) = \lambda_{G,T}$.
2. **Case 2**: if $\delta(S)$ is T-even. By Lemma 2.6, $\lambda_{G,T} = \min\{\lambda_{G_S,T_S}, \lambda_{G_{\bar{S}},T_{\bar{S}}}\}$. By Lemma 2.8, if $\mathcal{P}_\mathcal{S}$ is a T_S-pairing with $val_{G_S}(\mathcal{P}_\mathcal{S}) = \lambda_{G_S,T_S}$ and $\mathcal{P}_{\bar{\mathcal{S}}}$ is a $T_{\bar{S}}$-pairing with $val_{G_{\bar{S}}}(\mathcal{P}_{\bar{\mathcal{S}}}) = \lambda_{G_{\bar{S}},T_{\bar{S}}}$ then $\mathcal{P} = \mathcal{P}_\mathcal{S} \cup \mathcal{P}_{\bar{\mathcal{S}}}$ is a T-pairing with $val_G(\mathcal{P}) \geq \min(val(\mathcal{P}_\mathcal{S}), val(\mathcal{P}_{\bar{\mathcal{S}}})) = \min(\lambda_{G_S,T_S}\lambda_{G_{\bar{S}},T_{\bar{S}}}) = \lambda_{G,T}$.

Summary

The following results summarize this section.

Theorem

For every (G, T, c) the maximum value of a T-pairing equals the minimum cost of a T-cut.

Theorem

For every node u in T there exists a node $v \in T \setminus \{u\}$ such that $\{u, v\}$ is useful.

Scheme

We have developed an algorithm on Python using both Gurobi and Python languages. It can be run using Spyder.

1. In the first part, we gave four important functions: `Contraction`, `minCutValue`, `findsubsets` and `minCutEdges`;
2. After that, we developed the model (constraint and objective function). The final `while` cycle optimize the model.

USEFUL MODELS

```
import math
import random
from gurobipy import *

import networkx as nx
import itertools
```

FUNCTIONS DEFINITION

```
# Given a graph, this function identifies
#a set of nodes into a single vertex

def Contraction(graph, nodes):
    V = set()
    for i,j in graph:
        V = V.union({i})
        V = V.union({j})
    new_vertex = max(V) + 1
    # Create a minor of the graph
    minor = {}
    for i,j in graph.keys():
        [...]
```

```
# Given a graph, this function finds the value of a minimum odd cut
# T is a datum used during the recursion:
#it must be initially set equal to "vertices"
def minCutValue(graph, T):
    # Base step
    if T == set():
        return float('inf')
    # Create the graph using the module NETWORKX
    G = nx.DiGraph()
    for (i,j) in graph.keys():
        G.add_edge(i, j, capacity = graph[i,j])
        G.add_edge(j, i, capacity = graph[i,j])
```

```
# Let s and t be two random nodes in T
    s = min(T)
    t = max(T)
    # NETWORKX contains a function that allows to find the minimum s-t cut
    # This function is based on Ford-Fulkerson algorithm
    cut_value, partition = nx.minimum_cut(G,s,t)
    S, S_bar = partition
    if len(S.intersection(T)) % 2 == 1: # if S is T-odd: ...
        return min(cut_value,
                   minCutValue(Contraction(graph, {s,t}), T - {s,t}))
    else:
        return min(minCutValue(Contraction(graph, S), T - S),
                   minCutValue(Contraction(graph, S_bar), T - S_bar))
```

```
# Given a set, this function finds
#all its subsets of a fixed cardinality

def findsubsets(S,m):
    # From ITERTOOLS module
    return set(itertools.combinations(S, m))
```

```
def minCutEdges(graph, cut_value):
    for cardinality in range(1,n,2):
        subsets = findsubsets(vertices, cardinality)
        # "vertices" is global variable
        for S in subsets:
            # Create the cut
            dS = {}
            for i,j in graph.keys():
                if i in S and j not in S:
                    dS[i,j] = graph[i,j]
                if i not in S and j in S:
                    dS[i,j] = graph[i,j]
            dS_value = sum(dS.values())
            if dS_value == cut_value:
                return dS
    print('No odd cut of value %g has been found' % cut_value)
    return None
```

MODEL DEFINITION

```
m = Model()
# Suppress the standard output, il would be invoked too many times
m.setParam('OutputFlag', 0)

# Create the variables
vars = {}
for (i,j) in edges.keys():
   vars[i,j] = m.addVar(lb=0, vtype=GRB.CONTINUOUS, name='e[%d,%d]'%(i,j))
```

```
# To create the model data structure only once,
#after variables creation
m.update()

# Add the objective function
m.setObjective(sum(vars[i,j]*edges[i,j] ...
...for i,j in edges.keys()), GRB.MINIMIZE)
```

```
# Add degree-1 constraint
for i in vertices:
    # Look for the edges incident to i
    neighbors = []
    for j in vertices:
        if (i,j) in edges.keys():
            neighbors.append((i,j))
        elif (j,i) in edges.keys():
            neighbors.append((j,i))
    m.addConstr(sum(vars[i,j] for i,j in neighbors) == 1)
```

MODEL OPTIMIZATION

```
# Check the parity of the graph
elif n % 2 == 1:
    print('The graph is odd!')
else:
    # Now we are sure that the graph admits a solution
    while True:
        # Solve the problem
        m.optimize()
        print('\n-----------------------------------\n')
        solution = m.getAttr('x', vars)
        selected = [(i,j) for i,j in solution.keys() if solution[i,j] != 0]
        print('New incumbent solution:\n')
```

```
for i,j in selected:
            print('    ' + str((i,j)) + ' : ' + str(solution[i,j]))
        print('\nObjective function: %g' % m.objVal)
        print('Checking the presence of rationals...')
        # Find the value of a minimum odd cut
        cut_value = minCutValue(solution, vertices)
```

```
if cut_value < 1:
    # Find the edges of a minimum odd cut
    cut_edges = minCutEdges(solution, cut_value)
    # Add a constraint
    m.addConstr(sum(vars[i,j] for (i,j) in cut_edges.keys()) >= 1)
    print('Minimum odd cut: %g' % cut_value)
    print('Related edges:\n')
    for i,j in cut_edges.keys():
        print('    ' + str((i,j)) + ' : ' + str(cut_edges[i,j]))
    print('\nNew constraint added...')
else:
    print('This is a feasible solution!')
    break
```

Conclusions and further works

The following changes can be made:

- the dictionaries (.keys inside the code) could be used for the vertices and not for the arcs, but in this case we should solve the problem of the weights to assign to the arcs;
- the examples were placed in a separate python file and it could be able to structure it as a text file and recall it in the python matching file;

The initial idea was to use the Call-Back functions, as in TSP problem. Unfortunally, Gurobi does not allow the user to use the value of the variables except in the MIP phase, i.e. in a phase of the process in which there are only whole values of the solutions.

Riferimenti bibliografici


R. DIESTEL, *Graph Theory*, Electionic Edition, 2005


J.M. HARRIS, J.L.HIRST, M.J. MOSSINGHOFF, *Combinatorics and Graph Theory*, Springer Edition, 2008


R.RIZZI, *A Simple Minimum T-Cut Algorithm*, 2002