Deterministic Graph Sparsification of Dynamic fractional matching

Presented By:

Hanumat Lal Vishwakarma

2021CSM1019

Guided By:

Dr. Anil Shukla

Assistant Profesor, IIT Ropar

Content

- **Background**
- **A Common Strategy followed in Recent Literatures**
- **Dynamic Fractional Matching**
- **Problem Statement**
- **First Deterministic Algorithm**
- **Demonstrate with an example**
- **Our Proposed Algorithm**
- **Analysis**
- **Conclusions**
- **References**

Background

- **Dynamic Graph Algorithms:-** "Preserving graph properties on dynamic graph"

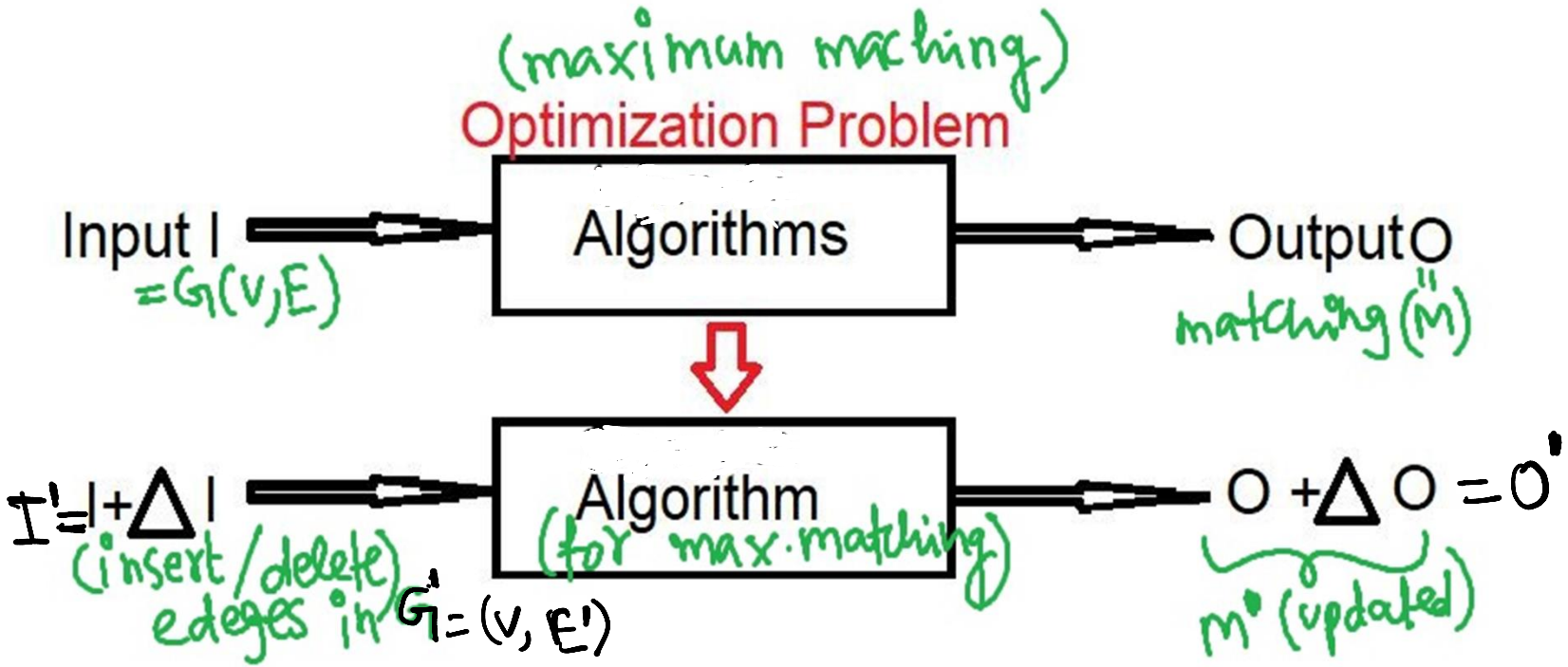


Fig1. Preserving Maximum Matching under edge updation

- **First thought :**

  -Re-Apply algorithm from scratch on updated graph

- **Dynamic Matching Problem under edge updation :-**

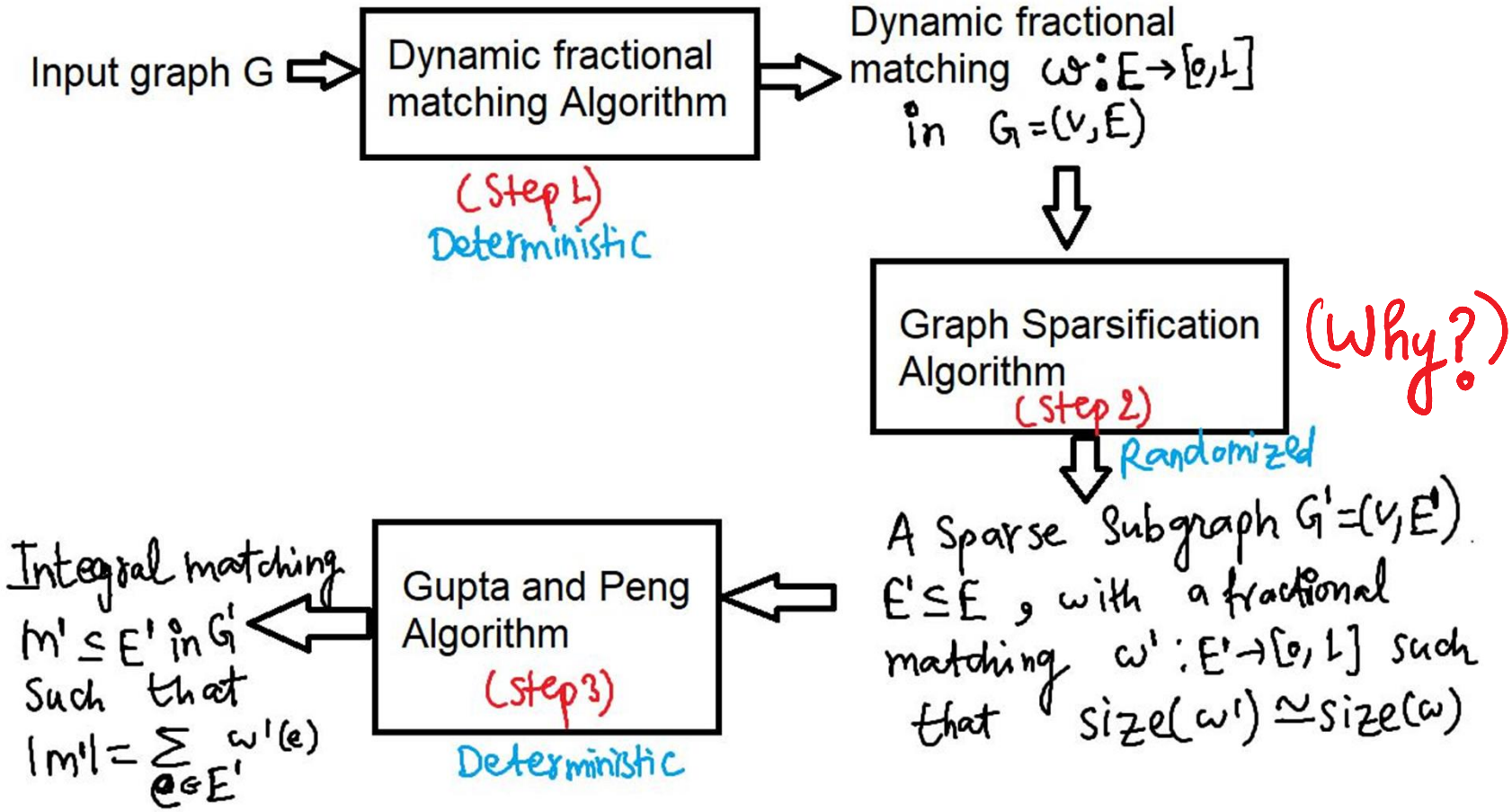
  Want: Maintain graph property “Maximum matching”

  Challenge: Graph changes over time as edges are inserted or deleted.

  Goal: Design an updation algorithm of the least time complexity that update the maximum matching as edges are inserted or deleted on run-time.

  *update time= time taken to handle an update(edge insert/delete)

A Common Strategy followed in Recent literatures

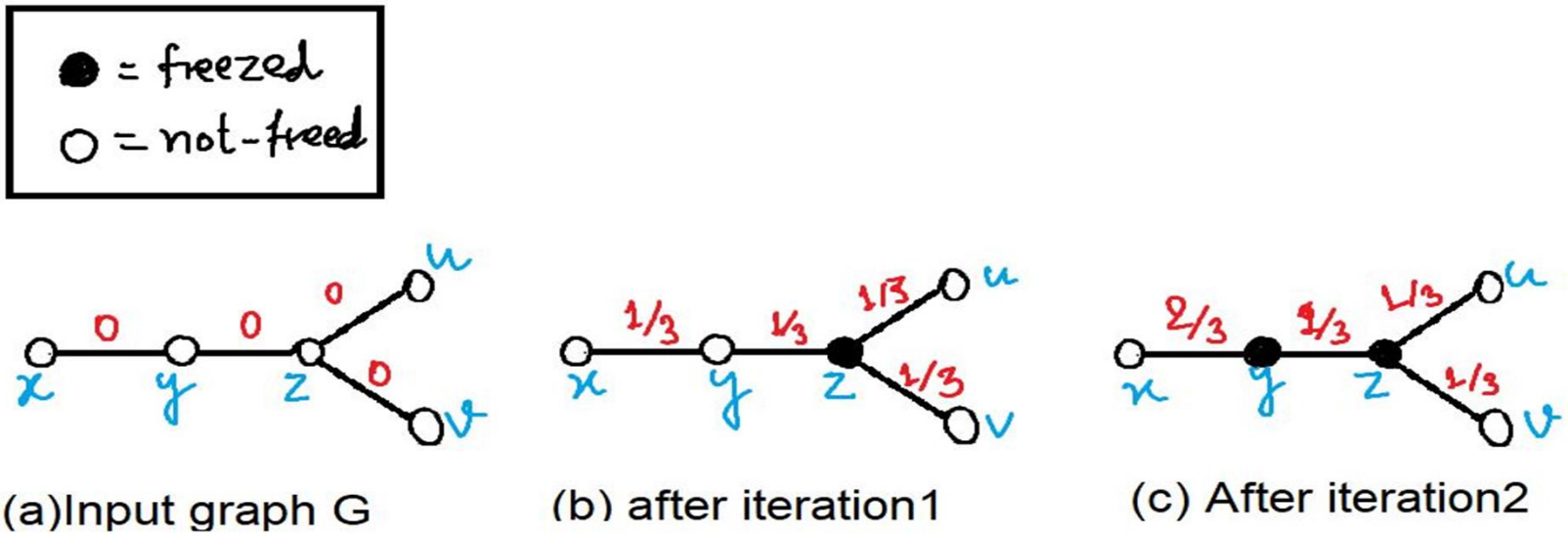


Dynamic fractional matching

Static Algorithm:-

w(e)=0 for all the edges e E
REPEAT,

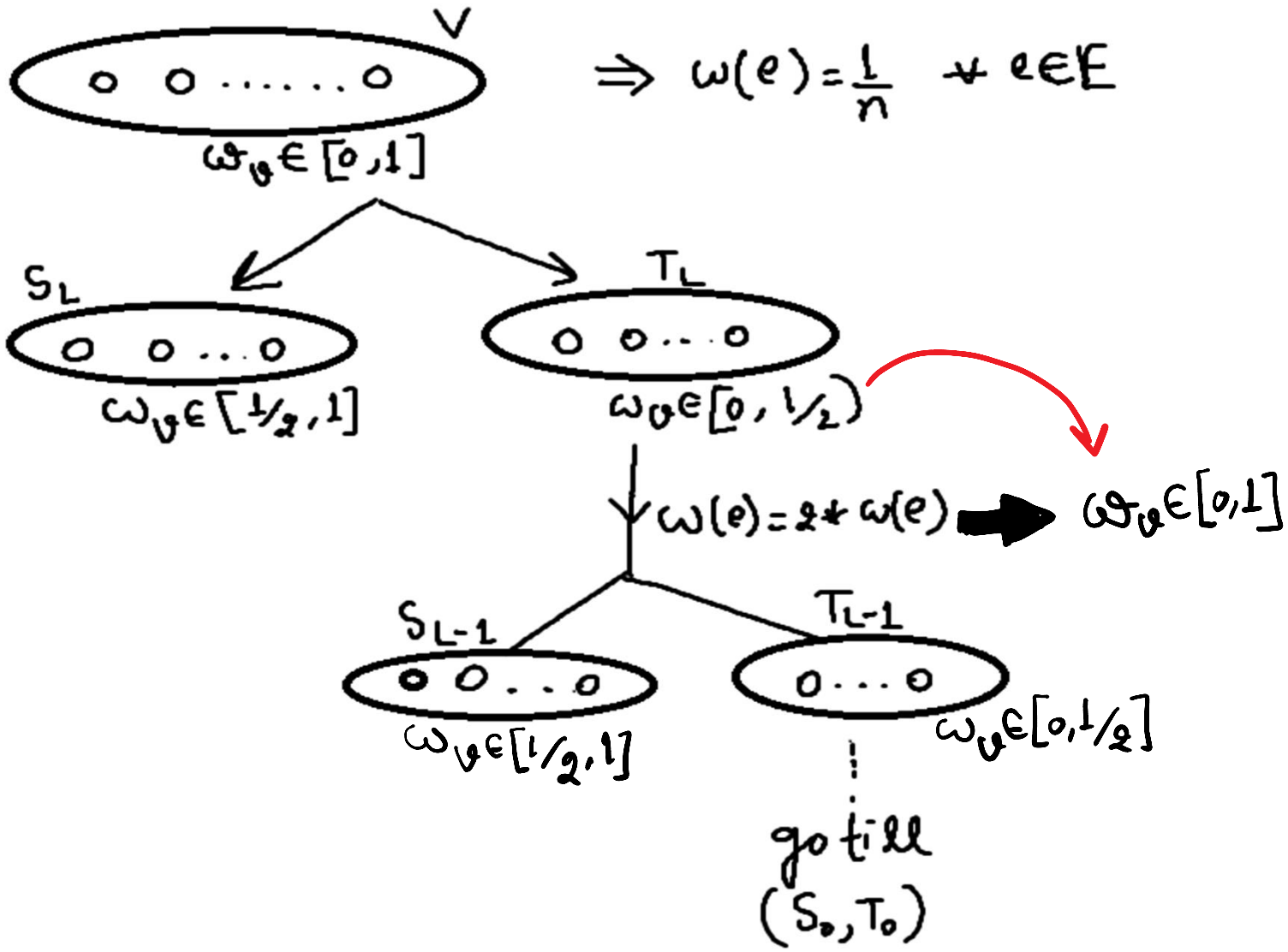
- Increase the edge weights continuously at same rate till node get tight.
- Freeze the edge incident on the tight node.



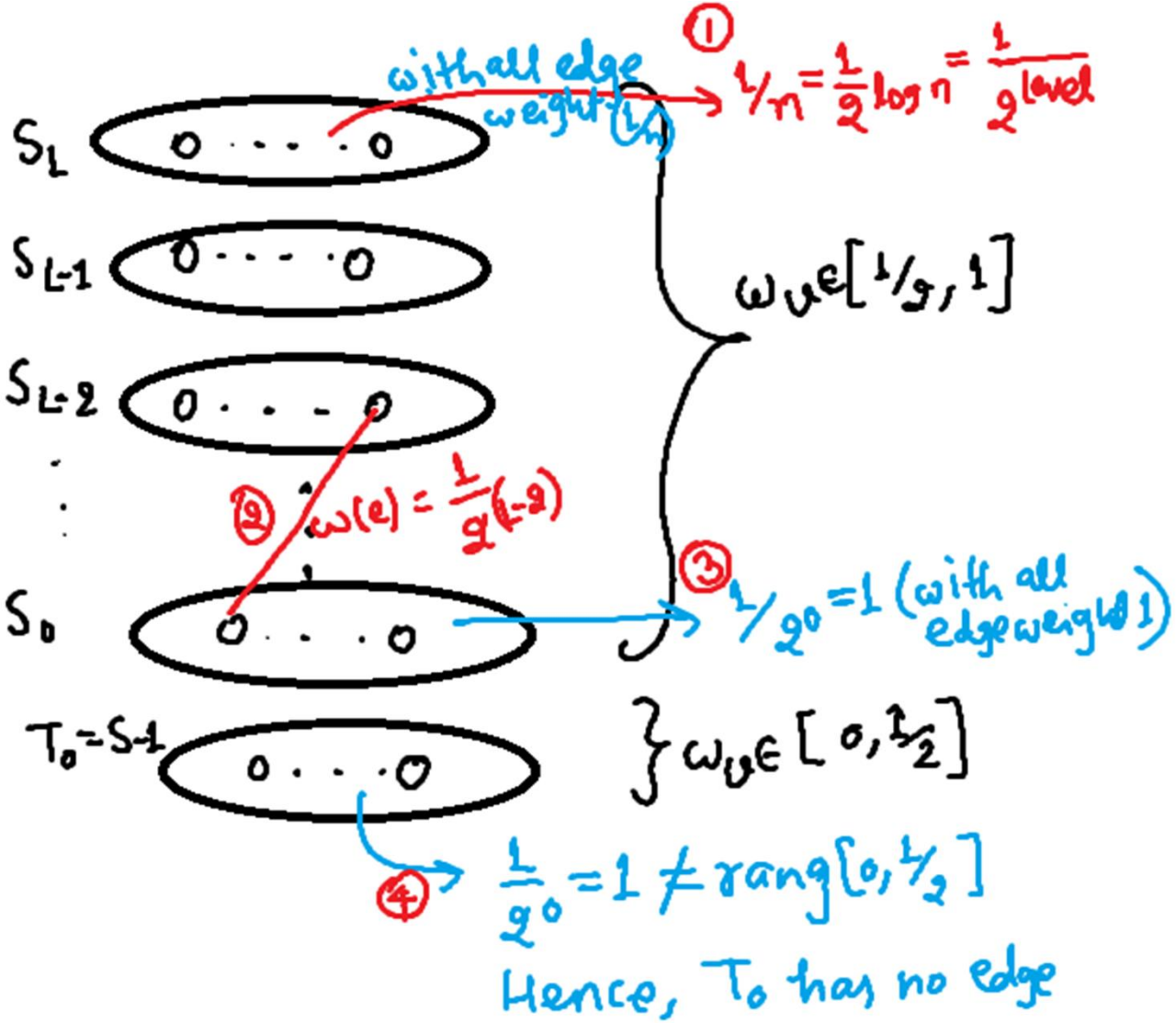
(a)Input graph G (b) after iteration1 (c) After iteration2

Fig2. demonstration in static setting

- **Descretize the algorithm:-**(increase edge weight by factor of 2)



- **Maintaining Heirarchical partitioning of G:-**



- **Dynamic Setting:-**

On Edge Deletion:-
Edge weights takes away for both end.
On Edge Insertion:- (between edge i and j)
Edge weights increases by 1/2^i on both end-point.

Fix-dirty-nodes(Algorithm):
while(dirty node x):
    if w(x) is too large, l(x)=l(x)+1;
    if w(x) is too small, l(x)=l(x)-1;

Problem Statement

Input: λ-uniform weighted graph G=(V,E), with fractional matching w: E->[0,1].
Output: 2λ-uniform weighted graph G'=(V,E'), E' is subset of with fractional matching w': E'->[0,1] such that size(w')=size(w), where size(w) is defined as sum of all the edge-weights in graph G.

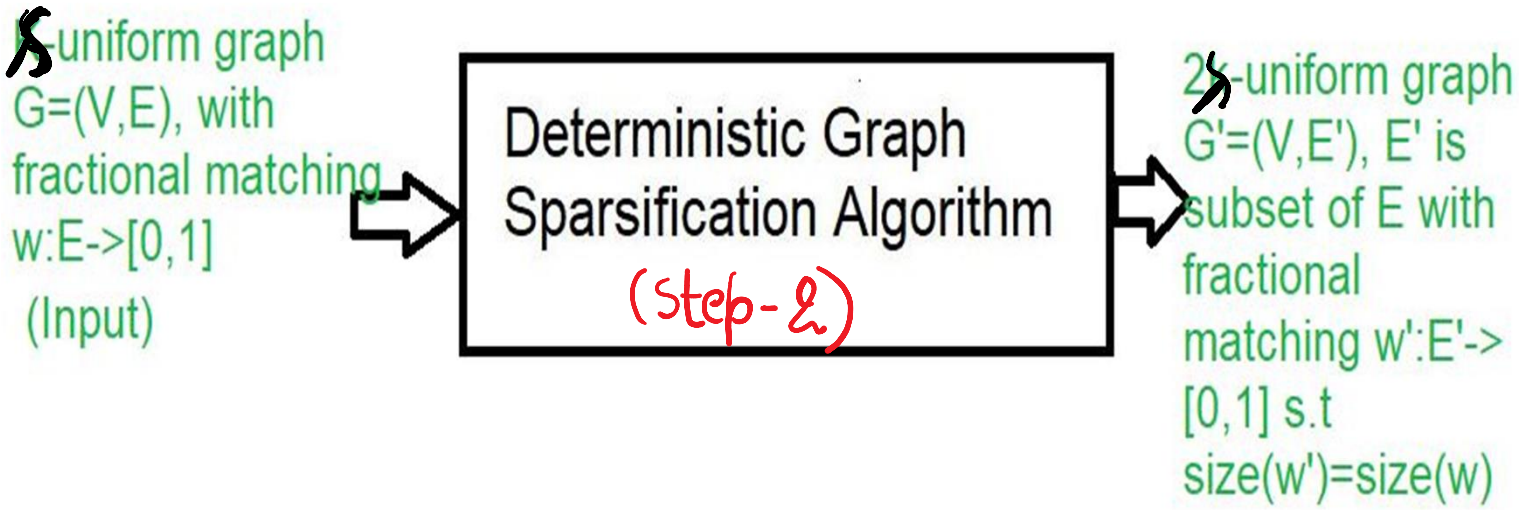
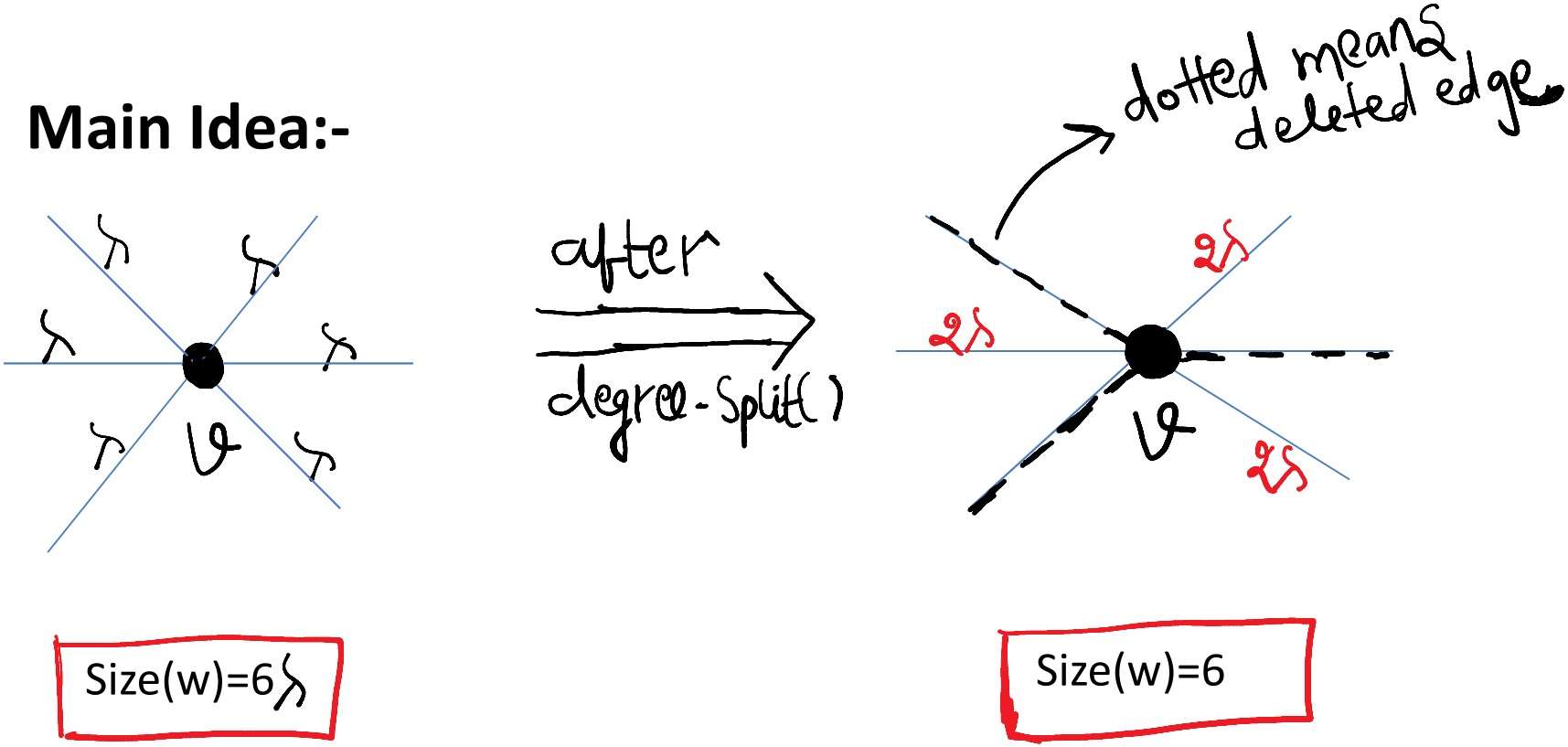


Fig3. Our Problem statement(step-2 of common strategy)

First Deterministic Solution

- **Degree_Split() Subroutine:-**

Main Idea:-



Degree-split Algorithm:-

For, Input G=(V,E)

Step 1. Partition E into a set of disjoint maximal walk {W1,W2,…,Wk}

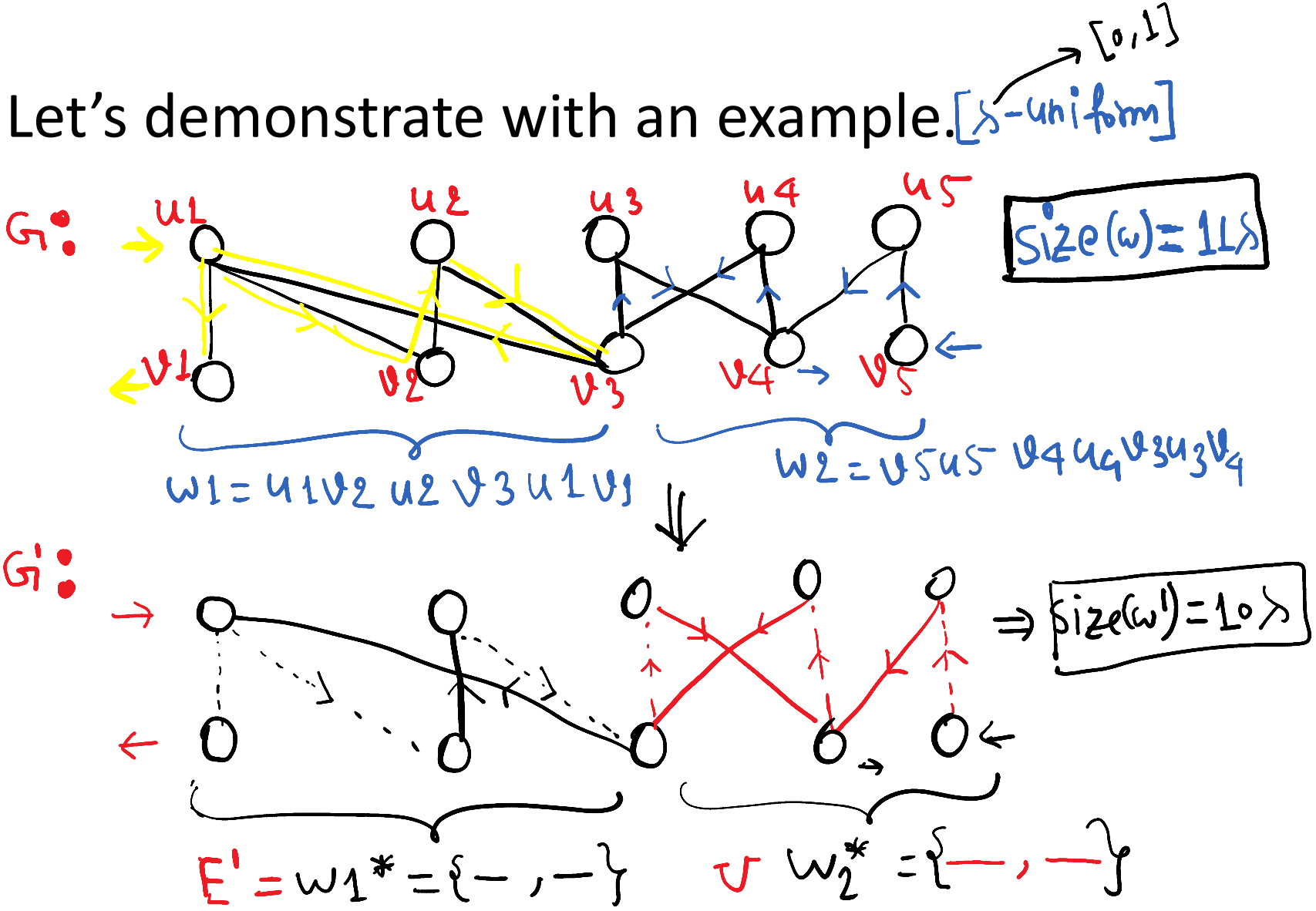
Step 2. for each i: [1,k],

Let Wi*=Collection of alternate edges from Wi.

Step 3. return E’=W1* U W2* …Wk*

Demonstrate with an Example

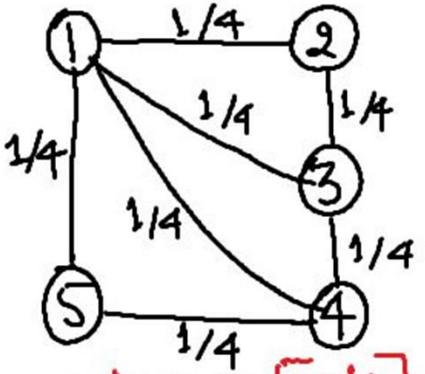
- Let's demonstrate with an example.



Our Proposed algorithm

1. For each vertex v in V, create a list L-v of its neighbors in non-increasing order of their degree.
2. Initialize E' as an empty set of edges
3. For each vertex v in V:
a. Compute k=ceil(degree(v)/2), where degree(v) is the degree of the vertex in G.
b. If v has already been relaxed, proceed to the next vertex (i.e., degree(v) is already sparsified).
c. Initialize S as an empty set of vertices containing the selected neighbors of v.
d. For each neighbor u in L-v:
i. If u<v and the edge u,v has not been added to E' yet, then continue to the next neighbor.
ii. If u<v and the edge u,v is already present in E', then reduce k by 1 and continue to the next neighbor.
iii. If u>=v and |S| <= k, add u to S.
iv. If |S| = k, break the loop.
e. Add the edge v,u for each u in S to the edge set E' of G'.
4. Return the new graph G' with vertex set V and edge set E', and double the weight of each edge in the graph G'.

ex①



size(ω) = ⌈7 * 1/4⌉ = 2

(1/4 - uniform graph)

Adjacency matrix:

1 : [2, 3, 4, 5]
2 : [1, 3]
3 : [1, 2, 4]
4 : [1, 3, 5]
5 : [1, 4]

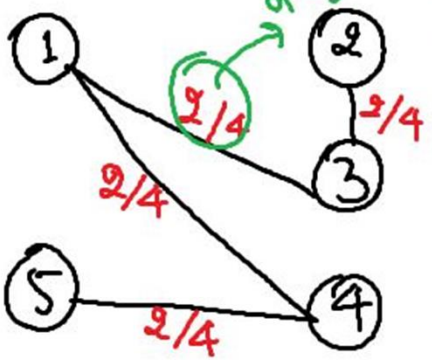
(after sorting neighbours as per degree)

1 : [3, 4, 2, 5]
2 : [1, 3]
3 : [1, 4, 2]
4 : [1, 3, 5]
5 : [1, 4]

after Apply Algo.

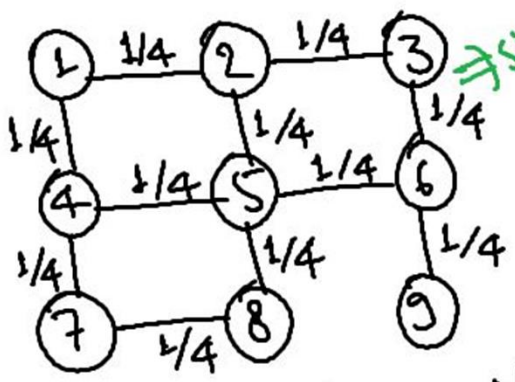
degree doubled after sparsification

size(ω) = ⌈4 * 2/4⌉ = 2 = size(ω)



(2/4 - uniform graph)

ex (2)



$\Rightarrow size(w) = \lceil 11/4 \rceil = 3$

G: 1/4-uniform graph

1: [2, 4] → neighbours are sorted as per degree

2: [5, 1, 3]

3: [2, 6]

4: [5, 1, 7]

5: [2, 4, 6, 8]

6: [5, 3, 9]

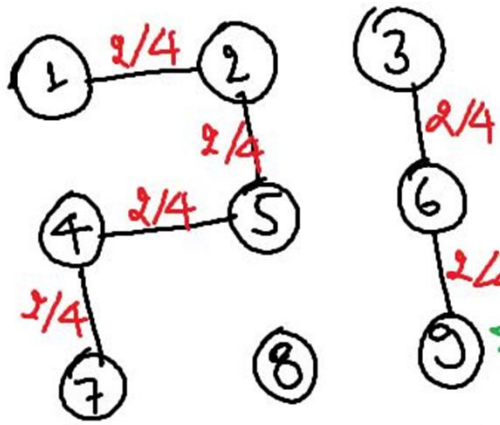
7: [4, 8]

8: [5, 7]

9: [6]

after algo.

final sparsified graph.



$\Rightarrow size(w') = \lceil 12/4 \rceil = 3$

G': 2/4-uniform graph

Hence, $size(w') = size(w)$

Online C++ Compiler

programiz.com/cpp-programming/online-compiler/

Programiz
C++ Online Compiler

Get 10 Free Images From Adobe Stock. Start Now.

ADS VIA CARBON

Interactive C++ Course

main.cpp

Run

```
    // add edges {v, u} to G' and double the weight in G
    for (auto u : S)
    {
        //Gprime[v].push_back({u, 2.0 / G[v].size()});
        //Gprime[u].push_back({v, 2.0 / G[u].size()});
        Gprime[v].push_back({u, 2.0 * w[{v, u}]});
        Gprime[u].push_back({v, 2.0 * w[{u,v}]});
        wprime += 2.0 * w[{v, u}];
    }
}

//print output graph G' and its weight w'
cout << "sparsified graph G' = {";
for (int v = 0; v < Gprime.size(); v++)
{
    cout << v << ":{";
    for (auto [u, w] : Gprime[v])
    {
        cout << u << "," << w << ",";
    }
    cout << "},";
}
cout << "}" << endl;
cout<<"size of fractional matching in sparsified graph G', size(w')= "
    <<wprime<<endl;
    return 0;
}
```

Output

Clear

```
0 1 0.25
0 2 0.25
0 3 0.25
0 4 0.25
1 2 0.25
2 3 0.25
3 4 0.25
5 7

0 1 0.25

0 2 0.25

0 3 0.25

0 4 0.25

1 2 0.25

2 3 0.25

3 4 0.25
size of fractional matching in original graph 'G', size(w)= 1.75
sparsifiied graph G' = {0:{2,0.5,3,0.5,},1:{2,0.5,},2:{0,0.5,1,0.5,},3:{0,0.5,4,0.5
    ,},4:{3,0.5,},}
size of fractional matching in sparsified graph G', size(w')= 2
```

Type here to search

Earnings upcoming

ENG IN 2:20 AM 5/11/2023

Online C++ Compiler

programiz.com/cpp-programming/online-compiler/

Programiz
C++ Online Compiler

Interactive C++ Course

main.cpp

Run

Output

Clear

```
76      // add edges {v, u} to G' and double the weight in G
77      for (auto u : S)
78      {
79          //Gprime[v].push_back({u, 2.0 / G[v].size()});
80          //Gprime[u].push_back({v, 2.0 / G[u].size()});
81          Gprime[v].push_back({u, 2.0 * w[{v, u}]});
82          Gprime[u].push_back({v, 2.0 * w[{u,v}]});
83          wprime += 2.0 * w[{v, u}];
84      }
85  }
86
87  //print output graph G' and its weight w'
88  cout << "sparsifified graph G' = {";
89  for (int v = 0; v < Gprime.size(); v++)
90  {
91      cout << v << ":{";
92      for (auto [u, w] : Gprime[v])
93      {
94          cout << u << "," << w << ",";
95      }
96      cout << "},";
97  }
98  cout << "}" << endl;
99  cout<<"size of fractional matching in sparsified graph G', size(w')= "
        <<wprime<<endl;
100     return 0;
101 }
```

```
0 1 0.25

0 3 0.25

1 2 0.25

1 4 0.25

2 5 0.25

3 4 0.25

3 6 0.25

4 5 0.25

4 7 0.25

5 8 0.25

6 7 0.25
size of fractional matching in original graph 'G', size(w)= 2.75
sparsifified graph G' = {0:{1,0.5,},1:{0,0.5,4,0.5,2,0.5,},2:{1,0.5,},3:{4,0.5,6,0.5
    ,},4:{1,0.5,3,0.5,},5:{8,0.5,},6:{3,0.5,},7:{},8:{5,0.5,},}
size of fractional matching in sparsified graph G', size(w')= 3
```

Type here to search

23°C Haze

ENG IN 2:21 AM 5/11/2023

Analysis

- Time Complexity: O(E log V)
- Space Complexity: O(E)

Implications:-

(1) Derandomize existing solutions.

(2) Create new deterministic solutions for Dynamic matching Problem

Conclusions

- Prior Works, **Arar et al. [ICALP'18]** and **David Wajc [STOC'20]**.
- No deterministic solutions before 2021.
- Degree-split was first deterministic solution.
- Our proposed algorithm is second deterministic solution.
- In future, Possibility of more such deterministic solutions and optimization might be possible.

References

- Bhattacharya, S., Kiss, P. (2021, May 4). Deterministic Rounding of Dynamic Fractional Matchings. Paper presented at the International Colloquium on Automata, Languages, and Programming (ICALP).
- Bhattacharya, S., Kiss, P., Saranurak, T., Wajc, D. (2022, November 9). Dynamic Matching with Better-than-2 Approximation in Polylogarithmic Update Time.
- Wajc, D. (2020). Rounding Techniques for Dynamic Fractional Matchings. In Proceedings of the 52nd Annual Symposium on Theory of Computing (STOC).
- Arar, M. A., Mastrolilli, M., &Liaghat, V. (2018). General-Purpose Rounding Schemes for Dynamic Fractional Matchings. In Proceedings of the 45th International Colloquium on Automata, Languages, and Programming (ICALP).
- Manoj Gupta & Richard Peng(April 11, 2013),Fully Dynamic (1 + ϱ)-Approximate Matchings .
- Vazirani, V. V. (2022). Online Bipartite Matching and Adwords.
- Karp, R. M., Vazirani, U. V., Vazirani, V. V. (1990). An optimal algorithm for online bipartite matching. In Proceedings of the Symposium on the Theory of Computing (STOC) (pp. 352-358). ACM.

Thank You