

# Multiple Source Shortest Path with unit weights

## 1 Introduction

**Given:** Let  $G$  be a directed graph  $(V, \vec{E})$ , embedded on a surface with genus  $g$ . All edge weights are unit.

**Find:** Consider boundary  $f$  of  $G$ .  $\forall v \in f$ , find a shortest path to  $\forall u \in V$ .

Let  $T$  be the BFS (Breadth first search) tree of  $G$ , and  $C$  be the BFS co-tree in  $G$ . Then there is exactly  $2g$  leftover edges  $L = e_1, e_2, \dots, e_{2g}$ .

There exists a unique cycle  $\lambda_i$  in  $C \cup e_i$ , and  $(\lambda_1, \lambda_2, \dots, \lambda_{2g}) = \Lambda$  defines homology basis. We define homological signature of an edge as follows:

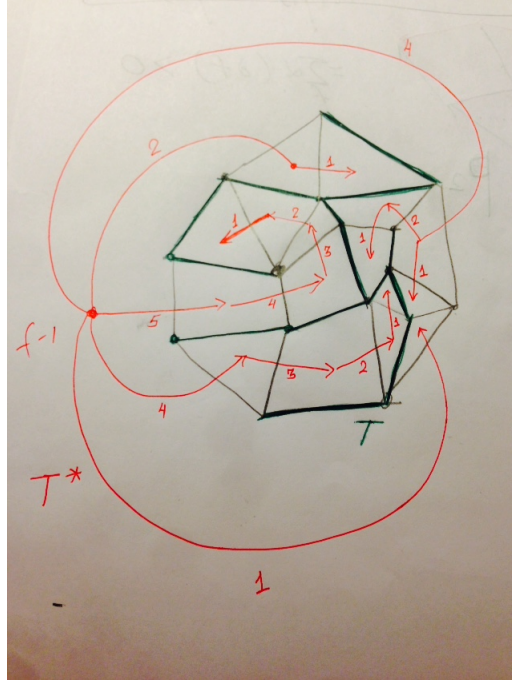
$$[e]_i = \begin{cases} 1 & , \text{if } e \in \lambda_i \\ -1 & , \text{if } rev(r) \in \lambda_i \\ 0 & , \text{otherwise} \end{cases}$$

Furthermore, we define  $\alpha$  recursively as follows:

$$\alpha(\vec{e}^*) = \begin{cases} 1 & , \text{if } e^* \text{ is a leaf edge of } C \\ \sum_{\text{tail}(\vec{e}') = \text{head}(\vec{e}^*)} \alpha(\vec{e}') & , \text{otherwise} \end{cases}$$

We can extend above definition with  $\alpha(\vec{e}) = \alpha(\vec{e}^*)$  and  $\alpha(e)^* = -\alpha(rev(\vec{e}^*))$ .

Let  $\tilde{w}(\vec{e}) = \langle 1, [\vec{e}], \alpha(\vec{e}) \rangle$  be new weight vector for each edge in  $G$ .



**Def:** An edge  $\vec{e}$  is "holier" than  $\vec{e}'$ , if  $\tilde{w}(\vec{e}) < \tilde{w}(\vec{e}')$  in lexicographic comparison. Therefore, we can define "holiness" of any  $S \subset G$  as follows:

$$Ho(S) = \sum_{\vec{e} \in S} \tilde{w}(\vec{e})$$

Holiest tree is a spanning tree with minimal "holiness". We build Holiest tree rooted at  $r$ , using slight tweak in the Bellman-Ford algorithm for finding shortest path tree rooted at  $r$ .

**BuildHoliestTree** $(G, \tilde{w}, r)$ :

```

Set  $d[r] \leftarrow \langle 0, [0], 0 \rangle$ 
 $\text{pred}(r) \leftarrow \text{NULL}$ 
for all  $v : v \neq r$ 
     $d[v] \leftarrow \langle \infty, [\infty], \infty \rangle$ 
     $\text{pred}(v) \leftarrow \text{NULL}$ 
put  $r$  into queue
while queue is not Empty:
    Let  $s \leftarrow$  dequeue item
    for all  $u : s \rightarrow u$ 
        if  $u$  is not marked
            mark  $u$  and put in the queue
        if  $\text{isTense}(s \rightarrow u)$ 
             $\text{relax}(s \rightarrow u)$ 

```

**isTense** $(s \rightarrow u)$ :

return  $d[s] + \tilde{w}(s \rightarrow u) < d[u]$

**relax** $(s \rightarrow u)$ :

$d[u] \leftarrow d[s] + \tilde{w}(s \rightarrow u)$   
 $\text{pred}[u] \leftarrow s$

**Observation:** Each edge will be added once to the queue.

**Corollary:** Each edge will be relaxed at most once.

**Lemma-1:** If there is no tense edge in  $G$ , then for each  $v : r \rightarrow \dots \rightarrow \text{pred}(\text{pred}(v)) \rightarrow \text{pred}(v) \rightarrow v$  is the holiest path from  $r$  to  $v$ .

**Proof:** Let's prove it by induction on  $d[v]$  distance from the root  $r$ .

Base:  $d[v] = 0$ , then  $v = r$ , so the claim is definitely true.

Induction Step: Suppose the claim is true for all vertex  $v \in V$  such that  $d[v] < d$  for some  $d$ . Consider vertex  $v$  such that  $d[v] = d$ . By induction hypothesis, all vertices with  $d[u] = d - 1$  have "holiest" path correctly updated. By definition,  $d[v] = \min_{s \rightarrow v} d[s] + \tilde{w}(s \rightarrow v)$ , here  $d[s] = d - 1$ . By Induction hypothesis,  $d[s]$  is "holiest" path, so if there is no tense edge in  $G$  then  $d[v] = \min_{s \rightarrow v} d[s] + \tilde{w}(s \rightarrow v)$  holds.  $\square$

**Corollary:** The algorithm will produce "holiest" tree rooted at  $r$  in linear time.

We now have produced our initial "Holiest" tree.

## 2 Minimum Cost Flow Problem and methods to solve them

There are several ways to define minimum cost flow and with different types of flows (non-negative and skew-symmetric), capacity or upper bound, edge demand or lower bound, edge cost, and with flow balance. The standard way to define:

$$\begin{array}{l}
 \min < \text{cost}, \text{flow} > \\
 \text{s.t } \sum_{u \rightarrow v} \text{flow}(u \rightarrow v) - \sum_{v \rightarrow w} \text{flow}(v \rightarrow w) = 0 \\
 \text{flow}(u \rightarrow v) = b(u \rightarrow v) \\
 \text{flow} \geq 0
 \end{array}$$

The easiest solution we can find is using augmenting cycle:

We can run any maximum flow algorithm to find feasible solution to the problem (neglecting the cost). Let the augmenting cycle be a cycle with negative cost. Then sending a flow through this cycle would reduce the total cost, while maintaining the feasible property.

We can also define reduced cost as follows:

Let  $\phi(v)$  be a any potential function on a vertex. Then  $\bar{\text{cost}}(u \rightarrow v) = \phi(u) - \phi(v) + \text{cost}(u \rightarrow v)$  satisfies the condition that  $\text{cost}(C) = \bar{\text{cost}}(C)$  for any cycle  $C$ .

**Lemma:** A feasible flow  $f$  is optimal  $\leftrightarrow$  there is no augmenting cycle in the residual graph.

**Proof:**  $\rightarrow$  Suppose the flow is optimal. If there is an augmenting cycle  $C$ , then we can send flow through  $C$  and reduce the cost of current flow, contradicting the  $f$  is optimal.

$\leftarrow$  Suppose there is no augmenting cycle. Let  $\phi(v) = [\text{Shortest path from } s \text{ to } v \text{ with respect to the cost function}]$ . Then  $\text{cost}(u \rightarrow v) = \phi(u) - \phi(v) + \text{cost}(u \rightarrow v) \geq 0$ . For the new cost function, sending more flow in a new residual graph would increase the cost of any flow, therefore  $f$  is optimal.  $\square$

We can find the augmenting cycle in a graph systematically as follows:  
Let  $T$  be any fixed spanning tree of  $G$ . Define new potential function for each vertex

$$\text{slack}_T(u \rightarrow v) = \begin{cases} 0 & , \text{ if } u \rightarrow v \in T \\ \sum_{e \in \text{cycle in } T \cup \{u \rightarrow v\}} \text{cost}(u \rightarrow v) & , \text{ Otherwise} \end{cases}$$

Above definition preserves the property that  $\text{slack}_T(C) = \text{cycle}(C)$  for any cycle  $C$  in  $G$ . Therefore, we can essentially find negative reduced cost edge in residual graph and push flow through it and do pivoting. Spanning tree  $T$  will be updated as follows:

**Update T:**

- Find bottleneck capacity in a cycle
- Push the flow amount equal to bottleneck capacity through the cycle
- Pivot out the bottleneck capacity edge, and pivot in the dart with negative reduced cost
- Recompute vertex potentials

### 3 Finding pivot quickly

- What data structure do we maintain in the  $G^*$ ? Finding shortest path in network can also be understood as a Linear Programming problem as follows:
- How do we find next pivot quickly using above structure?

There are two ways to represent the flow in the graph:

- $f(u \rightarrow v) = -f(v \rightarrow u)$ , for all edges
- $f(u \rightarrow v) \geq 0$  and  $f(u \rightarrow v) = 0$  if  $f(u \rightarrow v) > 0$

**Transshipment problem**

$$\begin{array}{l} \min < f, \$ > \\ \text{s.t } \partial f = b \\ f \geq 0 \end{array}$$

Under generic assumption on  $f, \$$ :

- Basis spanning tree  $T$ , there is a unique  $\text{flow}_T$  that satisfies the condition  $\partial \text{flow}_T = b$ ,  $\text{flow}_T(e)$  is nonzero only for edges in  $T$ .
- There exist a unique spanning tree  $T_{\text{OPT}}$  with  $\text{flow}_{T_{\text{OPT}}}$  is optimal.

Subsequently, we can define slack as follows:

For fixed spanning tree  $T$ , there is unique cycle  $C = T \cup \{e\}$  for edge  $e$  not in  $T$ .  $\text{slack}(e) = \sum_{l \in C} \$ (l)$ , and 0 otherwise. **Observe that slack is not negative, since otherwise there is no optimal solution to our LP**

The main LP is:

$$\boxed{\begin{array}{l} \min < f, \text{slack}_T > \\ \text{s.t } \partial f = \partial \text{flow}_T \\ f \geq 0 \end{array}}$$

$$\boxed{\begin{array}{l} \min < s, \text{flow}_T > \\ \text{s.t } \partial s = \partial \text{slack}_T \\ s \geq 0 \end{array}}$$

And in the case of non-planar embedded graph with genus  $g$ :

$$\boxed{\begin{array}{l} \min < f, \text{slack}_T > \\ \text{s.t } \partial f = \partial \text{flow}_T \\ f \geq 0 \end{array}}$$

$$\boxed{\begin{array}{l} \min < s, \text{flow}_T > \\ \text{s.t } \partial s = \partial \text{slack}_T \\ [s] = [\text{slack}_T] \\ s \geq 0 \end{array}}$$

Consider the primal LP, We can rewrite the constraints as follows:

$$\partial f = \partial \text{flow}_T \iff \sum_u f(u \rightarrow v) - f(v \rightarrow u) \iff$$

$$\gamma(u) \sum_u (f(u \rightarrow v) - f(v \rightarrow u)) = \sum_{u \rightarrow v} f(u \rightarrow v)(\gamma(u) - \gamma(v)) = \sum_{u \rightarrow v} f(u \rightarrow v)\gamma(u \rightarrow v)$$

Observe here that  $\gamma(u \rightarrow v) = -\gamma(v \rightarrow u)$  We can define  $s$  in terms of  $\gamma$  by setting the negative values to be 0, and we will get the exact dual program defined above.

## 4 Analysis

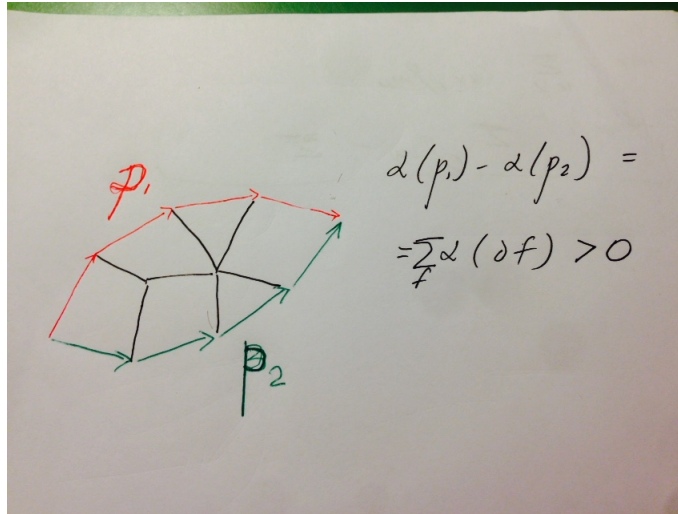
- Building initial tree
- Pivoting
- Number of times each edge is pivoted
- Overall running time

Couple questions regarding the slack and flow:

- If the fixed tree  $T$  is arbitrary tree (not necessarily the Holiest Tree, then the flow in the answer does not have to be optimal) but solution to the linear program  $\min < f, \text{slack}_T >$  is equal to the answer from fixed tree  $T$ , not necessarily the optimal solution. That is because for each vertex, the demand satisfies the constraint and if we consider the tree  $T$ , then the value of  $\min < f, \text{slack}_T >$  would be 0, implying it is the optimal solution. (Sum cannot be negative since otherwise there is no optimal solution)
- The reason we picked the slack as the way we defined is due to the fact that slack is not negative, ensuring that the nothing bad happens.
- What makes the non-planar case special with  $2g$  extra constraints?
- How does the slack in dual representation help us to find the pivots quickly?

## 5 Additional

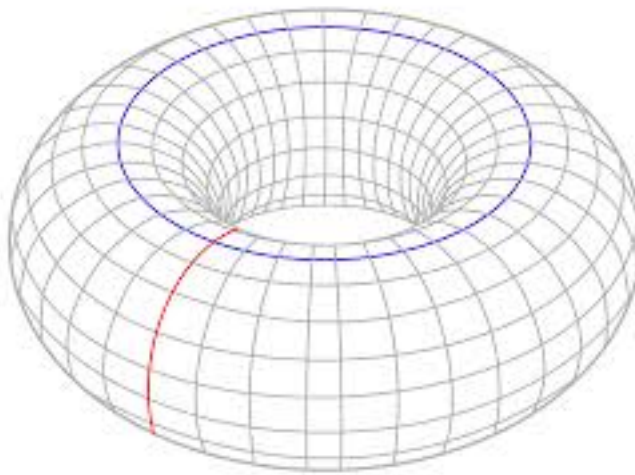
**NOTE:** Necessity of the  $\alpha$  definition on the edges for "Holiness"  
Consider the following picture:



By the definition of *alpha*:

$$\alpha(p_1) - \alpha(p_2) = \sum_f \alpha(df) > 0$$

This will ensure that any two paths  $p_1, p_2$ , whose  $w(p_1) = w(p_2)$  and  $[p_1]_\Lambda = [p_2]_\Lambda$ , has  $\alpha(p_1) \neq \alpha(p_2)$

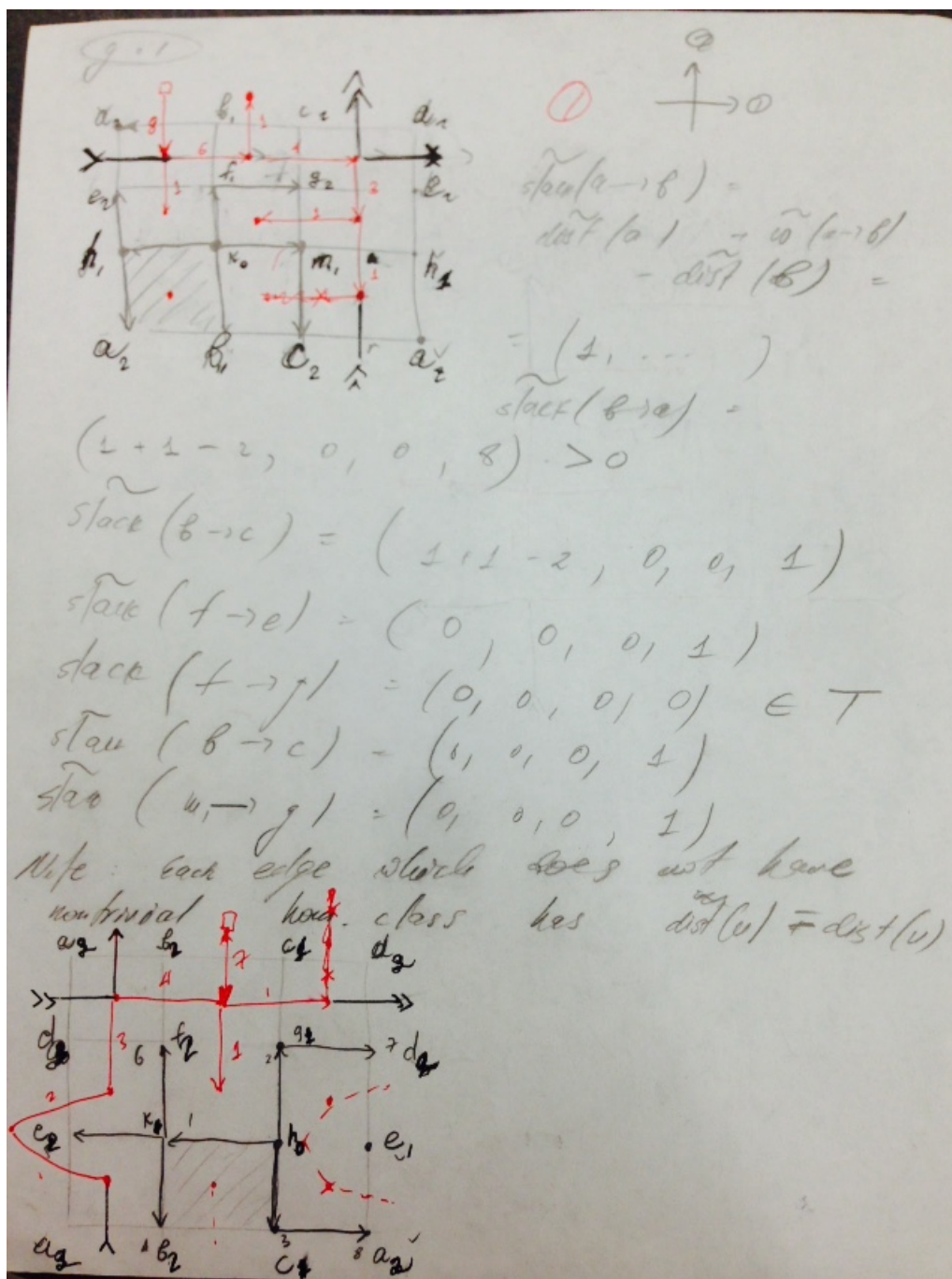


[http : //en.wikipedia.org/wiki/Homology\(mathematics\)](http://en.wikipedia.org/wiki/Homology(mathematics))

## 6 Working on examples:

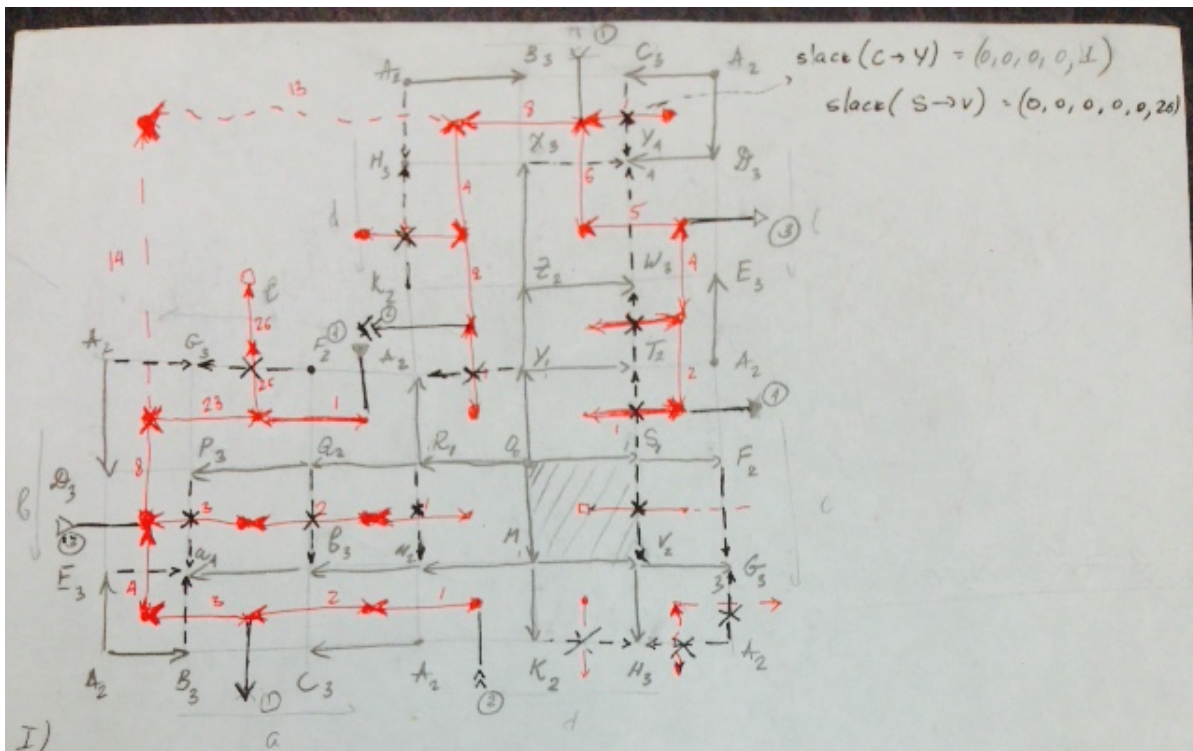
Difference of Holiest Tree and leftmost tree:

On genus  $g = 1$  surface:





On genus  $g = 2$  surface:



$$sh_{\alpha}(b_5 \geq c) > 0$$

$$\text{stack}(A \rightarrow H) = (0, -1, -1, -1, -1, -13)$$

$$\widetilde{\text{Stack}}(B \rightarrow a) = (0, -1, -1, 0, 0, -3)$$

for 1, 2, 3, 4 cycles.



11) state  $(A \rightarrow 11) = (0, 1$

$$\widehat{\text{face}}(w-ry) = (0, 0, 0, 5, 1, 5)$$

$$\text{span} \{4 \rightarrow 6\} = (0, 4, -1, -1, 1, 14)$$

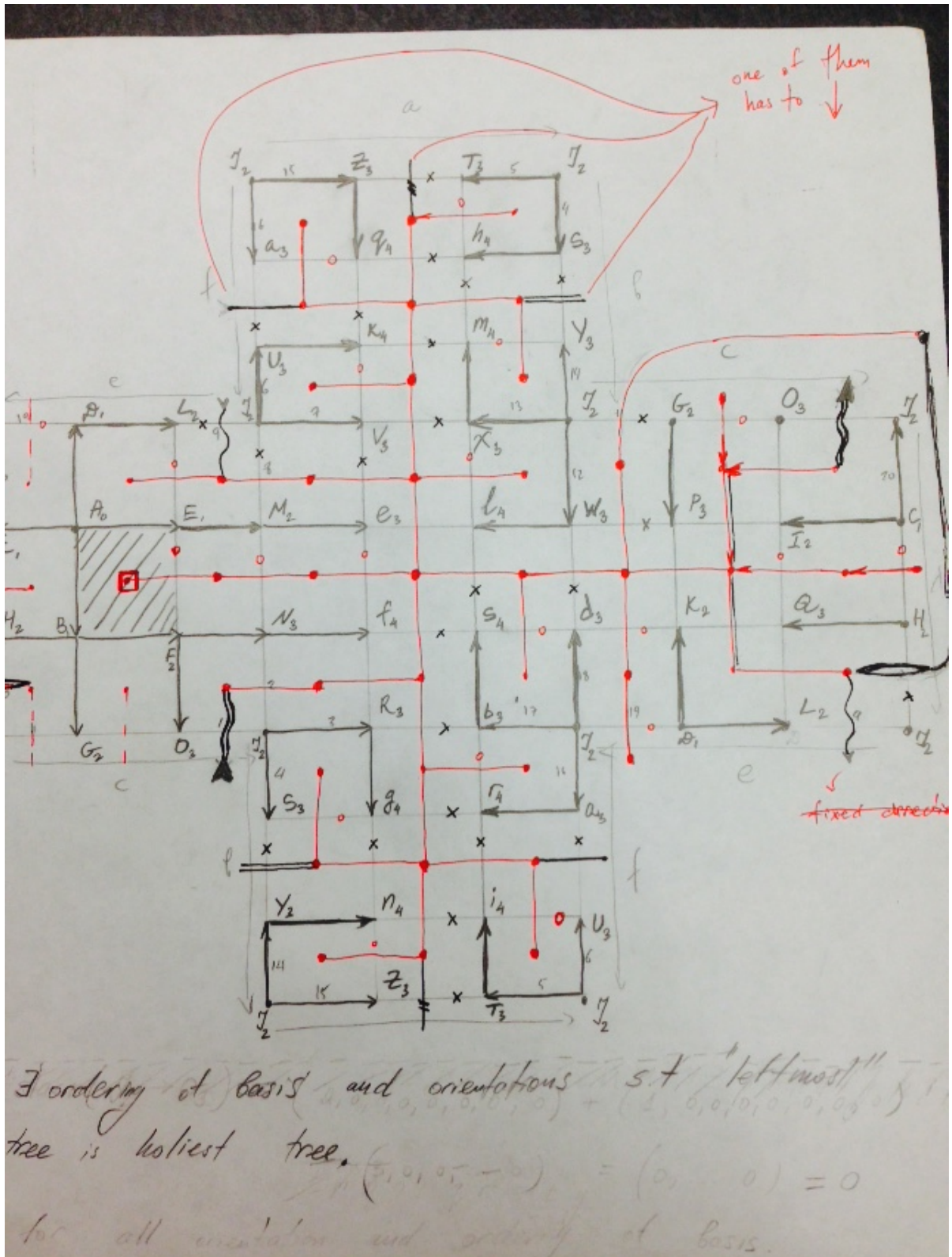
$$\widehat{\text{sp}}_A(E - a) = (0, 1, -1, 0, 0, 4)$$

$$\tau_{\text{ap}}(B \rightarrow a) = (0, 1, -1, 0, 0, 3)$$

 1  
 2, 3, 4

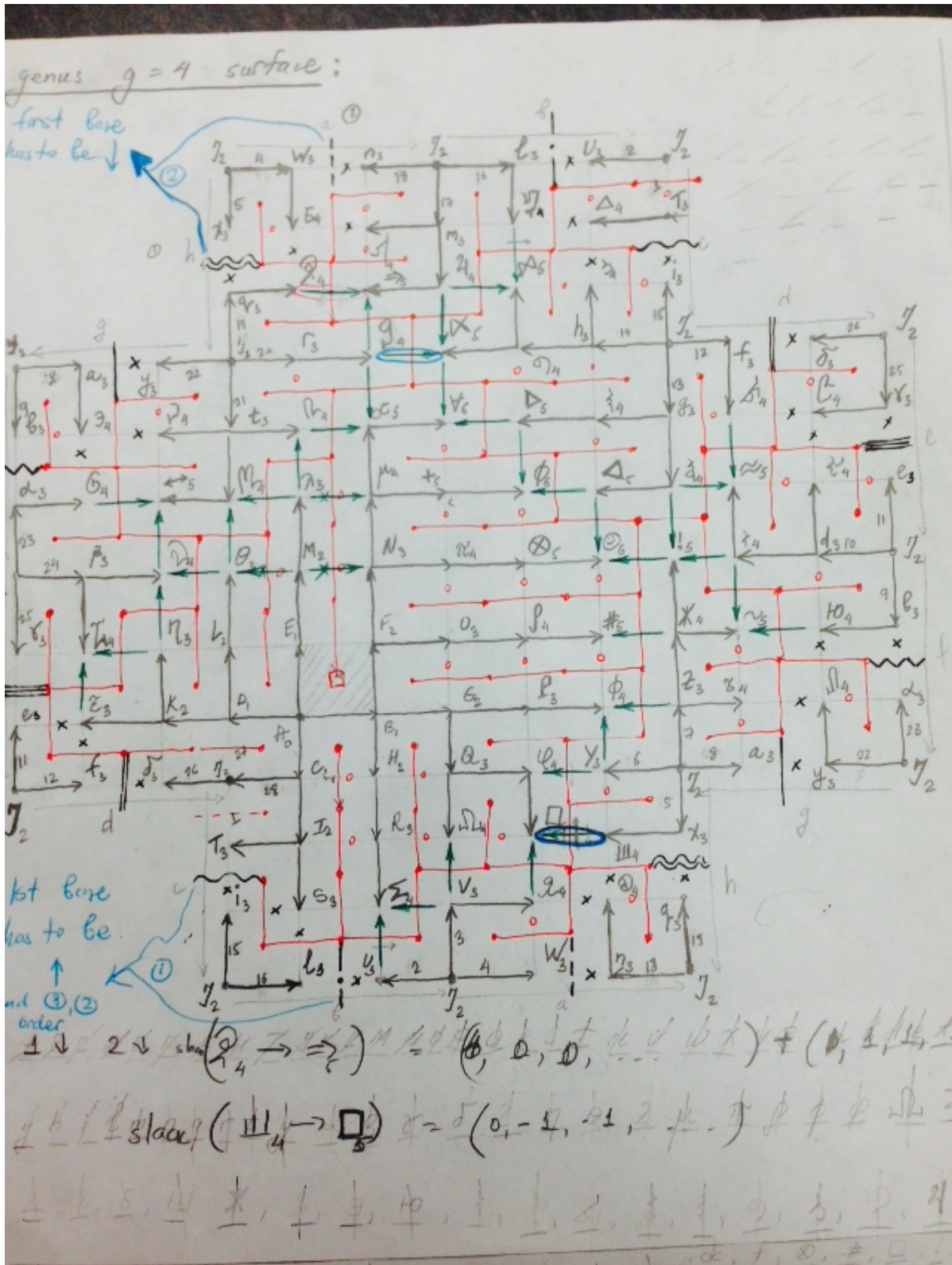
So we might be able to argue  $\exists$  ordering and direction s.t.  $\text{slack}(u \rightarrow v) \geq 0$  for all  $u \rightarrow v \notin T$ .

On genus  $g = 3$  surface:

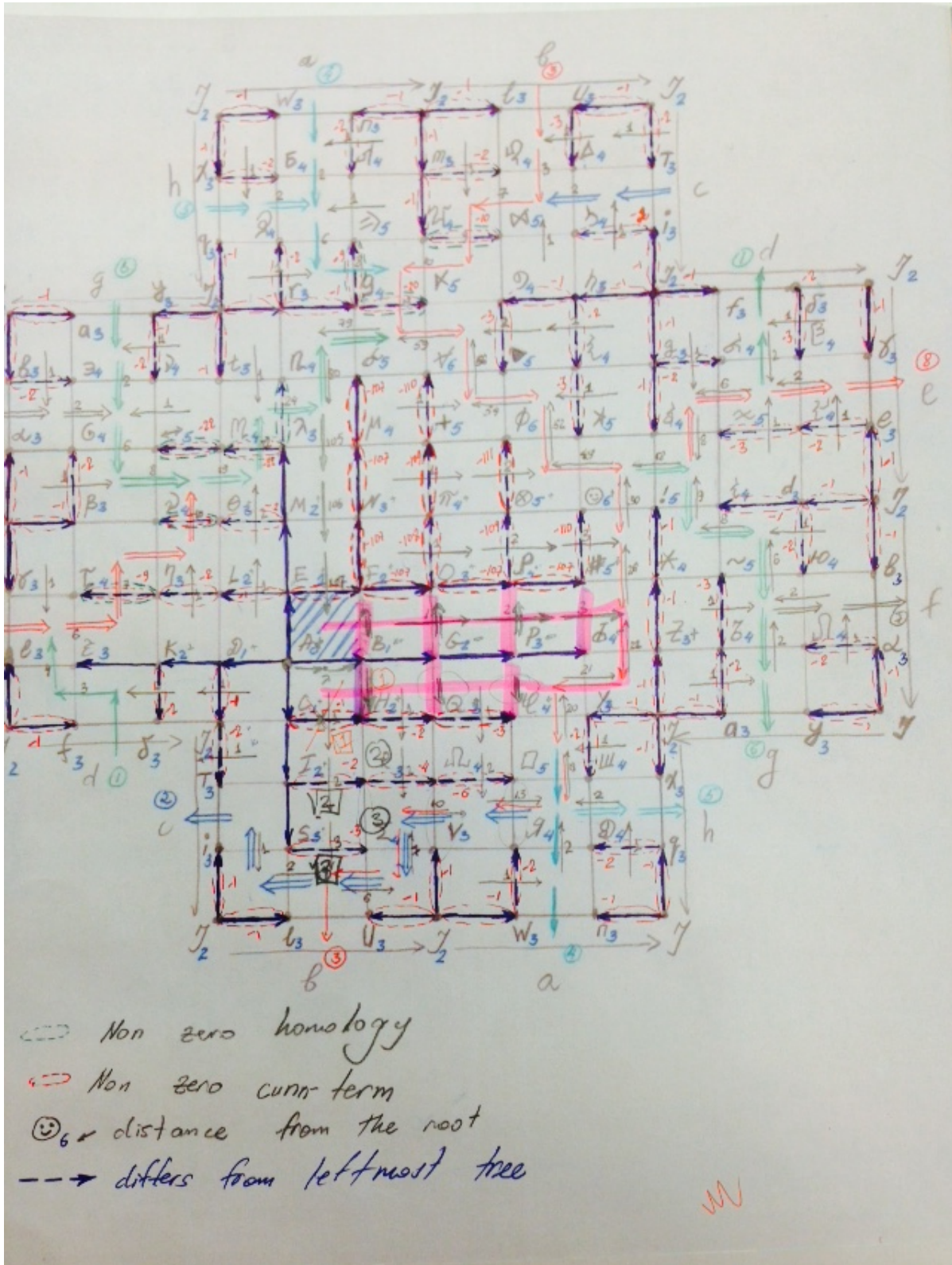




On genus  $g = 4$  surface with initial Holy Tree build:

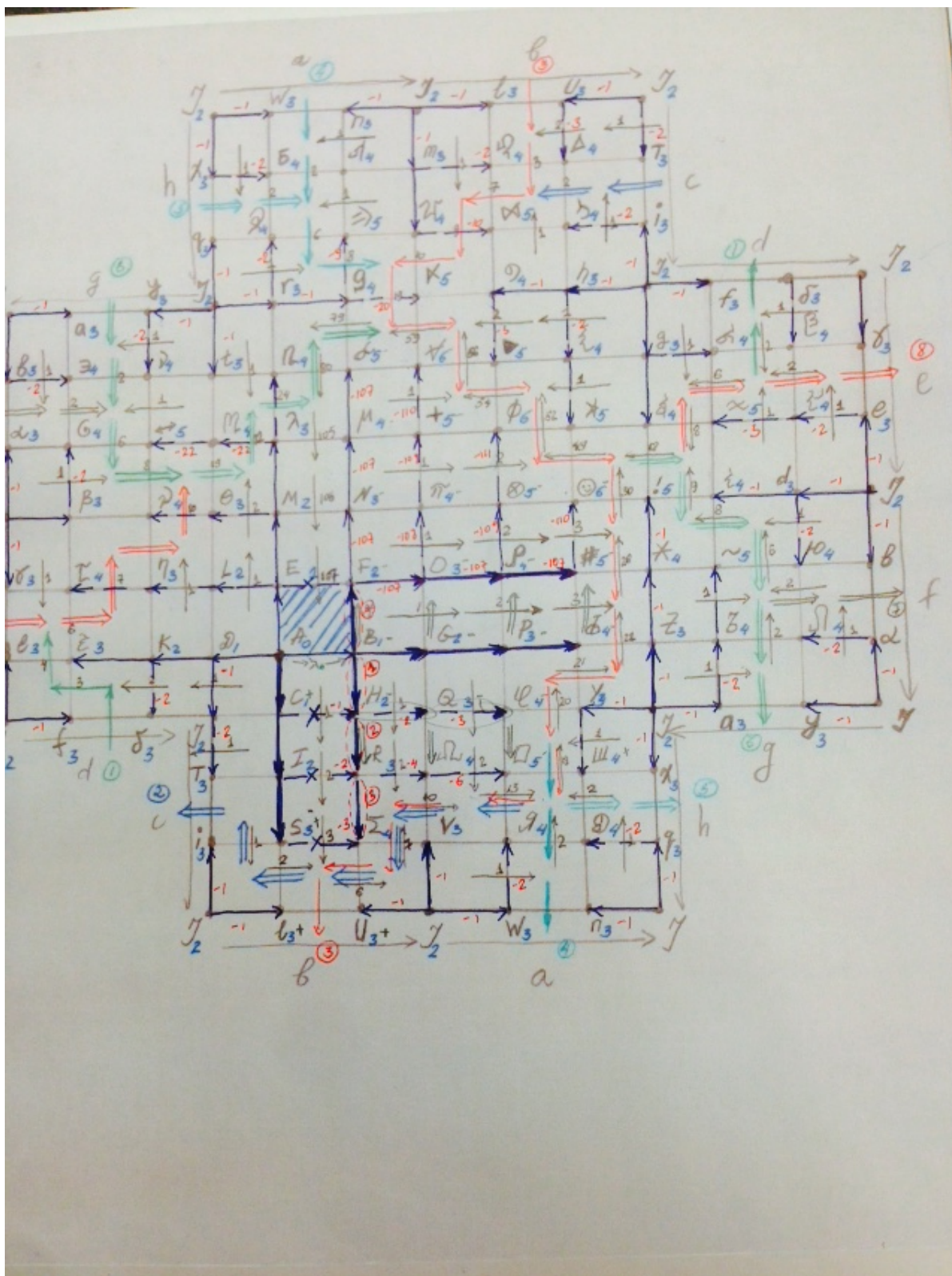


On genus  $g = 4$  surface with initial pivot:

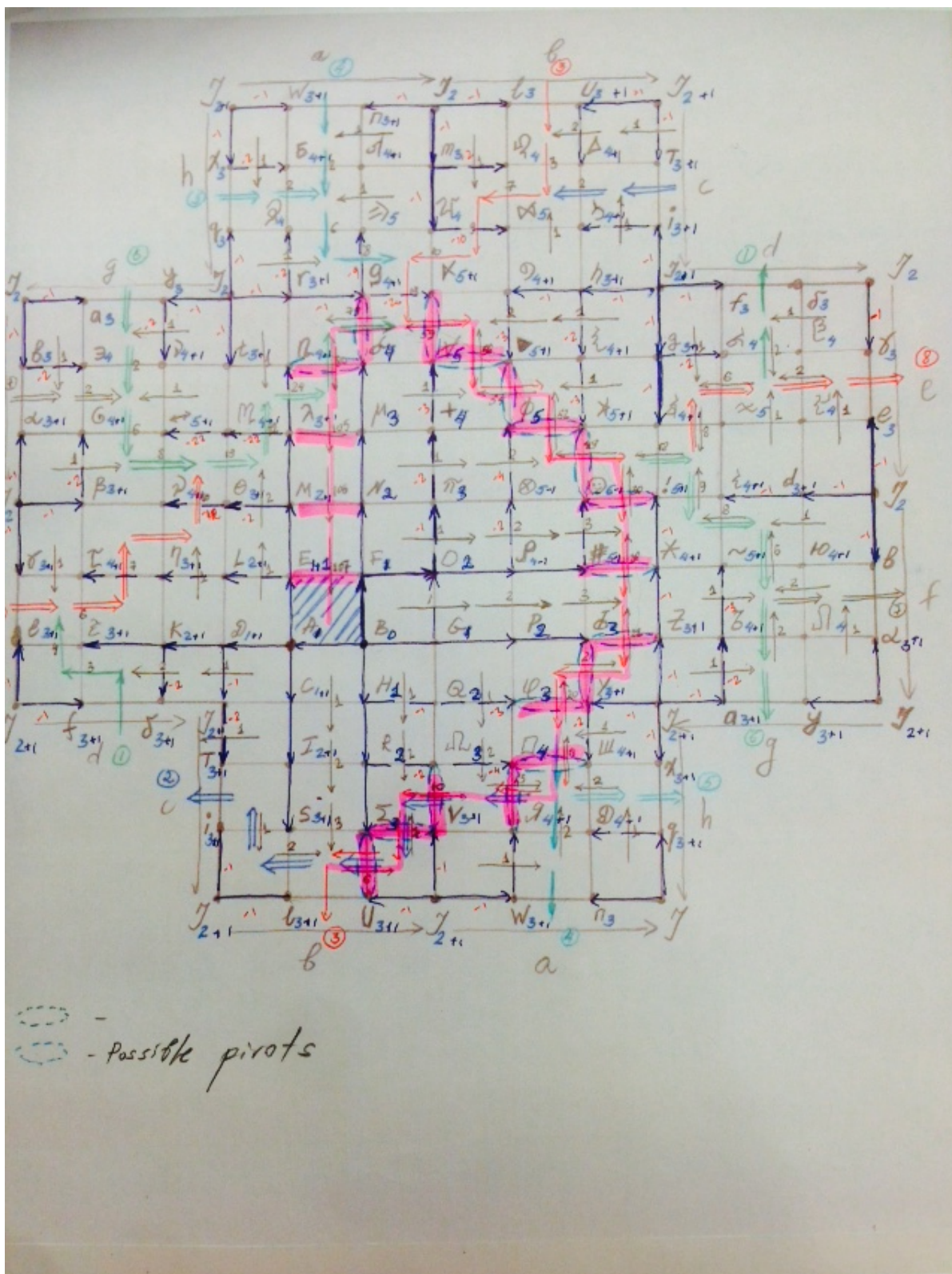




On genus  $g = 4$  surface with initial pivot:



On genus  $g = 5$  surface with initial pivot:



## 7 References:

- Cabello, Sergio, Erin W. Chambers, and Jeff Erickson. "Multiple-source shortest paths in embedded graphs." *SIAM Journal on Computing* 42.4 (2013): 1542-1571.
- Eisenstat, David, and Philip N. Klein. "Linear-time algorithms for max flow and multiple-source shortest paths in unit-weight planar graphs." *Proceedings of the forty-fifth annual ACM symposium on Theory of computing*. ACM, 2013.
- Erickson, Jeff. "Maximum flows and parametric shortest paths in planar graphs." *Proceedings of the twenty-first annual ACM-SIAM symposium on Discrete Algorithms*. Society for Industrial and Applied Mathematics, 2010.