Multiple-source shortest paths with unit weights in embedded graphs

Abstract

We describe a new algorithm that computes multiple-source shortest paths from vertices in a given boundary face to all other vertices in an embedded graph with unit weight edges.

1 Introduction

Recently, Eisanstat and Klein [?] introduced a new algorithm for computing multiple source shortest path problem for a planar graphs in linear time. Our paper attempts to generalize this idea to embedded graphs. In their paper, they maintain so-called leafmost shortest tree to get around an issue of ambiguous shortest paths between a pair of vertices. There is no direct way to generalize leafmost tree computing process to an embedded graphs, largely because there is no way to define leafmost edge of a data structure in an embedded graphs. In the following section we introduce terms that alleviate the process of ambiguous pivoting.

We can formally define multiple-source shortest paths problem as follows:

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 face f of G. $\forall v \in f$, find a shortest path to $\forall u \in V$.*

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

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

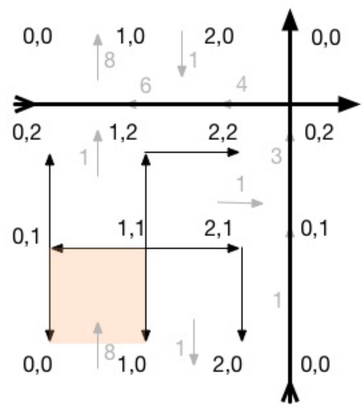
$$[e]_i = \begin{cases} 1 & \text{, if } e \in \lambda_i \\ -1 & \text{, if } rev(r) \in \lambda_i \\ 0 & \text{, otherwise} \end{cases}$$

Furthermore, we define leafmost term α recursively as follows:

$$\alpha(\vec{e}^{*}) = \begin{cases} 1 & \text{, if } rev(e^*) \text{ is a leaf dart in } C \\ \displaystyle\sum_{\text{tail}(\vec{e}^{\,\prime *})=\text{head}(\vec{e}^{*})} \alpha(\vec{e}^{\,\prime}) & \text{, otherwise} \end{cases}$$

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

Let $\tilde{w}(\vec{e}) = \langle 1, [\vec{e}], \alpha(\vec{e}) \rangle$ be new weight vector for each edge in G. We refer to each component by length, homology, and leafmost terms respectively.



Definition. *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:*

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

2 Need of holiness

We provide a simple explanation on why it is necessary to introduce holiness to cope with ambiguity.

???

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, w̃, r):
  Set dist[r] ← ⟨0, [0⃗], 0⟩
    pred(r) ← NULL
  for all v : v ≠ r
    dist[r] ← ⟨∞, [∞], ∞⟩
    pred(r) ← NULL
  put r into queue
  while queue is not empty:
    Let u ← dequeue item
    for all u → v
      if v is not marked
        mark v and put in the queue
      if isTense(u → v)
        relax(u → v)
```

```
isTense(u → v):
  return dist[u] + w̃(u → v) < dist[v]
```

```
relax(u → v):
  dist[v] ← dist[u] + w̃(u → v)
  pred[v] ← u
```

Observation. *Each vertex will be added once to the queue.*

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

Lemma 2.1. *If there is no tense edge in G, then for each $v : r \to \ldots \to pred(pred(v)) \to pred(v) \to v$ is the holiest path from r to v.*

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

Base. *$dist[v].length = 0$, then v = r, so the claim holds trivially.*

Induction Step. *Suppose the claim is true for all vertex $v \in V$ such that $dist[v].length < d$ for some d. Consider vertex v such that $dist[v].length = d$. By induction hypothesis, all vertices with $dist[u].length = d-1$ have holiest path correctly updated. By definition, $dist[v] = \min_{u \to v}\{dist[u]+\tilde{w}(u \to v)\}$, here $dist[u].length = d-1$. By Induction hypothesis, $dist[u]$ is not tense and can construct holiest path to u, so if there is no tense edge in G then $dist[v] = min\{dist[u] + \tilde{w}(u \to v)\}$ holds.*

□

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

We now have produced our initial Holiest tree.

3 Moving Along an Edge

Consider a single edge uv, which is on the boundary face f of G. Suppose we already computed the holy-tree T_u rooted at u. We transform T_u into the holy-tree T_v as follows. First, we insert a new vertex s in the interior of the uv, bisecting it into two edges su and sv with weights:

$$w_0(s \to u) = \langle 0, [\vec{0}], 0 \rangle$$

$$w_0(s \to v) = \langle 1, [w(u \to v)], \alpha(w(u \to v)) \rangle = w(u \to v)$$

Observe that this condition implies $s = u$, therefore $T_s = T_u$. We reduce distances to u and v as follows:

$$w_\epsilon(s \to u) = \langle 0, -[w(u \to v)], -\alpha(w(u \to v)) \rangle$$
$$w_\epsilon(s \to v) = \langle 1, [\vec{0}], 0 \rangle$$

Since we reduced distance to all vertices in the graph equally, the process does not introduce any pivots. Then we define a parametric weights as follows:

$$w_\lambda(s \to u) = \langle 0, -[w(u \to v)], -\alpha(w(u \to v))\rangle - \lambda$$
$$w_\lambda(s \to v) = \langle 1, [\vec{0}], 0\rangle + \lambda$$

Every other dart $x \to y$ has constant parametric weight $w_\lambda(x \to y) = w(x \to y)$. We then maintain the holy tree T_λ rooted at s, with respect to the weight function w_λ, as λ increases continuously from 0 to $\langle 1, [\vec{0}], 0\rangle$. When $\lambda = w(u \to v)$, $T_\lambda = T_v$.

In the following algorithm, **pred** defines Holy tree rooted at u, and **dist** is corresponding distance to each vertex in the graph.

<u>**MoveAlongEdge**</u>($G, u \to v, dist, pred$):
  Add new vertex s
  $pred[u], pred[v] \leftarrow s$
  $\lambda \leftarrow 0$

  $w(s \to u) \leftarrow \langle 0, -[w(s \to u)], -\alpha(w(s \to u))\rangle$
  AddSubtree($\langle 0, -[w(s \to u)], -\alpha(w(s \to u))\rangle, u$)

  $w(s \to v) \leftarrow \langle 1, [\vec{0}], 0\rangle$
  AddSubtree($\langle 0, -[w(s \to u)], -\alpha(w(s \to u))\rangle, v$)

  while $\lambda < \langle 1, [\vec{0}], 0\rangle$:
    pivot $\leftarrow$ FindNextPivot
    If **pivot** is non NULL **AND** ($\lambda + slack(\textbf{pivot})/2) < \langle 1, [\vec{0}], 0\rangle$
      Pivot(**pivot**)
      $\lambda \leftarrow \lambda + slack(\textbf{pivot})/2$
    else
      $\delta = \langle 1, [\vec{0}], 0\rangle - \lambda$
      AddSubtree(δ, u)
      AddSubtree($-\delta, v$)
      $\lambda \leftarrow \lambda + \delta$

4 Bounding number of pivots

We introce a clocking lemma to prove that each edge is involved in pivoting process at most $O(g)$ times.

Lemma 4.1. *Here is the claim of the clocking lemma.*

Proof: Here is the proof. □

Theorem 4.1. *Total running time of MSSP is $O(gn)$.*

Proof: Building initial holy tree takes $O(n)$ time. The process of moving arond the face and pivoting takes $O(gn)$ as each dart enters and replaces $O(g)$ times. □

5 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?

6 Analysis

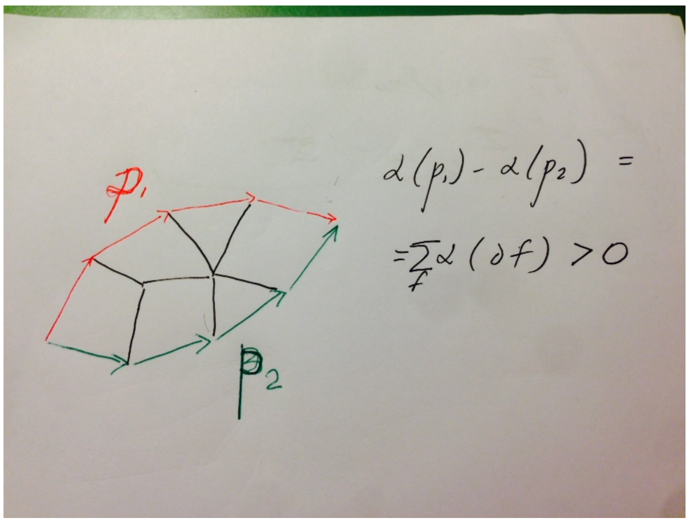
- Building initial tree
- Pivoting
- Number of times each edge is pivoted
- Overall running time

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?

7 Additional

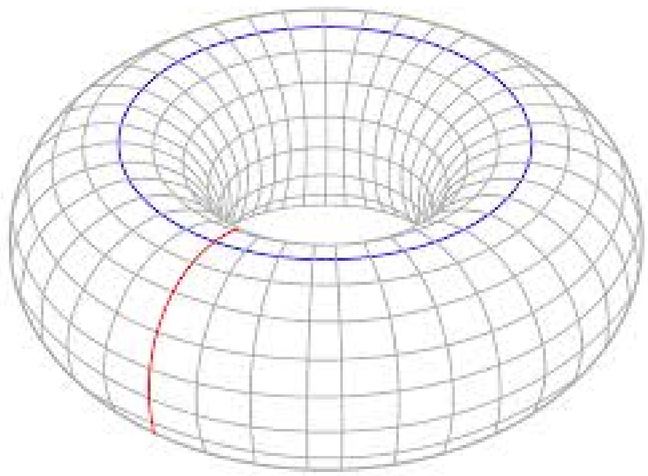
NOTE: Necessity of the α definition on the edges for holiness.
Consider the following picture:



By the definition of *alpha*:

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

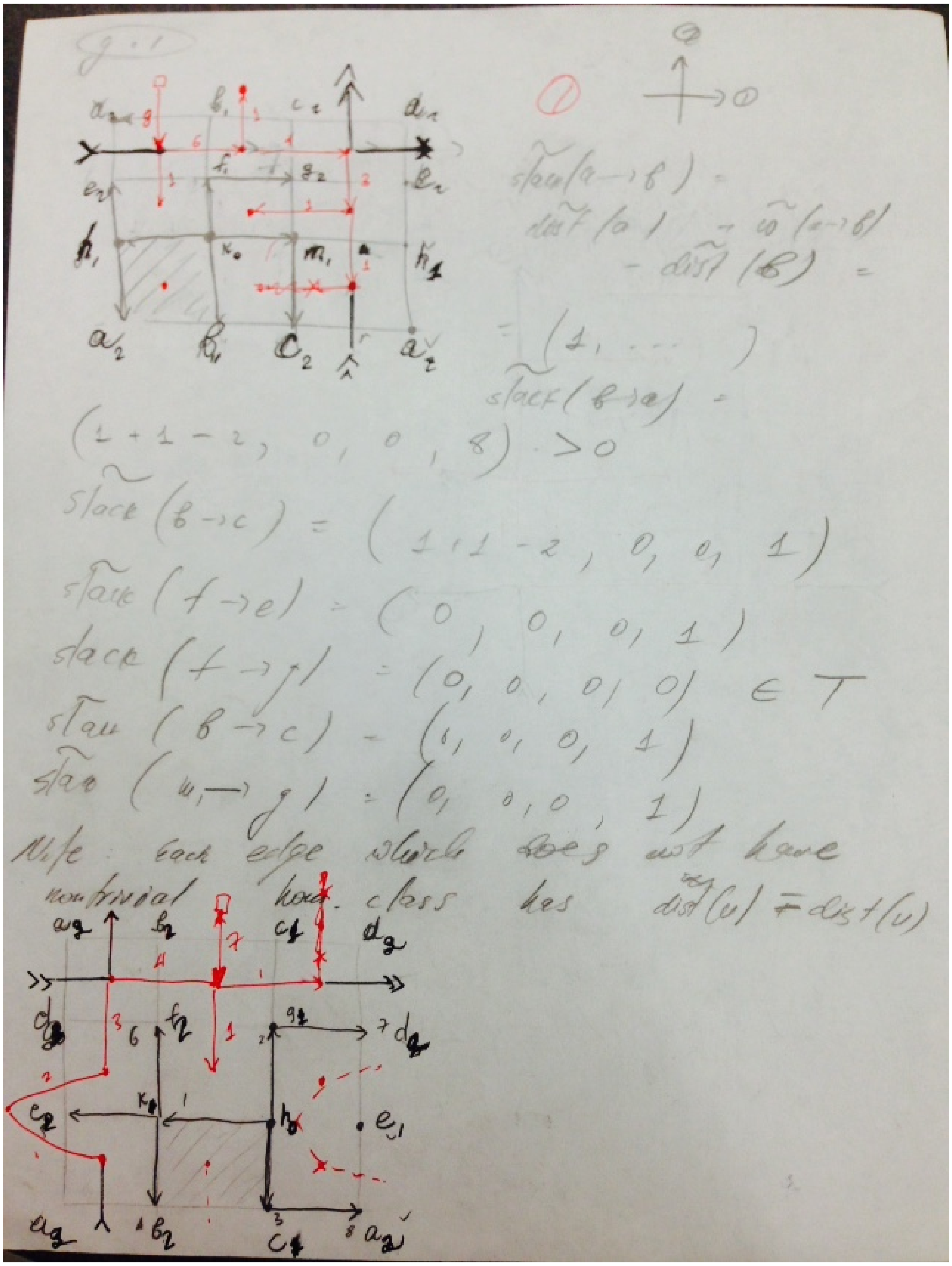
This will ensure that any two paths p_1, p_2, whose $w(p_1) = w(p_2)$ and $[p_1]_\Lambda = [p_2]_\Lambda$, has $\alpha(p_1) \neq \alpha(p_2)$



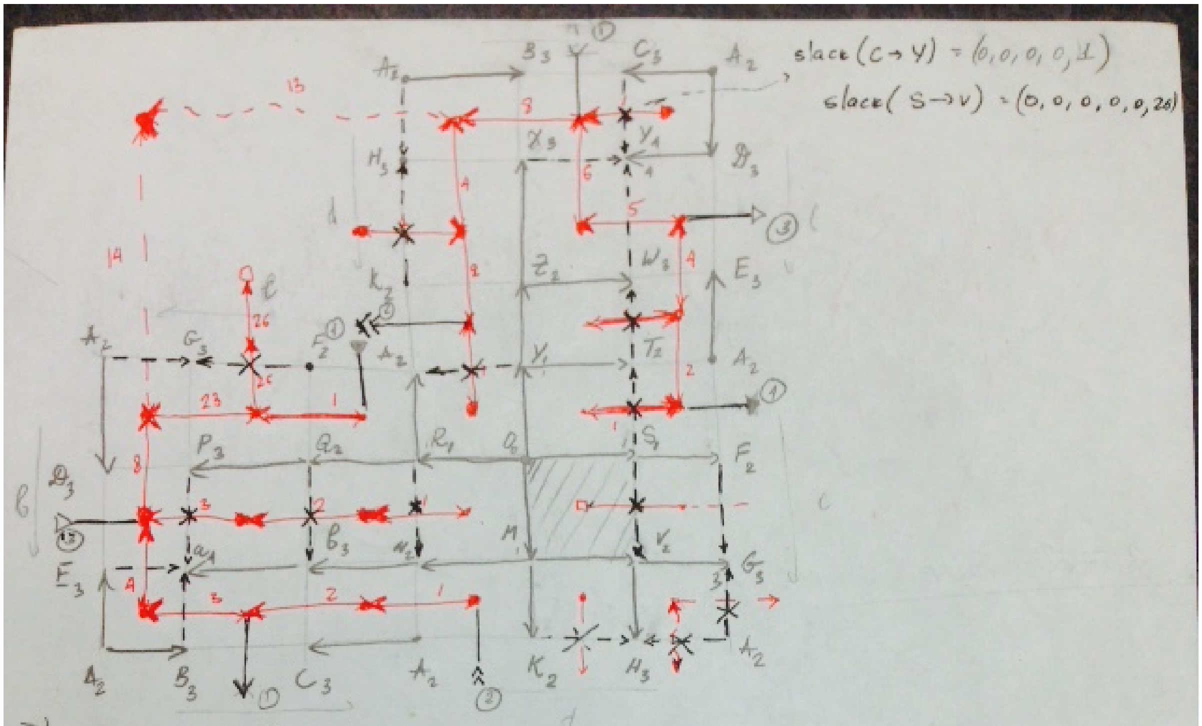
$http://en.wikipedia.org/wiki/Homology_(mathematics)$

8 Working on examples:

Difference of Holiest Tree and leftmost tree:
On genus $g = 1$ surface:



On genus $g = 2$ surface:



I)

$\widetilde{\text{slack}}(B_5 \to C) > 0$

$\widetilde{\text{slack}}(A \to H) = (0, -1, -1, -1, -1, -13)$

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

for 1, 2, 3, 4 cycles.

II) $\widetilde{\text{slack}}(A \to H) = (0, 1$

$\widetilde{\text{slack}}(W \to Y) = (0, 0, 0, 1, 1, 5)$ 1

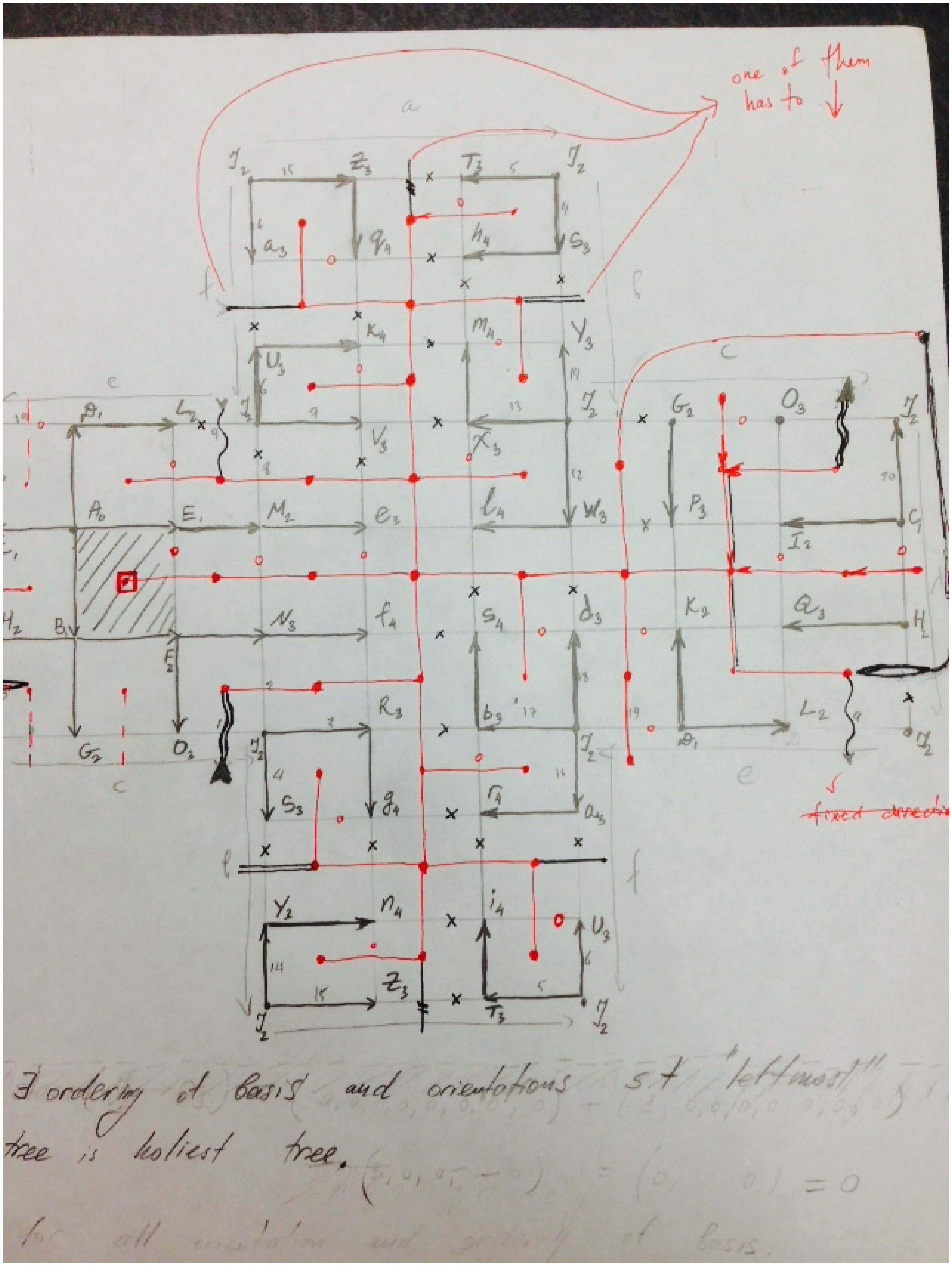
$\widetilde{\text{slack}}(A \to G) = (0, 1, -1, -1, -1, 14)$ 2, 3, 4

$\widetilde{\text{slack}}(E - a) = (0, 1, -1, 0, 0, 4)$

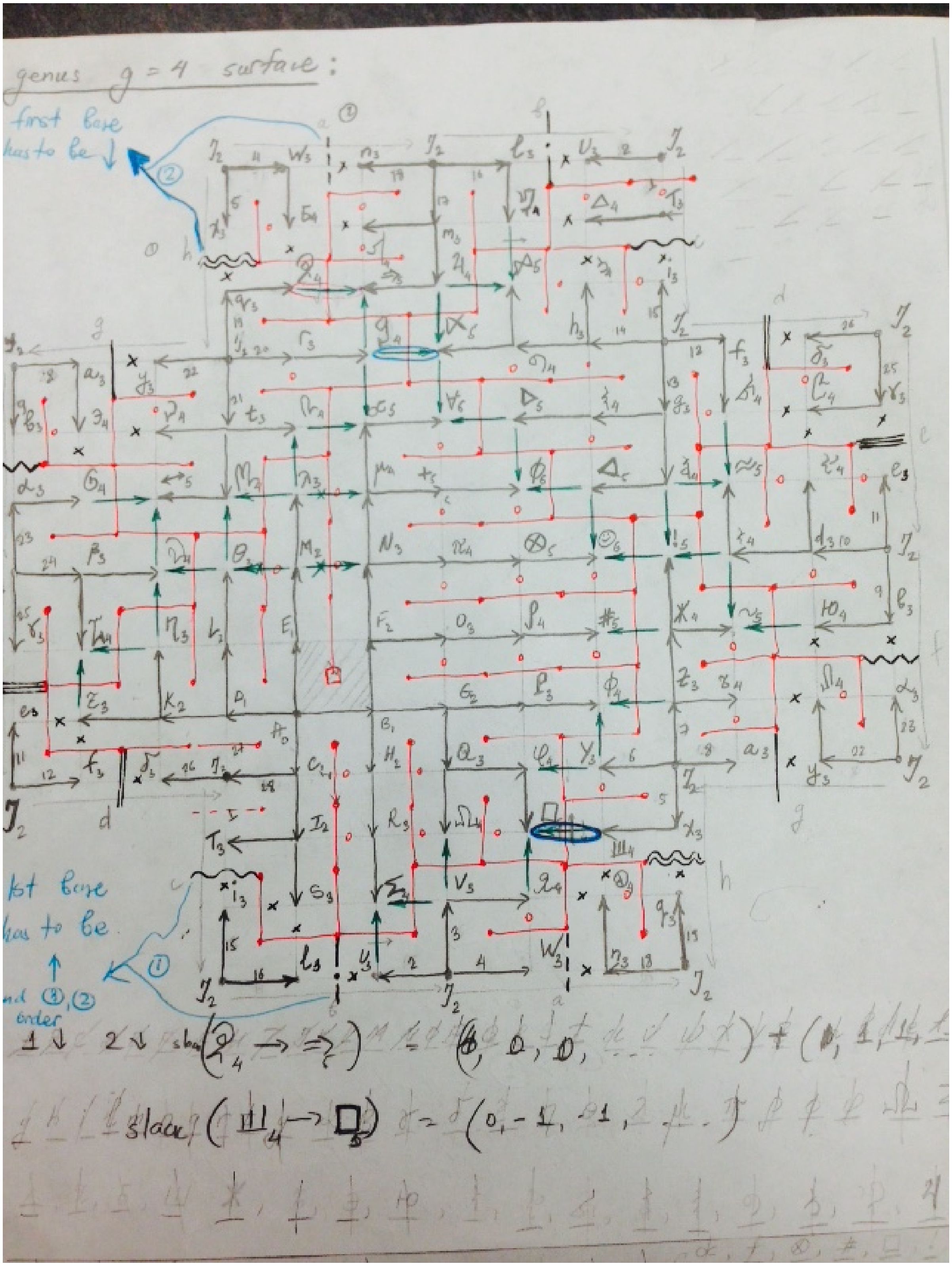
$\text{slack}(B \to a) = (0, 1, -1, 0, 0, 3)$

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

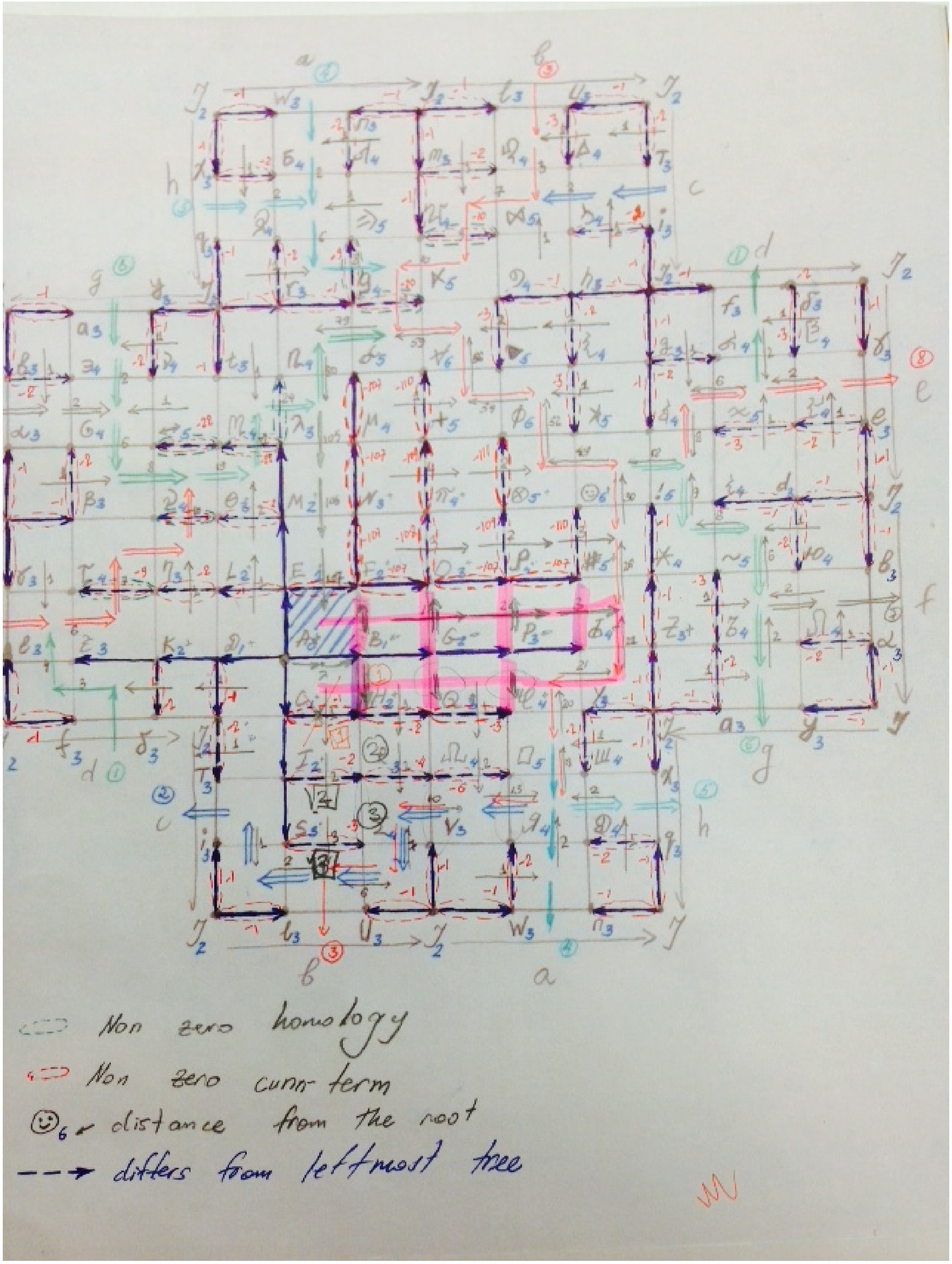
On genus $g = 3$ surface:



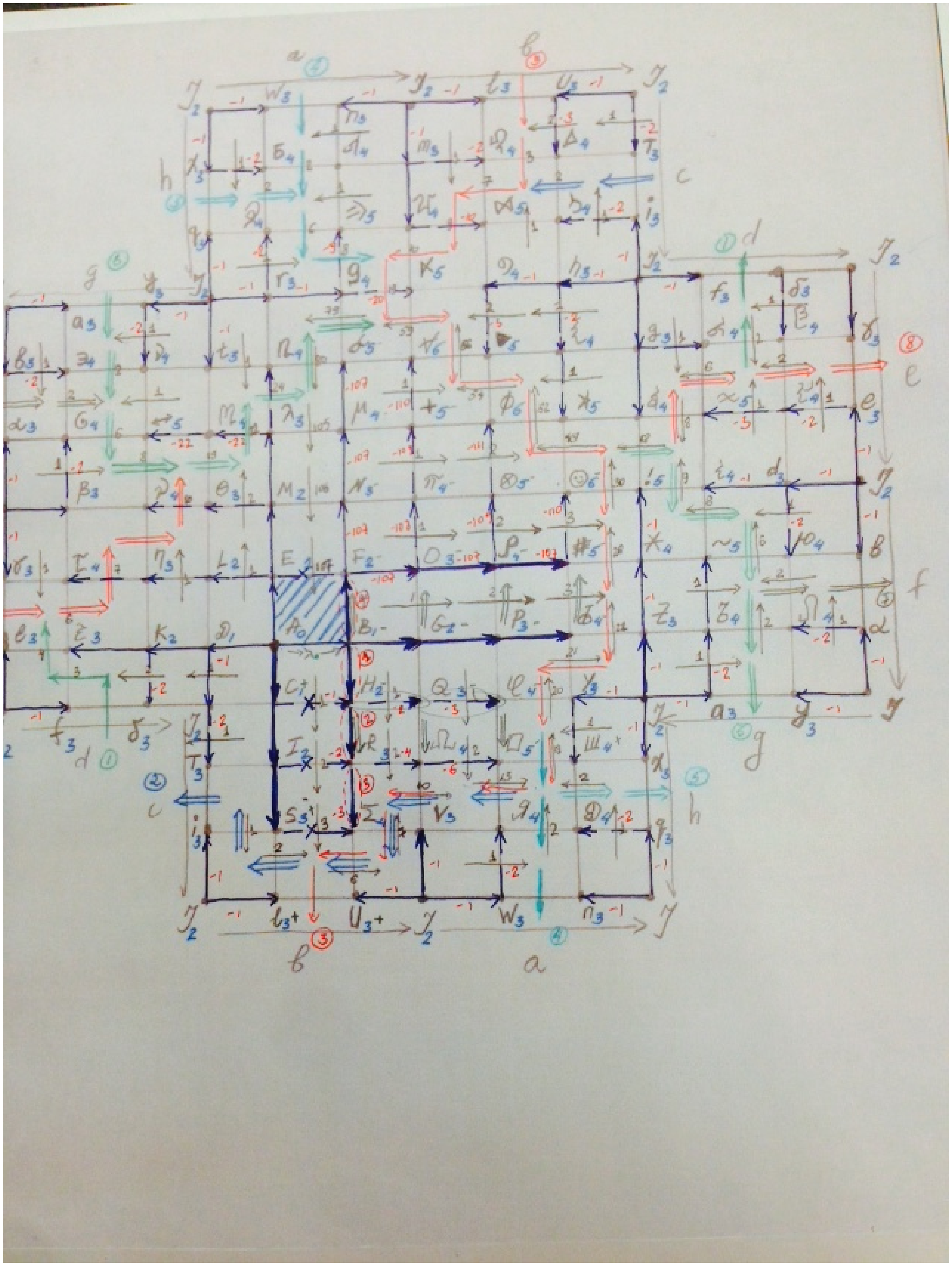
On genus $g = 4$ surface with initial Holy Tree build:



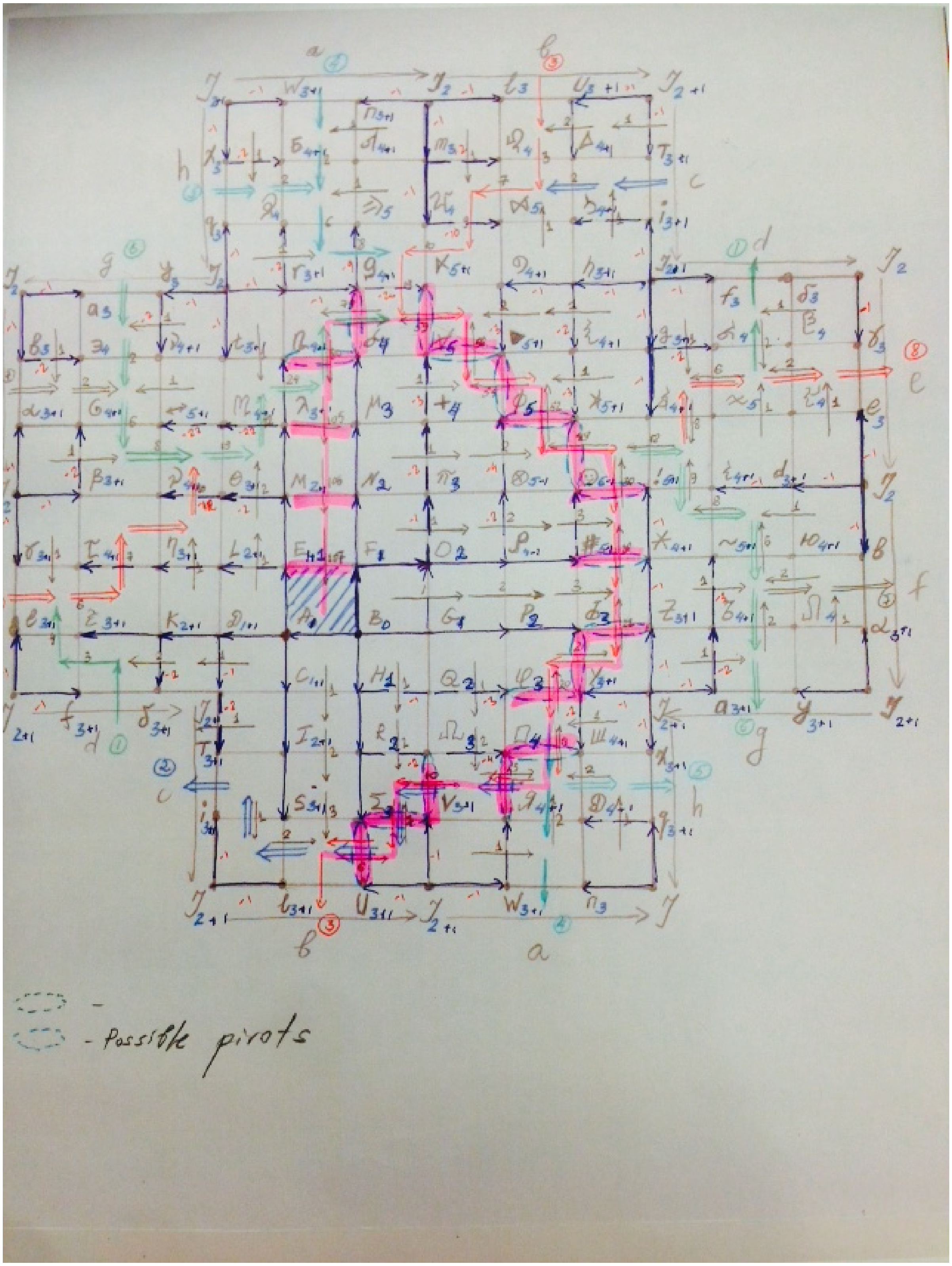
On genus $g = 4$ surface with initial pivot:



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



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



9 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.