

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

isTense($u \rightarrow v$):

return $dist[u] + \tilde{w}(u \rightarrow v) < dist[v]$

relax($u \rightarrow v$):

$dist[v] \leftarrow dist[u] + \tilde{w}(u \rightarrow v)$
 $pred[v] \leftarrow u$

Observation: Each vertex 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 $v : r \rightarrow \dots \rightarrow pred(pred(v)) \rightarrow pred(v) \rightarrow v$ is the holiest path from r to v .

Proof: Let's prove it by induction on $dist[v][0]$ 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, $dist[v] = \min_{u \rightarrow v} dist[u] + \tilde{w}(u \rightarrow v)$, 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 $dist[v] = \min\{dist[u] + \tilde{w}(u \rightarrow v)\}$ holds. \square

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

We now have produced our initial "Holiest" tree.

2 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:

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

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

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

$$w_\epsilon(s \rightarrow u) = \langle 0, -[w(u \rightarrow v)], -\alpha(w(u \rightarrow v)) \rangle$$

$$w_\epsilon(s \rightarrow v) = \langle 1, [\vec{0}], 0 \rangle$$

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:

$$w_\lambda(s \rightarrow u) = \langle 0, -[w(u \rightarrow v)], -\alpha(w(u \rightarrow v)) \rangle - \lambda$$

$$w_\lambda(s \rightarrow v) = \langle 1, [\vec{0}], 0 \rangle + \lambda$$

Every other dart $x \rightarrow y$ has constant parametric weight $w_\lambda(x \rightarrow y) = w(x \rightarrow y)$. 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 $\lambda = w(u \rightarrow v)$, $T_\lambda = T_v$.

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

```

MoveAlongEdge( $G, u \rightarrow v, dist, pred$ ):
  Add new vertex  $s$ 
   $pred[u], pred[v] \leftarrow s$ 
   $\lambda \leftarrow 0$ 

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

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

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

```

3 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:

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

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.

← 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 $\bar{\text{cost}}(u \rightarrow v) = \phi(u) - \phi(v) + \text{cost}(u \rightarrow v) \geq 0$. 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

4 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 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_{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:

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

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

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

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

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 γ by setting the negative values to be 0, and we will get the exact dual program defined above.

5 Analysis

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

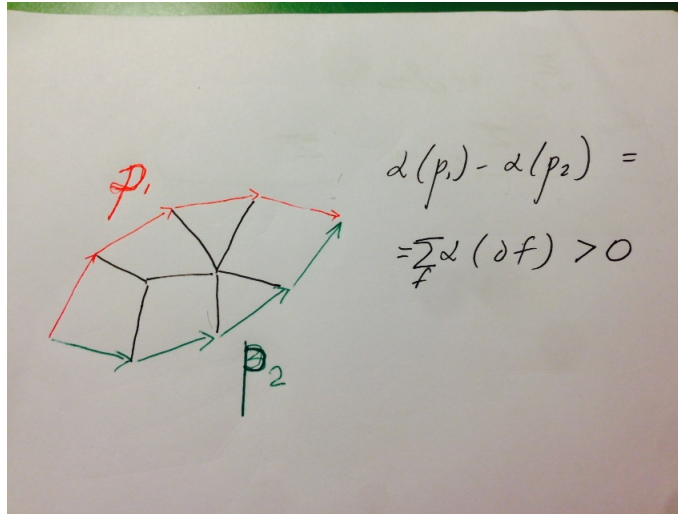
Couple questions regarding the slack and flow:

- 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?

6 Additional

NOTE: Necessity of the α definition on the edges for "Holiness"

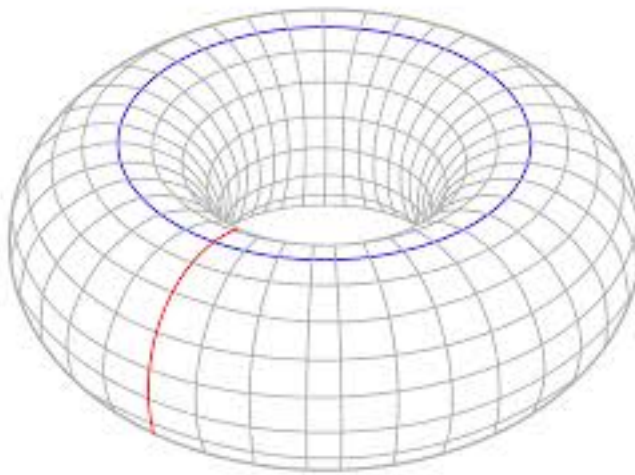
Consider the following picture:



By the definition of *alpha*:

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

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 $\alpha(p_1) \neq \alpha(p_2)$

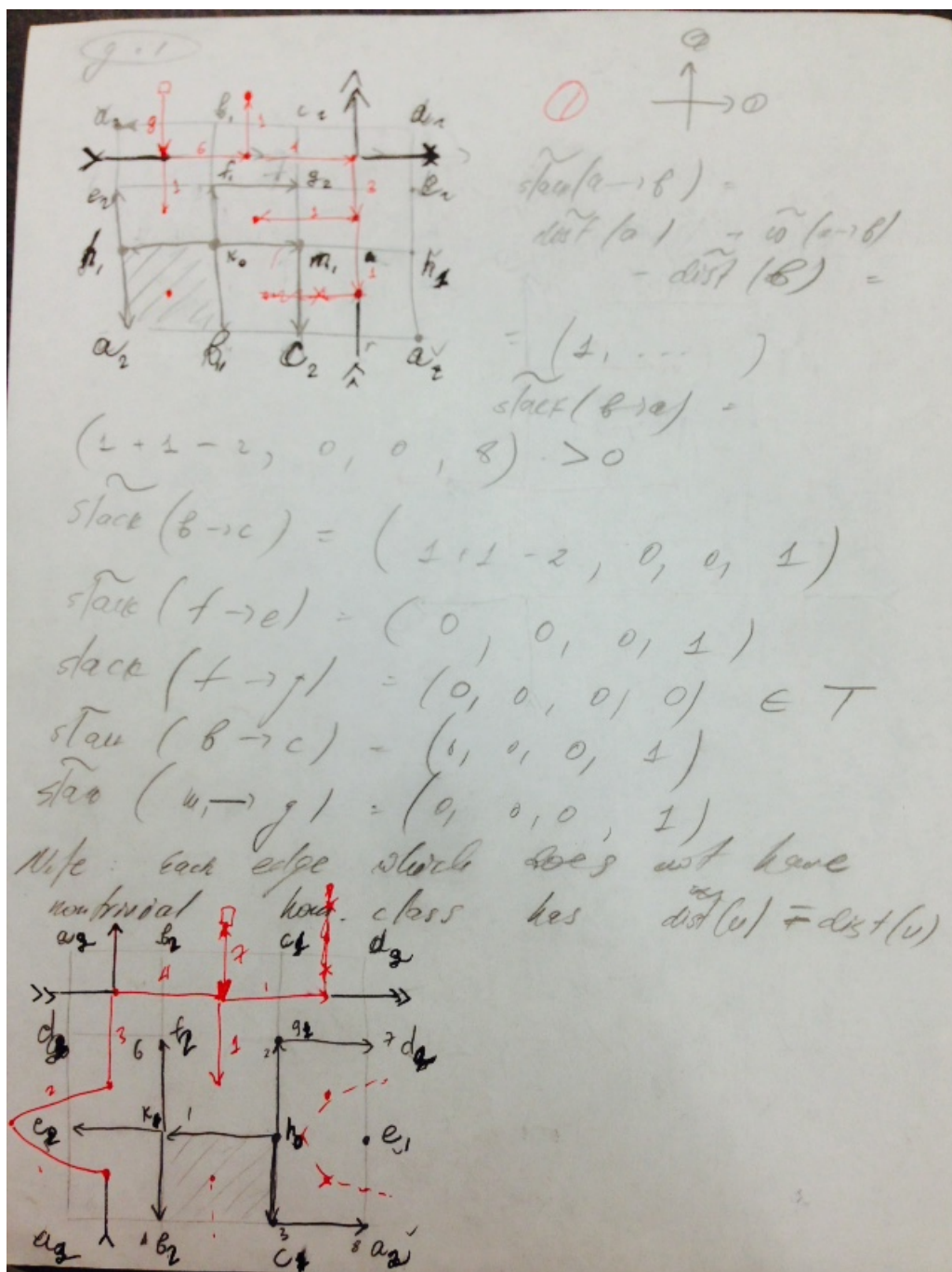


[http : //en.wikipedia.org/wiki/Homology\(mathematics\)](http://en.wikipedia.org/wiki/Homology(mathematics))

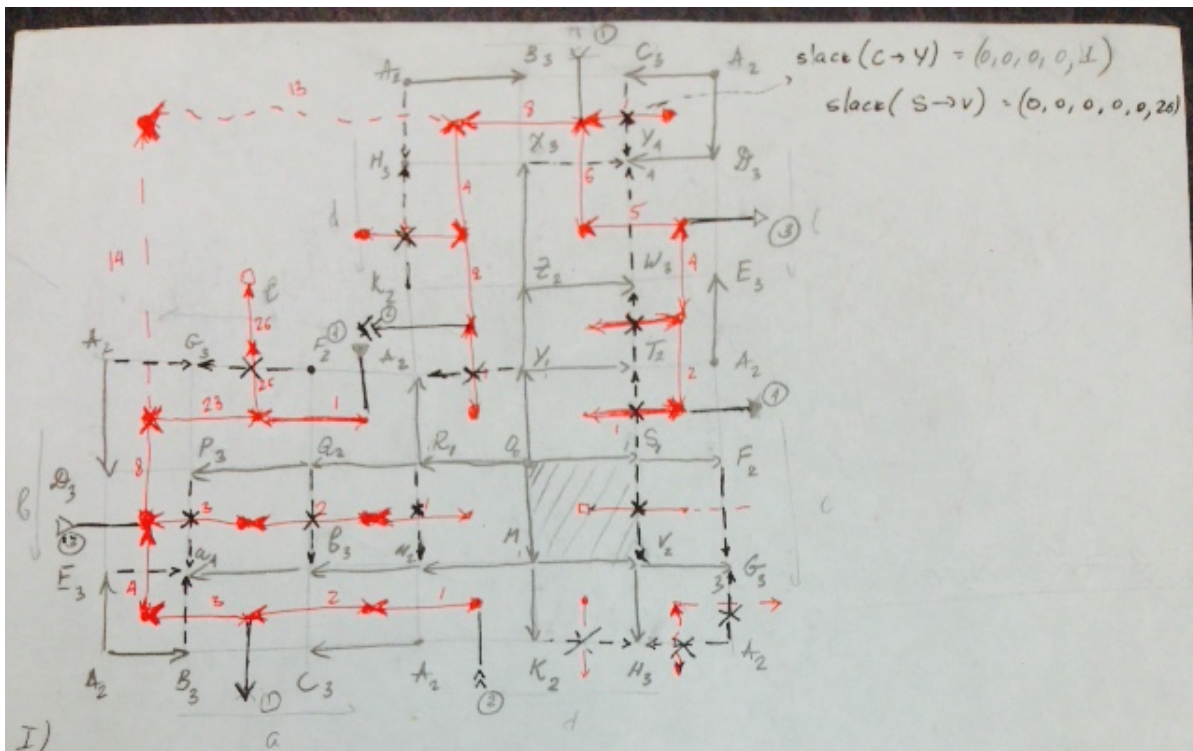
7 Working on examples:

Difference of Holiest Tree and leftmost tree:

On genus $g = 1$ surface:



On genus $g = 2$ surface:



$$s_{k\alpha}(b_5 > c) > 0$$

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

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

for 1, 2, 3, 4 cycles.

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

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

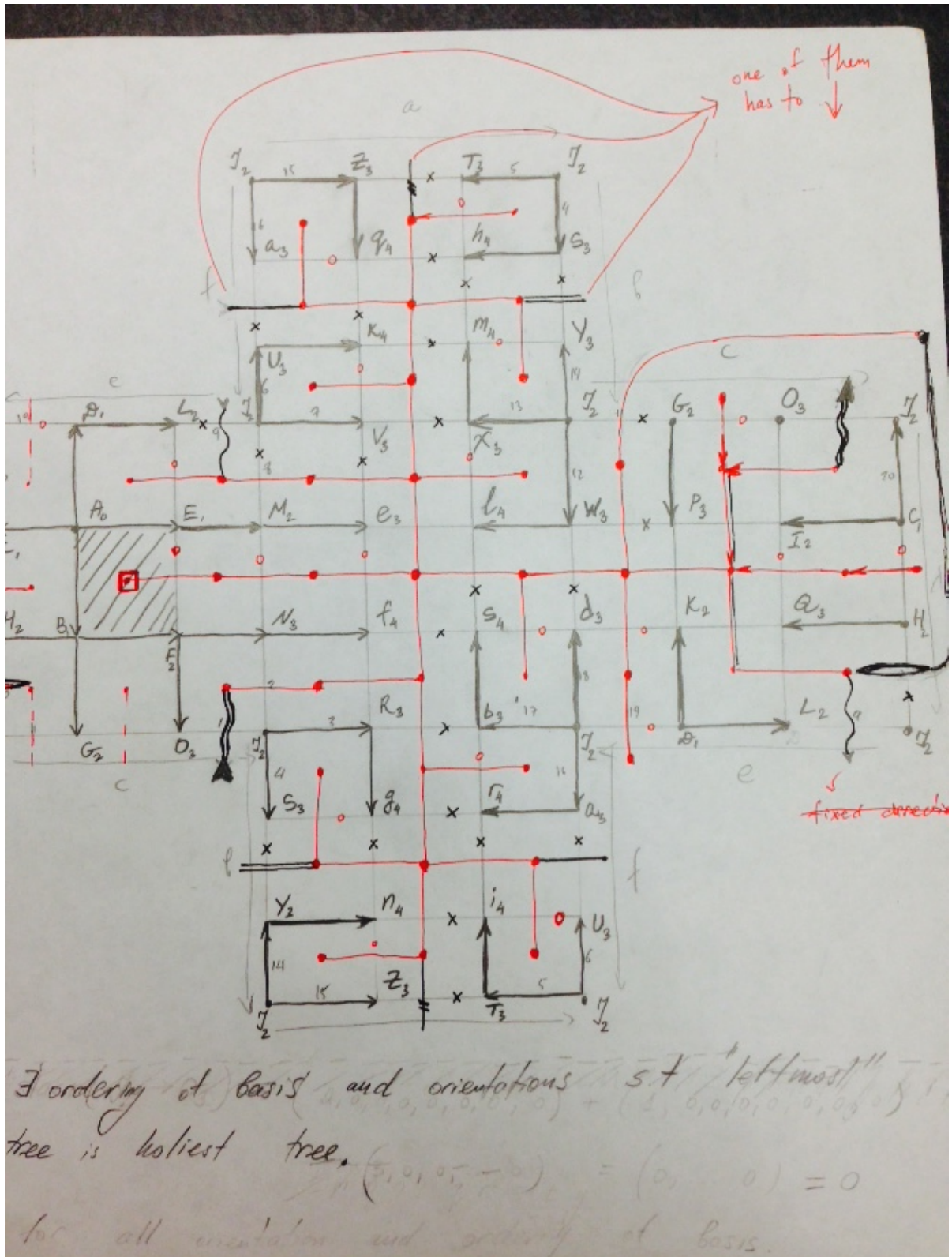
$$\begin{aligned} \text{span} \{A - 6I\} &= (0, 4, -1, -1, -1, 14) \\ \text{span} \{B - 2I\} &= (0, 0, 0, 3, 1, 5) \end{aligned}$$

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

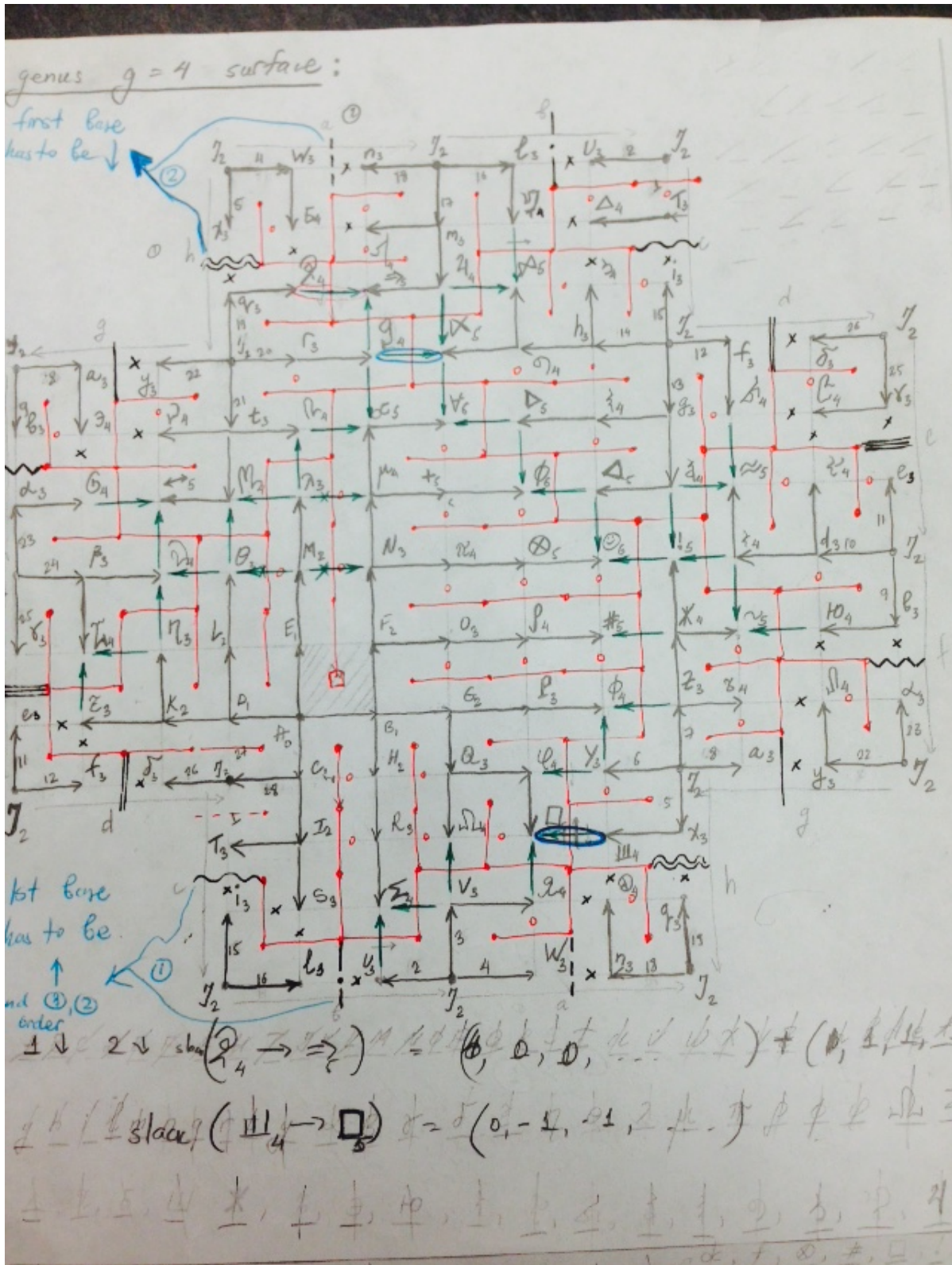
$$\text{slap}(B \rightarrow a) = (0, 1, -1, 0, 0, 3)$$

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

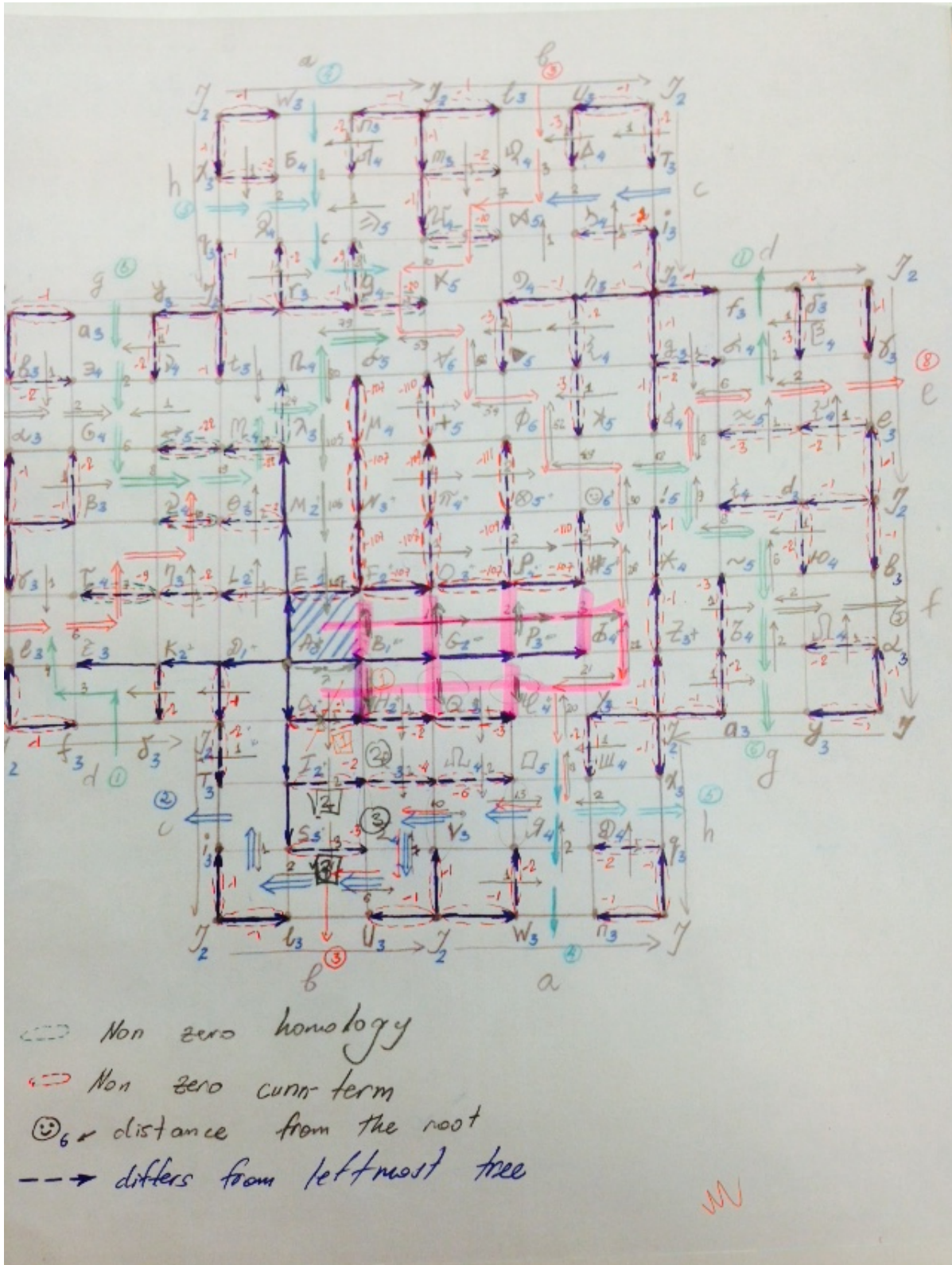
On genus $g = 3$ surface:



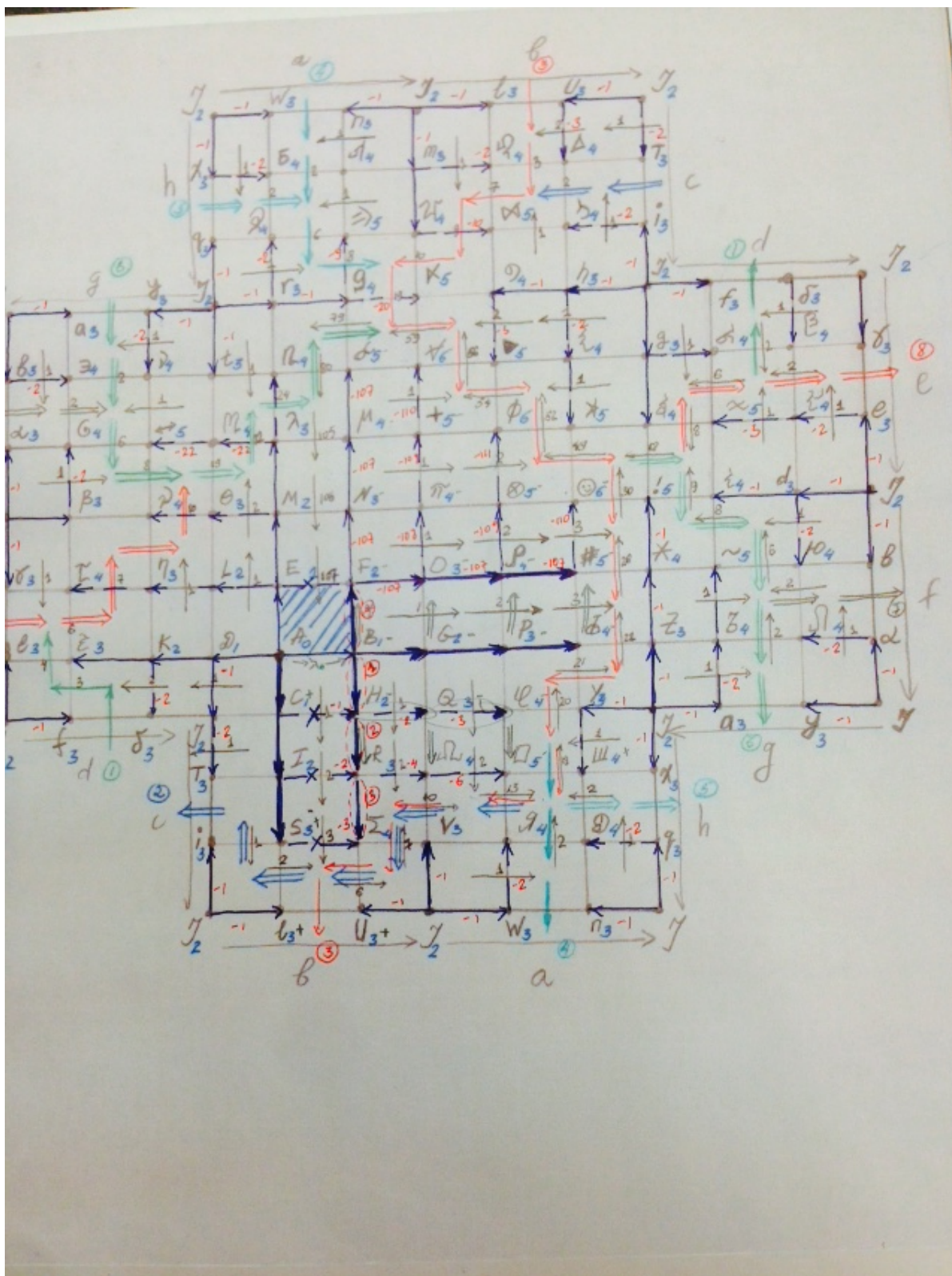
On genus $g = 4$ surface with initial Holy Tree build:



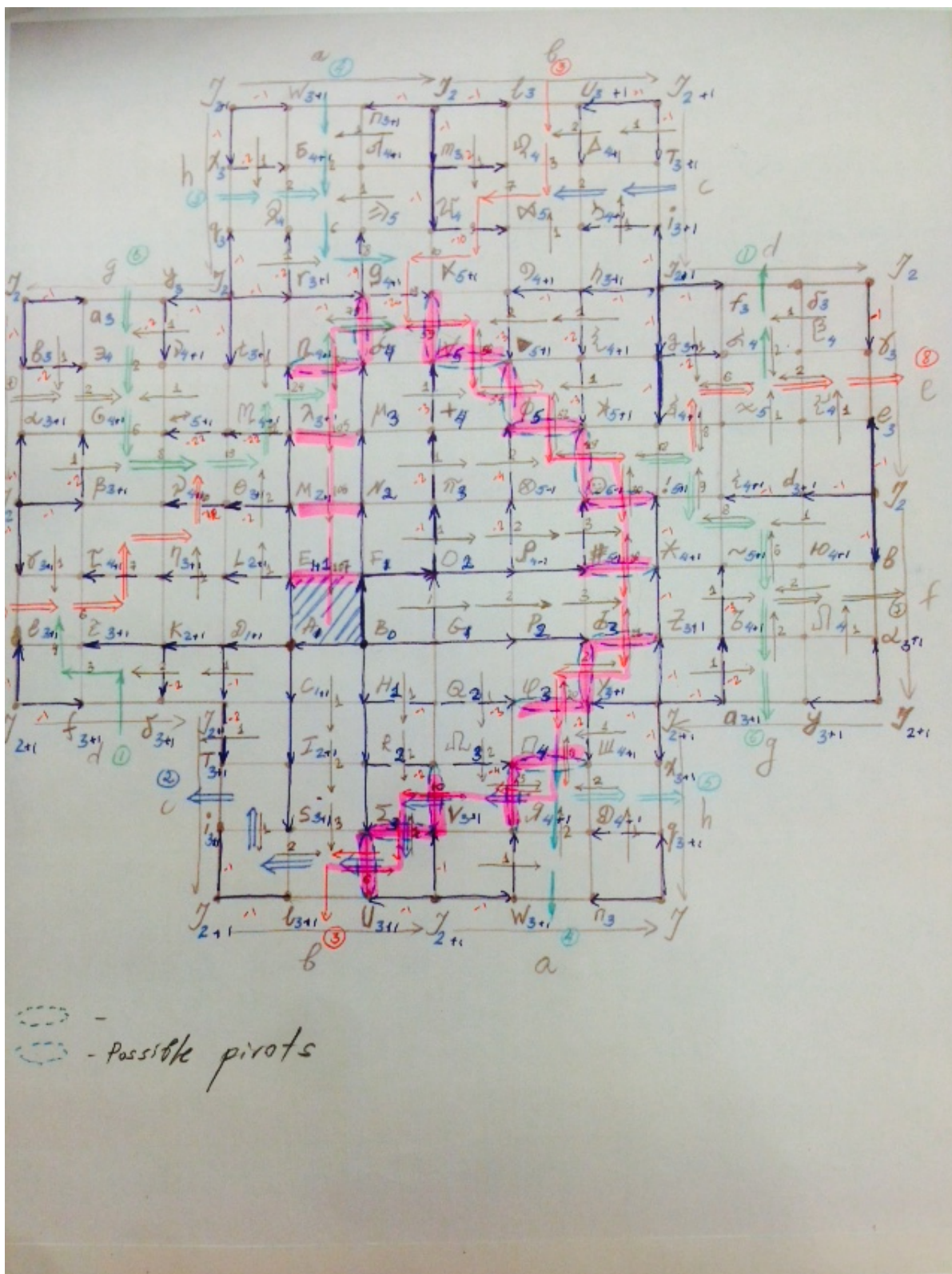
On genus $g = 4$ surface with initial pivot:



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



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



8 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.