

On Decompositions, Generation Methods, and related concepts in the theory of Matching Covered Graphs

Student: Janmenjaya Panda

Roll No: ME20B087

Degree: B. Tech.

Guide: Nishad Kothari

Degree: is pertinent to the case of tight cut decomposition, dependency relations, tight cut decomposition, brace and brace generation methods and related concepts in the theory of matching covered graphs.

This PR introduces new methods per [Guideline 1](#) and matches [Nishad Kothari](#) following:

- The creation of a new class `MatchingCoveredGraph`, in the module `sage.graphs.matching_covered_graph` and the implementation of the following functions:
- `maximal_barrier()`: Return the (unique) maximal barrier of the graph containing the (provided) vertex.
 - `canonical_partition()`: Return the canonical partition.
 - `tight_cut_decomposition()`: Return a maximal set of disjoint (s, t) -cuts of the graph as the shores with these cuts of the (matching covered) graph.
 - `bricks_and_braces()`: Return the list of (undirected) bricks and braces of the (matching covered) graph.
 - `number_of_bricks()`: Return the number of bricks.
 - `number_of_braces()`: Return the number of braces.
 - `number_of_petersen_bricks()`: Return the number of Petersen bricks of the graph of which the graph is isomorphic to `graphs.PetersenGraph()`.
 - `is_BRICK()`: Check if the (matching covered) graph is a brick.
 - `is_BRACE()`: Check if the (matching covered) graph is a brace.
 - `is_renewable_edge()`: Check if the edge of the (matching covered) graph is a renewable edge.



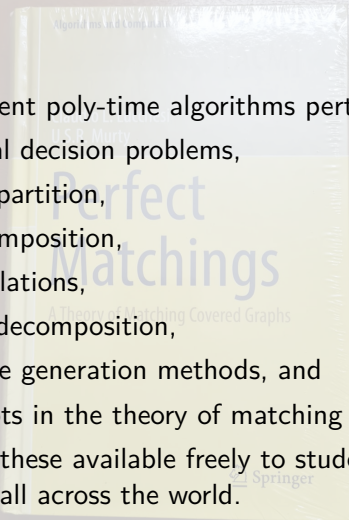
Indian Institute of Technology Madras

Problem Statement

Implementing efficient poly-time algorithms pertaining to

- ▶ the fundamental decision problems,
- ▶ the canonical partition,
- ▶ tight cut decomposition,
- ▶ dependency relations,
- ▶ (optimal) ear decomposition,
- ▶ brick and brace generation methods, and
- ▶ related concepts in the theory of matching covered graphs,

and to make all of these available freely to students, educators as well as researchers all across the world.

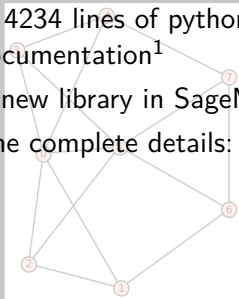


- ▶ Compilation of all existing algorithms available in the literature
- ▶ Devise of new/ equivalent definitions, theorems and derivations of efficient poly time algorithms from constructive/ existential proofs
- ▶ Implementing each of those in python/ cython with SageMath as the code base with the relevant documentations, examples and test cases
- ▶ Communicating with SageMath through Google Summer of Code (GSoC) 2024 to make all of these available publicly.

Novelty and Results


```
1 G = Graph()
2 G.add_edges([(0, 1), (0, 2), (0, 3), (0, 4), (0, 5),
3             (1, 2), (1, 6), (2, 5), (3, 5), (3, 4),
4             (4, 7), (5, 6), (5, 7), (6, 7)])
5 M = Graph(G.matching())
6 show(G)
7
8 try:
9     answer = is_brick(G, M, matching_covered_check=True)
10    print('Is G a brick? {}'.format(answer))
```

- ▶ 31 functional methods, 12 graph generator methods and 21 algorithms (10 existing algorithms, 11 formulated/ derived)
- ▶ ≈ 4234 lines of python/ cython code and ≈ 8807 lines of documentation¹
- ▶ A new library in SageMath: Matching Covered Graphs²
- ▶ The complete details: [Link](#)




... Is G a brick? True

¹An estimation

²Note that it will be made public with the next (10.4) release of SageMath  4/5

Relevance



Sage 10.4.beta9 Reference Manual

Q Search

Files

- Home
- Generic graphs (could also be directed)
- Undirected graphs
- Bipartite graphs
- Matching
- View classes
- Common digraphs
- Common undirected graphs
- Graph database
- Databases of graphs
- Families of graphs derived from classical geometries over finite fields
- Various families of graphs
- Basic graphs
- Chessboard graphs

```
is_matching_coveredatching', algorithm=None, onp_certificate='Edmonds', solver=False, verbose=None, integrality_tolerance=0) [source]
```

Check if the graph is matching covered.

A connected nontrivial graph wherein each edge participates in some perfect matching is called a *matching covered graph*.

If a perfect matching of the graph is provided, for bipartite graph, this method implements a linear time algorithm as proposed in [LM2024] that is based on the following theorem:

Given a bipartite graph $G = (V_1 \cup V_2, E)$, let M be a perfect matching of G . Then there exists a graph D from G such that $V(D) := V(G)$ and for each edge in G direct the corresponding edge from A to B in D , if it is in M or otherwise direct it from B to A . The graph G is matching covered iff D has no directed cycle.

For nonbipartite graph, if a perfect matching of the graph is provided, this method implements an $O(V^3)$ algorithm where $|V|$ denotes the number of vertices of the graph. Otherwise, respectively, this implementation is inspired by the $O(E^2)$ algorithm of [LZ2001] and the one explained in [LZ2001]. For nonbipartite graph, the implementation is based on the following Berge-Fulkerson conjecture:

Given a nonbipartite graph G with a perfect matching M . The graph G is matching covered if and only if for each edge uv not in M , there exists an M -alternating odd length uv -path starting and ending with edges not in M .

The time complexity may be dominated by the time needed to compute a maximum matching of the input graph. If the input graph is not connected, we have to check for each component if it is trivial graph, a `ValueError` is returned.

- `onp_certificate` – (default: `'None'`): a perfect matching of the graph, that can be used to solve any problem in polynomial time.
- If set to `None`, a matching is computed using the other parameters.
- `algorithm` – string (default: `'Edmonds'`): the algorithm to be used to compute a maximum matching of the graph among
 - `'Edmonds'`: selects Edmonds' algorithm as implemented in NetworkX,
 - `'LP'`: uses a Linear Program formulation of the matching problem.

ON THIS PAGE

- Graph Format
- Supported formats
- Generators
- Labels
- Database
- Visualization
- Mutability
- Network
- `k_shortest_paths_networkx()`
 - `laplacian_eigenvalues()`
 - `list_cycles()`
 - `list_spanning_trees()`
 - `max_flow_value()`
 - `min_cut_value()`
 - `number_of_connected_components()`
 - `number_of_isolated_vertices()`
 - `number_of_loops()`
 - `number_of_self_loops()`
 - `bipartite_double()`
 - `bipartite_sets()`
 - `bounded_outdegree_n()`
 - `C()`
 - `chromatic_number()`
 - `chromatic_polynomial()`
 - `chromatic_index()`
 - `chromatic_number()`
 - `chromatic_symmetric_function()`
 - `clone()`
 - `clone_complex()`
 - `clone_maxflow()`
 - `clone_number()`

Matchings and perfect matchings: A fundamental basis of graph theory, algorithms and optimization

Existence of several open problems (Barnette's conjecture, Berge-Fulkerson conjecture, and so on): use matchings at their heart

A free complete public efficient library/module for Matching covered graphs in sage: a solution to numerical and computational aspect of research in matching theory