

参赛学生姓名：张璟霖、颜梓晨

中学：华南师范大学附属中学

省份：广东省

国家/地区：中国

指导老师姓名：桂鹏

指导老师单位：华南师范大学附属中学

论文题目：The Maximum Sachs Subgraph:

Computational Algorithm and its Connection

with Fractional Matching and Perrank

Title of Research Report: The Maximum Sachs Subgraph:
Computational Algorithm and its Connection with Fractional
Matching and Perrank

Author: Keng Lam Zhang, Zichen Yan

Abstract: We present a polynomial-time algorithm that determines both the existence and the maximum cardinality (in terms of vertices covered) of a Sachs subgraph in an undirected simple graph. Our algorithm leverages the connection between Sachs subgraphs and fractional matchings, specifically exploiting the half-integrality property of extreme points in the fractional matching polytope. By solving a specific linear program, we obtain an optimal extreme-point solution such that the support graph of positive-weight edges naturally decomposes into a matching and vertex-disjoint odd cycles, forming a maximum Sachs subgraph. Furthermore, we establish a fundamental equivalence between three seemingly distinct graph-theoretic notions: the maximum Sachs subgraph, perrank, and fractional matching.

Keywords: Sachs subgraph, factors, perrank, fractional matching, 2-matching, bipartite covering

Table of Content

1. Introduction	4
2. Preliminaries	8
3. Main Theorem	12
4. Algorithm	14
5. Potentials for Further Research	24
References	24
Acknowledgement (in Chinese)	26

1 Introduction

1.1 Research Background

In this paper, we study Sachs factors and Sachs subgraphs in undirected graphs, defined below:

Definition 1. A *Sachs subgraph* S is a subgraph whose connected components are exclusively K_2 edges or cycles, and is a *Sachs factor* only if it covers every vertex of G .

The above concept was introduced by the German mathematician Horst Sachs in 1964 in his research on the coefficient of the characteristic polynomial for any undirected simple graph G (we will go into the details below)[9]. Since then, the Sachs subgraph has served the purpose of aiding structural analysis in graphs.

Sachs subgraphs are closely related to matchings, cycle-factors, and $\{1, 2\}$ -factors.

Definition 2. A *matching* is a set of edges in a graph such that no two of which share a common end vertex. A matching is called *perfect* if it covers all vertices.

Definition 3. A *factor*, in general, is a subgraph which covers all vertices of a graph. In particular, a k -*factor* is a subgraph covering the entire vertex set of the original graph with each vertex having a degree of k ; an $[a, b]$ -*factor* also covers all the vertices, where the degree of each vertex may take up any integer value in the interval $[a, b]$

- The research on matchings, $\{1, 2\}$ -factors, and Sachs subgraphs has a long history. While the former is not a subject of discussion here, its extensive similarity with Sachs subgraphs still makes it a reliable metric of research when in comparison with Sachs subgraphs.

Factors have been much thoroughly investigated from the perspective of existence conditions. The first mention of n -factor was in 1891, as Danish mathematician Petersen proved a theorem under his name, stating that every cubic, bridgeless graph contains a perfect matching[7]. It wasn't until decades later, though, that Tutte provided a formal definition of n -factor[11], three years after his initial discovery of the famous 1-factor theorem in 1947[10]. Another 23 years later, Lovász expanded n -factor into $\{g, f\}$ -factor in 1970, also proving a necessary and sufficient condition for a $\{g, f\}$ -factor to exist[5].

One may have noticed from the definition that a Sachs factor has subtle differences in definition from a $\{1, 2\}$ -factor. Unlike its ancestor, though, the Sachs subgraph has been the topic of discussion only in topics such as k -Sachs-critical graphs[4], the singularity of Sachs subgraphs[6], and Sachs subgraphs and even their application in chemistry (for instance as part of a test for the existence of non-bonding orbitals in hydrocarbon structures) have been thoroughly discussed over the past decades. They have also hastened the need for a search algorithm similar to those for $\{1, 2\}$ -factors, a field that, regrettably, remains somewhat under-studied. It is under such circumstances that prompted the development of such a search algorithm.

1.2 Sachs' Theorem

When Sachs first introduced the Sachs subgraph in 1964, he also introduced a theorem on the connection between Sachs subgraphs and the characteristic polynomial of an undirected simple graph[9].

A definition of the characteristic polynomial of G is provided to better explain the theorem:

Definition 4. If an undirected simple graph G contains n vertices and corresponds to adjacency matrix A , then its *characteristic polynomial* is defined as

$$P_G(\lambda) = \det(\lambda I - A) = \lambda^n + a_1\lambda^{n-1} + a_2\lambda^{n-2} + \dots + a_n,$$

where λ refers to the eigenvalues of A .

Sachs' theorem provides the following formula that dictates the value of a_i :

Theorem 1. For coefficient a_i of the characteristic polynomial of G ,

$$a_i = \sum_{H \in \mathcal{S}_i} (-1)^{c(H)} 2^{s(H)},$$

where $\mathcal{S}_i$ is the set of all Sachs subgraphs of G that contain exactly i vertices, $c(H)$ is the total number of connected components in the subgraph H , and $s(H)$ is the number of cycles in the subgraph H .

1.3 Perrank and the Maximum Sachs Subgraph

Additionally, one may create a derivative of the characteristic polynomial by replacing the determinant in the original definition with a permanent:

Definition 5. If an undirected simple graph G contains n vertices and corresponds to adjacency matrix A , then its *permanental polynomial* is defined as

$$\text{Per}(G, \lambda) = \text{per}(\lambda I - A),$$

where

$$\text{per}(A) = \sum_{\sigma \in S_n} \prod_{i=1}^n A_{i, \sigma(i)}.$$

Similar to that of rank in determinant, the perrank in permanent is defined as follows:

Definition 6. The *perrank* of a matrix A is the size of the maximum square submatrix of A with non-zero permanent. Alternatively, for an undirected simple graph G , $\text{perrank}(G)$ is the order of the maximum subgraph of G whose adjacency matrix has non-zero permanent.

Under such a definition, notice how for ordinary non-negative adjacency matrices, every term $2^{s(H)}$ in Sachs' theorem is strictly positive, while the permanent sum, unlike the determinant sum where terms can cancel out to zero due to alternating signs $(-1)^{c(H)}$, has no negative signs and no cancellations. Therefore, from the perspective of Sachs subgraphs, $\text{perrank}(G)$ is the order of the maximum Sachs subgraph of $G[1]$.

1.4 Motivation and Our Contribution

Since characterizing the structural features of Sachs subgraphs can be hard to commence, reduction had been a preferred measure in the design of the algorithm; in other words, we attempted to search for a connection of sorts that may relate Sachs subgraphs with a more studied field.

After several failed attempts to correlate the maximum Sachs subgraph with $\{1, 2\}$ -factors, we turned to bipartite graphs; this was largely a result of the two types of connected components in Sachs subgraphs. Through a stable process, an algorithm was created that could effectively confirm the existence of a Sachs factor— in some way, this means that the job is half-done. However, when it comes to determining the order of the maximum Sachs subgraph, it has come to realization that such a process fails to maintain the stability of the subgraph's order: with different configurations, distinct numbers of points were generated in the new graph, and there exists no concise formula of any sort that may determine the number of vertices of the original maximum Sachs subgraph. Thus, we need something else to consolidate the link.

It took another month before the idea of introducing fractional matching finally came in. While the problem of maximum matching can be formulated as an integer programming problem, fractional matching can be viewed as its LP relaxation. Also, thanks to the dedication of mathematicians such as L. G. Khachian and N. Karmarkar, several solution algorithms have already been devised that may effectively characterize the matchings; in fact, as early as 2013, Google had already devised LP solvers as part. Also, fractional matching has been adopted previously as a means to study bipartite graphs; see [3]. After two months of discussion and arduous writing, the final work is culminated here in this essay.

In brief words, this essay will attempt to address this gap by providing an algorithm that may determine the existence of Sachs factor as well as the maximum size of a Sachs subgraph with a polynomial time complexity of $O(|E||V| + |V|)$, and by adopting the concept of extreme points in fractional matching, this essay creates an equivalence relationship between maximum Sachs subgraph, fractional matching, and perrank.

2 Preliminaries

2.1 Fractional Matching

Apart from Sachs' theorem, the Sachs subgraph has pivotal connections to other fields in graph theory as well; an example of this involves extreme points in fractional matching, a fractional generalization of matching that associates not only with Sachs subgraphs but also other aspects of graph theory and may help to establish a link between Sachs subgraphs and other fields.

As with characteristic and permanent polynomials, we will first provide an explanation of what fractional matching is:

Definition 7. Given a graph G , a *fractional matching* in G is a mapping f that assigns to each edge $e \in E$ a fraction $f(e) \in [0, 1]$ such that for every vertex $v \in V$, the sum of fractions of edges adjacent to v is at most one. In other

words, if the set of edges adjacent to vertex v is denoted as $\delta(v)$, then f must satisfy

$$\sum_{e \in \delta(v)} f(e) \leq 1, \quad \forall v \in V.$$

From the perspective of linear algebra, an $|E|$ -dimensional point may be constructed with the coordinate vector

$$\mathbf{w} = (f(e_1), f(e_2), \dots, f(e_{|E|})) \in \mathbb{R}^{|E|}.$$

Since there exist value limitations on $f(e)$ and $\sum_{e \in \delta(v)} f(e)$, the set of points created from all the mappings would form a convex polytope $\mathcal{P}(G) \subset \mathbb{R}^{|E|}$.

2.2 Extreme Points and Half-Integrality

Definition 8. A fractional matching is an *extreme point* if it is the "corner" or "vertex" point of $\mathcal{P}(G)$.

Since $\mathcal{P}(G)$ is convex, a natural algebraic property of an extreme point f is that there do not exist two distinct feasible matchings $f_1, f_2 \in \mathcal{P}(G)$ ($f_1 \neq f_2$) and a scalar $\lambda \in (0, 1)$ such that

$$f = \lambda f_1 + (1 - \lambda) f_2.$$

Even though the values assigned by a fractional matching may be any value in the interval $[0, 1]$, Balinski[2] established the following result for extreme points:

Theorem 2. *Every extreme point of the polytope $\mathcal{P}(G)$ assigns edge weights $f(e)$ strictly from the set*

$$\left\{0, 1, \frac{1}{2}\right\}.$$

Moreover, its weight-1 edges form a matching and its weight- $\frac{1}{2}$ edges form vertex-disjoint odd cycles, disjoint from the matching.

A sketch of Balinski's proof goes as follows:

Proof. Assume the contrary. Suppose $f \in \mathcal{P}(G)$ is an extreme point of the polytope, but with at least one edge e whose weight is not in $\{0, \frac{1}{2}, 1\}$.

Define the subgraph induced by the following set of non-half-integral edges as a "bad" subgraph:

$$E_{\text{bad}} = \left\{ e \in E \mid f(e) \notin \left\{ 0, \frac{1}{2}, 1 \right\} \right\}.$$

To arrive at a contradiction, one may construct a non-zero direction vector $x \in \mathbb{R}^{|E|}$ supported only on E_{bad} such that for a sufficiently small $\epsilon > 0$, both perturbed solutions

$$f_1 = f + \epsilon x \quad \text{and} \quad f_2 = f - \epsilon x$$

remain fully feasible inside the polytope $\mathcal{P}(G)$: if such an x exists, then f can be written as

$$f = \frac{1}{2}f_1 + \frac{1}{2}f_2,$$

which violates the extreme point condition.

To maintain feasibility, the vector x must satisfy two constraints: for every vertex v with a tight degree constraint ($\sum_{e \in \delta(v)} f(e) = 1$),

$$\sum_{e \in \delta(v)} x(e) = 0;$$

meanwhile, small ϵ automatically preserves

$$0 \leq (f \pm \epsilon x)(e) \leq 1$$

as $0 < f(e) < 1$ for all edges in E_{bad} .

To allow such a perturbation, we will now turn to structural analysis of the "bad" subgraph.

Consider one connected component C of S . Every leaf of C has a slack capacity constraint: its single incident fractional edge has weight below 1 and it has no other positive incident edge. A tree component has at least two such leaves. Its edge columns therefore outnumber the available tight-vertex equations, contradicting the independence just established. More explicitly, a

tree with t vertices has $t-1$ edges, whereas its two slack leaves leave at most $t-2$ tight-vertex equations. This also handles a component with a single fractional edge.

A component with one cycle and an attached tree has a slack leaf, so the number of tight-vertex equations is less than the number of edges. On t vertices, a component with more than one cycle has at least $t+1$ edges but at most t equations. Therefore a component of S can only be a cycle whose vertex constraints are all tight.

On an even cycle, alternating plus and minus signs give a nonzero null vector for the incidence matrix. Hence the cycle is odd. The tight equations on an odd cycle are $f(e_{prev}) + f(e_{next}) = 1$ at each vertex; following them around the cycle forces every edge weight to be $\frac{1}{2}$. Equivalently, walking around an odd cycle alternates a starting weight and its complement; the last equation forces the starting weight to equal its complement. Distinct components are vertex-disjoint.

Furthermore, a weight-1 edge cannot touch a fractional edge because its endpoints are saturated. This proves both the half-integrality and the structural assertion.

Half-integrality without extremality would not suffice: a two-edge path with both weights $\frac{1}{2}$ is feasible but does not form an odd cycle. ■

This result can be quite effective when applied to characterize the structures in a Sachs subgraph; specifically, for any edge $e \in G$, a value may be assigned under the following mapping for a given Sachs subgraph H in G :

$$f(e) = \begin{cases} 1, & e \text{ is a } K_2 \text{ component of } H, \\ \frac{1}{2}, & e \text{ belongs to a cycle component of } H, \\ 0, & e \text{ does not belong to } H. \end{cases}$$

Under such an assignment, since the set of vertices in distinct branches of H never converge, all vertices v covered by H in G satisfy

$$\sum_{e \in \delta(v)} f(e) = 1,$$

while all the remaining vertices in G satisfy

$$\sum_{e \in \delta(v)} f(e) = 0.$$

This mapping converts a Sachs subgraph into a feasible fractional matching. It need not produce an extreme point when the Sachs subgraph contains an even cycle— a component that can be deleted for ensuring an extreme point construction.

3 Main Theorem

3.1 A Key Lemma

The following lemma may remove the factor of even cycles from consideration of the maximum Sachs subgraph:

Lemma 1. *There exists a maximum Sachs subgraph with only K_2 or odd cycles as connected components.*

Proof. If a Sachs subgraph contains the even cycle

$$C = v_1 v_2 \dots v_{2k} v_1,$$

then it may be replaced with alternating path

$$v_1 v_2, v_3 v_4, \dots, v_{2k-1} v_{2k},$$

which covers the original vertices in C .

Applying the replacement on every even cycle results in a Sachs subgraph with only K_2 and odd cycles. ■

• In matching theory, such a structure is defined as a basic 2-matching:

Definition 9. A *basic 2-matching* consists of matchings (from edges with weight 1) and odd cycles with no overlapping vertices (from edges with weight $\frac{1}{2}$).

Qian established the result that extreme points of fractional matching contain basic 2-matchings[8].

3.2 Main Theoretical Results

The following identity follows from the classical extreme-point characterization of fractional matchings and the characterization of perrank by maximum Sachs subgraphs. We include a direct argument to explain how these results yield the optimization formulation used in Section 4.

Theorem 3. *For an undirected simple graph G ,*

$$\text{perrank}(G) = 2\nu_f(G),$$

where

$$\nu_f(G) = \max_{f \in \mathcal{P}(G)} \sum_{e \in E(G)} f(e).$$

Proof. Let

$$s(G) = \max\{|V(H)| : H \text{ is a Sachs subgraph of } G\}.$$

Under our definition of perrank, $s(G) = \text{perrank}(G)$.

We prove that $s(G) = 2\nu_f(G)$.

First, let H be any Sachs subgraph. Assign weight 1 to each edge that is a K_2 component of H , weight $1/2$ to each edge in a cycle component of H , and weight 0 to all remaining edges of G . This assignment is a feasible fractional matching x . Every vertex of H has incident weight exactly 1, so

$$|V(H)| = 2 \sum_{e \in E(G)} x_e \leq 2\nu_f(G).$$

Taking the maximum over H gives $s(G) \leq 2\nu_f(G)$.

Conversely, the fractional matching polytope is nonempty and bounded, so it has an optimal extreme point x^* . By the extreme-point characterization, the edges with weight 1 form a matching, while the edges with weight $1/2$ form vertex-disjoint odd cycles, disjoint from that matching. Consequently, the positive support of x^* , with isolated vertices omitted, is a Sachs subgraph H^* . Every vertex of H^* has incident weight exactly 1. Hence

$$|V(H^*)| = 2 \sum_{e \in E(G)} x_e^* = 2\nu_f(G),$$

which gives $s(G) \geq 2\nu_f(G)$.

Therefore

$$\text{perrank}(G) = s(G) = 2\nu_f(G).$$

■

A alternative proof of Theorem 3 would be to apply the Tutte–Berge theorem on fractional matching

$$2\nu_f(G) = n - \max_{S \subseteq V(G)} (i(G - S) - |S|)$$

and utilize the following result produced by Aaghabali et al.[1]:

$$\text{perrank}(G) = n - \max_{S \subseteq V(G)} (i(G - S) - |S|).$$

Direct comparison of the two formulas instantly gives the desired result.

4 Algorithm

Now we are prepared to present the algorithm computing the maximum Sachs subgraph, which provides a computational approach to finding a maximum Sachs subgraph by utilizing linear programming by solving an LP to maximize the sum of fractional edge weights and thereby yielding an optimal extreme-point solution. From the deductions above, by extracting the specific edges assigned a weight of exactly 1 or $\frac{1}{2}$, the algorithm systematically isolates the matching and cycle components required to construct the final subgraph.

The pseudo-code of the algorithm goes as follows:

Algorithm 1 MAXIMUM-SACHS

Require: Graph $G = (V, E)$

1: Solve the linear program

$$\begin{aligned} & \text{maximize} && 2 \sum_{e \in E} f(e) \\ & \text{subject to} && \sum_{e \ni v} f(e) \leq 1 \quad \text{for every } v \in V, \\ & && f_e \geq 0 \quad \text{for every } e \in E, \end{aligned}$$

and obtain an optimal extreme-point solution f^* .

2: $M \leftarrow \{e \in E : f^*(e) = 1\}$

3: $F \leftarrow \{e \in E : f^*(e) = \frac{1}{2}\}$

4: $H \leftarrow$ the subgraph of G with edge set $M \cup F$ and without isolated vertices.

5: **return** H

Now that the algorithm is provided, the greatest necessity is to prove its correctness. This can be distilled simply into:

Theorem 4. *Algorithm 1 correctly computes the maximum Sachs subgraph of the input graph G .*

Proof. Let $s(G) = \max\{|V(S)| : S \text{ is a Sachs subgraph of } G\}$, and let z^* be the optimal value of the linear program (FM). It will be shown eventually that $s(G) = z^*$.

Claim 1. *Every Sachs subgraph yields a feasible fractional matching.*

Proof. Let S be any Sachs subgraph. Define

$$f(e) = \begin{cases} 1, & e \text{ is a } K_2 \text{ component of } H, \\ \frac{1}{2}, & e \text{ belongs to a cycle component of } H, \\ 0, & e \text{ does not belong to } H. \end{cases}$$

Notice that this is the same function as the one at the end of Section 2.2, and is therefore a feasible solution to (FM). Each K_2 component contributes 2 to the

objective function, matching its vertex count. A cycle of length r contributes $2 \cdot r \cdot \frac{1}{2} = r$, also matching its vertex count. Thus, $2 \sum_{e \in E} f(e) = |V(S)|$, implying:

$$z^* \geq s(G) \quad (1)$$

■

Claim 2. *An optimal extreme-point solution yields a Sachs subgraph.*

Proof. Consider an optimal extreme-point solution f^* of (FM). By the half-integrality structure of extreme points in the fractional matching polytope, $f^*(e) \in \{0, \frac{1}{2}, 1\}$; weight-1 edges form a matching, and weight-1/2 edges form pairwise vertex-disjoint odd cycles. Hence, every non-trivial component of $H = \{e : f^*(e) > 0\}$ is a K_2 or an odd cycle, making H a Sachs subgraph. For every vertex v covered by H , $\sum_{e \ni v} f^*(e) = 1$. Therefore:

$$|V(H)| = \sum_{v \in V} \sum_{e \ni v} f^*(e) = 2 \sum_{e \in E} f^*(e) = z^*$$

Thus:

$$s(G) \geq |V(H)| = z^* \quad (2)$$

■

Combining (1) and (2) yields $s(G) = z^*$. Therefore, the output H is a Sachs subgraph covering the maximum number of vertices. ■

4.1 Time Complexity

We give a fully combinatorial realization of Step 1. Construct the bipartite double cover $B = (V_L, V_R, E_B)$, with a left and right copy of each vertex. For each $\{u, v\} \in E$, put both $u_L v_R$ and $v_L u_R$ in E_B . Thus B has $2|V|$ vertices and $2|E|$ edges.

Algorithm 2 Bipartite Double-Cover Extraction

Require: Graph $B = (V_L, V_R, E_B)$

- 1: Find a maximum matching Q in B using augmenting paths.
- 2: For every $e = \{u, v\} \in E$, set

$$f(e) = \frac{\mathbf{1}_{u_L v_R \in Q} + \mathbf{1}_{v_L u_R \in Q}}{2}.$$

- 3: The edges of weight $\frac{1}{2}$ form disjoint paths and cycles. On each path, select alternating edges at weight 1; on an even cycle, do the same; keep every odd cycle at weight $\frac{1}{2}$. Preserve edges already at weight 1.
 - 4: Return the resulting fractional matching f^* .
-

The path in Step 3 always has an even number of edges; this follows from the optimality argument below.

Theorem 5. *Algorithm 2 returns an optimal extreme fractional matching. The accompanying C++ implementation runs in $O(|V||E| + |V|)$ time and $O(|V| + |E|)$ space.*

Proof. Given any feasible fractional matching g in G , give the two directed copies of each edge e in B value $g(e)$. Every vertex in B has fractional matching degree at most 1, and the total value is $2 \sum_e g(e)$. The bipartite matching polytope is integral; hence a maximum matching Q of B has $|Q| \geq 2\nu_f(G)$.

Conversely, given Q , Step 2 defines a feasible fractional matching f : the incident sum at v is half the number of matching edges at v_L and v_R , hence at most 1. Also, $2 \sum_e f(e) = |Q|$, implying $|Q| \leq 2\nu_f(G)$. We conclude

$$|Q| = 2\nu_f(G).$$

For a weight- $\frac{1}{2}$ edge exactly one directed copy belongs to Q . Each original vertex has at most one selected copy entering and one leaving, so its degree in the weight- $\frac{1}{2}$ support is at most 2. A weight-1 edge uses both copies and is disjoint from the other positive edges. Therefore the half-weight support consists of disjoint paths and cycles.

If a half-weight path had $2k + 1$ edges, setting its alternating $k + 1$ edges to weight 1 and the rest to zero would be feasible and would increase $\sum_e f(e)$ by $\frac{1}{2}$. This contradicts optimality. Thus every such path has $2k$ edges. Selecting its k alternating edges preserves $\sum_e f(e)$. An even cycle has the same property; an odd cycle remains at half weight. The final support consists of disjoint K_2 's and odd cycles, and its objective stays $|Q|$. This support also determines an extreme point: tight degree equalities uniquely determine the odd-cycle weights, while the matching edges are fixed at 1.

For the bound, each augmenting-path search visits each directed edge of B at most once. There are at most $|V|$ searches, giving $O(|V||E| + |V|)$ time. Reading and sorting the simple-graph edges to reject duplicates adds $O(|V| + |E| + |E| \log |E|)$, absorbed into that bound because $|E| \leq |V|(|V| - 1)/2$. Component extraction is $O(|V| + |E|)$. The graph, matching arrays and work arrays use $O(|V| + |E|)$ space. ■

The stated running time applies to the augmenting-path implementation supplied in Section 4.2. Through the established bipartite matching reduction, a Hopcroft–Karp implementation gives the faster bound $(O(|E|\sqrt{|V|} + |V|))$, including initialization. Our analysis concerns the explicit program presented here, whose matching operations, integer weight representation, and component-conversion steps correspond directly to the construction above.

4.2 C++ Implementation

The implementation uses only the C++17 standard library. Its input variables `nv` and `ne` are $|V|$ and $|E|$, as in the pseudo-code; `edges[i]` is e_i . The integer arrays `f2[i]` and `f_star2[i]` store $2f(e_i)$ and $2f^*(e_i)$, so no rounding is needed; `z_star` is z^* . Input is `nv, ne`, followed by `ne` undirected edges with zero-based endpoints; the graph must be simple. For each selected edge it prints its two original endpoints and exact weight 1 or 0.5. These are the edges of M and F , respectively, and their union is the returned subgraph H .

An empty output is valid. Its internal breadth-first search finds augmenting

paths; the following path/cycle pass performs the exact conversion in Algorithm 2.

```
1 #include <algorithm>
2 #include <iostream>
3 #include <queue>
4 #include <utility>
5 #include <vector>
6 using namespace std;
7 int main(){
8     int nv, ne;
9     if (!(cin >> nv >> ne)){
10         return 0;
11     }
12     if (nv < 0 || ne < 0){
13         return 1;
14     }
15     vector<pair<int, int> > edges(ne);
16     vector<vector<pair<int, int> > > adj(nv);
17     for (int i = 0; i < ne; ++i){
18         auto &[u, v] = edges[i];
19         if (!(cin >> u >> v) || u < 0 || v < 0 ||
20             u >= nv || v >= nv || u == v){
21             return 1;
22         }
23         adj[u].push_back({v, i});
24         adj[v].push_back({u, i});
25     }
26     auto sorted = edges;
27     for (auto &[u, v] : sorted) if (u > v) swap(u, v);
28     sort(sorted.begin(), sorted.end());
```

```

29     if (unique(sorted.begin(), sorted.end()) != sorted
30         .end()){
31         return 1;
32     }
33     //left u, right v
34     vector<int> ml(nv, -1), mr(nv, -1), match_edge(nv,
35         -1);
36     vector<int> seen_l(nv), seen_r(nv), par(nv),
37         par_edge(nv);
38     int z_star = 0;
39     for (int root = 0; root < nv; ++root){
40         if (adj[root].empty()){
41             continue;
42         }
43         int stamp = root + 1, free_r = -1;
44         queue<int> q;
45         q.push(root);
46         seen_l[root] = stamp;
47         while (!q.empty() && free_r == -1) {
48             int u = q.front(); q.pop();
49             for (auto [v, id] : adj[u]) {
50                 if (seen_r[v] == stamp) continue;
51                 seen_r[v] = stamp;
52                 par[v] = u;
53                 par_edge[v] = id;
54                 if (mr[v] == -1) {
55                     free_r = v;
56                     break;
57                 }
58             }
59             int next = mr[v];

```

```

56         if (seen_l[next] != stamp) {
57             seen_l[next] = stamp;
58             q.push(next);
59         }
60     }
61 }
62 if (free_r == -1) continue;
63 ++z_star;
64 for (int v = free_r; v != -1; ) {
65     int u = par[v], old = ml[u];
66     ml[u] = v;
67     mr[v] = u;
68     match_edge[v] = par_edge[v];
69     v = old;
70 }
71 }
72
73 // f2[i] = 2 * f(e_i); f_star2[i] = 2 * f*(e_i),
74
75 vector<int> f2(ne), f_star2(ne), used(ne);
76 vector<vector<int>> half(nv);
77 for (int v = 0; v < nv; ++v)
78     if (mr[v] != -1) ++f2[match_edge[v]];
79 for (int i = 0; i < ne; ++i) {
80     if (f2[i] == 2) f_star2[i] = 2;
81     if (f2[i] == 1) {
82         auto [u, v] = edges[i];
83         half[u].push_back(i);
84         half[v].push_back(i);
85     }
86 }

```

```

85     }
86     for (int v = 0; v < nv; ++v)
87         if (half[v].size() > 2) return 1;
88
89     auto trace = [&](int start) {
90         vector<int> path;
91         int u = start;
92         while (true) {
93             int id = -1;
94             for (int j : half[u]) if (!used[j]) {
95                 id = j;
96                 break;
97             }
98             if (id == -1) break;
99             used[id] = 1;
100             path.push_back(id);
101             auto [a, b] = edges[id];
102             u = a ^ b ^ u;
103         }
104         return pair<vector<int>, int>{path, u};
105     };
106
107     // A half-weight path has even length at the
108     optimum
109     for (int v = 0; v < nv; ++v) {
110         if (half[v].size() != 1 || used[half[v][0]])
111             continue;
112         auto [path, end] = trace(v);
113         if (path.size() % 2 || half[end].size() != 1)
114             return 1;

```

```

112         for (int j = 0; j < (int)path.size(); j += 2)
113             f_star2[path[j]] = 2;
114     }
115     // Every remaining component is a cycle
116     for (int i = 0; i < ne; ++i) if (f2[i] == 1 && !
used[i]) {
117         auto [cycle, end] = trace(edges[i].first);
118         if (end != edges[i].first) return 1;
119         if (cycle.size() % 2) {
120             for (int id : cycle) f_star2[id] = 1;
121         } else {
122             for (int j = 0; j < (int)cycle.size(); j
+= 2)
123                 f_star2[cycle[j]] = 2;
124         }
125     }
126
127     vector<int> degree(nv);
128     for (int i = 0; i < ne; ++i) {
129         degree[edges[i].first] += f_star2[i];
130         degree[edges[i].second] += f_star2[i];
131     }
132     int covered = 0;
133     for (int d : degree) {
134         if (d != 0 && d != 2) return 1;
135         covered += (d == 2);
136     }
137     if (covered != z_star) return 1;
138     // The output is H: M has f*(e)=1, F has f*(e)
=1/2

```

```

139     for (int i = 0; i < ne; ++i) if (f_star2[i])
140         cout << edges[i].first << '␣' << edges[i].
            second << '␣'
141         << (f_star2[i] == 2 ? "1" : "0.5") << '\n'
            ';
142     }

```

5 Potentials for Further Research

Future research on this topic may continue to build on Balinski's result [2], with the aim of developing results for Sachs subgraphs that parallel fundamental concepts in matching theory. Notable examples include min-max formulas for perrank, the Gallai–Edmonds decomposition (a partition of the vertex set into three subsets that reveals the structure of maximum matchings), and barrier structures (vertex sets determined by the Berge formula). We hope to obtain analogous structural insights for Sachs subgraphs.

In addition to the applications discussed above, further research could impose additional structural constraints on Sachs factors, such as limiting the number of K_2 (matching edges) and cycles, and determine conditions under which Sachs factors satisfying these restrictions exist.

References

- [1] Mehdi Aghabali, Saieed Akbari, and Z. Tajfirooz. Order of the largest sachs subgraphs in graphs. *Linear and Multilinear Algebra*, 65:204–209, 05 2016.
- [2] Michel L. Balinski. Integer programming: Methods, uses, computations. *Management Science, INFORMS*, vol. 12 (3), pages 253–313, 1965.

- [3] Daniel Hathcock and R. Ravi. Perfect fractional matchings in bipartite graphs via proportional allocations, 2026.
- [4] Daniel A. Jaume, Diego G Martinez, Cristian Panelo, and Kevin Pereyra. k -sachs-critical graphs. *Procedia Computer Science*, 273:333–340, 2025. XIII Latin American Algorithms, Graphs, and Optimization Symposium (LAGOS 2025).
- [5] László Lovász. Subgraphs with prescribed valencies. *J. Comb. Theory*, 8:391–416, 1970.
- [6] Haicheng Ma and Xianghong He. Singularity conditions and probabilities of tricyclic graphs. *Axioms*, 14(12), 2025.
- [7] Julius Petersen. Die theorie der regulären graphs. *Acta Mathematica*, 15:193–220, 1891.
- [8] Jiawei Qian, Frans Schalekamp, David P. Williamson, and Anke van Zuylen. On the integrality gap of the subtour lp for the 1,2-tsp, 2014.
- [9] Horst Sachs. Beziehungen zwischen den in einem graphen enthaltenen kreisen und seinem charakteristischen polynom. *Publicationes Mathematicae Debrecen*, 11:119–134, 1964.
- [10] William T. Tutte. The factorization of linear graphs. *Journal of The London Mathematical Society-second Series*, pages 107–111, 1947.
- [11] William T. Tutte. The factorization of locally finite graphs. *Canadian Journal of Mathematics*, 2:44 – 49, 1950.

致谢（中文）

本文的研究对象Sachs子图与图论中的 $\{1,2\}$ -因子在定义上存在紧密的联系，又在化学上可以用于确定分子的结构与热稳定性，故引起了参赛同学和桂鹏老师的兴趣；研究最终定位为补充性研究，课题为对最大Sachs子图的存在性条件及边数进行探索并在此过程中尝试与其他领域（最终敲定为线性规划/分数匹配）建立联系。桂鹏老师参与了研究的选题、背景材料收集、整体语言的组织及部分铺垫内容的论文写作；参赛的张璟霖、颜梓晨同学在桂鹏老师的指导下完成了对Sachs子图结构的转化、算法设计及时间复杂度规划等一系列工作，并撰写了文中主要结果的理论证明及算法的伪代码和C++实现。指导老师的工作均是无偿的。

自2026年3月起，张璟霖、颜梓晨同学定期（每周一次）与桂鹏老师进行沟通学习并参与团队的科研工作。最初几次沟通由桂鹏老师就图论的基础知识与本课题的研究背景及现状对参赛同学进行指导；在参赛同学掌握基础知识、了解研究背景以后，他们通过将Sachs子图的结构转化为半整数性权值及分数匹配极点问题，发现可将最大Sachs子图问题归结为分数匹配问题，并可利用二部图覆盖处理。基于上述发现，他们设计和实现了一个求解最大Sachs子图的高效算法，并建立了Sachs子图与更为广泛研究的线性规划领域的联系。在这个过程中，桂鹏老师对于项目工作中的关键思想和困难的地方（如将偶圈从最大Sachs子图中消去、命题的陈述方式、主要理论成果的夹逼法思路等）给予了指导；大部分的证明和算法实现由参赛同学完成。