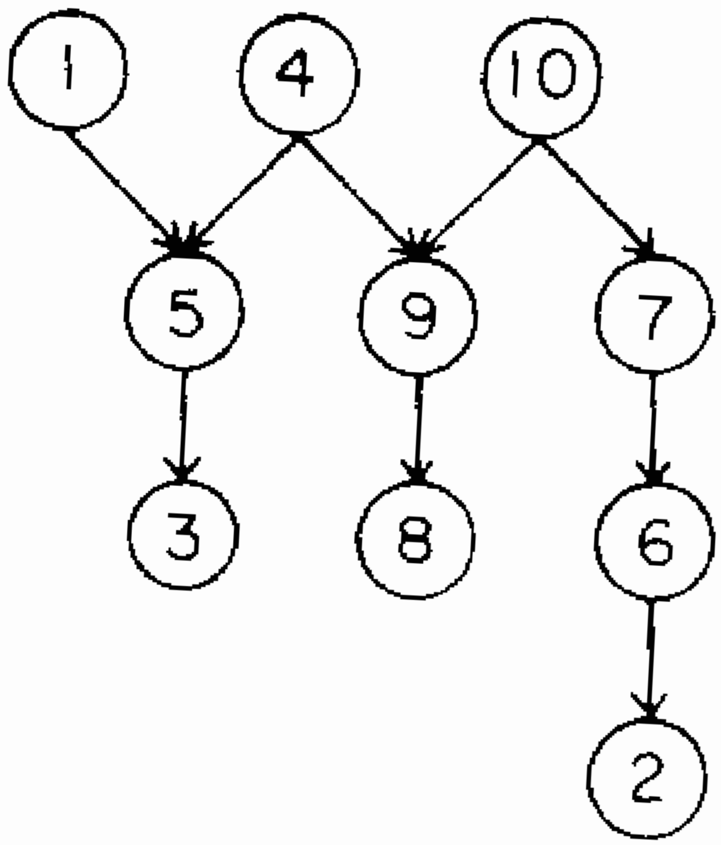
Yoshimura Kuh Channel Routing

Class project by Nicholas Landi and Ajinkya Munge

VCG



VCG

- Vertical constraints are added at every index of the track(Top and bottom input tracks)
- A net that is over another net is called a predecessor of the net it is over
- A descendant of a net is a net that is below another net
- Sweep the input tracks and make a vertical constraint graph (VCG)

Zoning

TABLE I

Column	S(i)	Zone
1	2	1
2	1 2 3	1
3	1 2 3 4 5	1
4	1 2 3 4 5	1
5	1 2 4 5	1
6	2 4 6	2
7	4 6 7	3
8	4 7 8	4
9	4 7 8 9	4
10	7 8 9	4
11	7 9 10	5
12	9 10	5

Zoning

- Zoning is used in place of a horizontal constraint graph (HCG)
- Used to determine which nets are able to merge without the bloat of making an HCG
- The nets are first assigned to different columns based on the top and bottom rails
- Set of nets in every columns are checked if they are a subset of nets in next column or if the set of next column are a subset of nets in column under consideration
- The zone is union of these columns sets if the previous condition is satisfied
- If this condition is not satisfied, new zone has to be created

Merging

```
        procedure Algorithm #1 (z_s, z_t)
        begin;
a1:     L = { };
a2:     for z = z_s to z_t do;
    begin;
a3:             L = L + {nets which terminate at zone z};
a4:             R = {nets which begin at zone z + 1};
a5:             merge L and R so as to minimize the increase
                of the longest path length in the vertical
                constraint graph;
a6:             L = L - {n_1, n_2, ···}, where n_j is a net
                merged at step a5;
        end;
    end;
```

Heuristic Methods

(1) for $m \in Q$

$$f(m) = C_\infty * \{u(m) + d(m)\} + \max\{u(m), d(m)\},$$

$$C_\infty \gg 1$$

(2) for $n \in P$, $m \in Q$

$$g(n, m) = C_\infty * h(n, m) - \{\sqrt{u(m) * u(n)} + \sqrt{d(m) * d(n)}\}$$

where

$$h(n, m) = \max\{u(n), u(m)\} + \max\{d(n), d(m)\} - \max\{u(n) + d(n), u(m) + d(m)\}$$

—the increase of the longest path length passing through n or m, by merging of n and m.

Merging Algorithm

```
given P, Q;
    begin;
a1:     while Q is not empty do;
        begin;
a2:         among Q, find m* which maximizes f(m);
a3:         among P, find n* which minimizes g(n, m*),
            and which is neither ancestor nor descendent
            of m*;
a4:         merge n* and m*;
a5:         remove n* and m* from P and Q,
            respectively;
        end;
    end;
```

- f(m) and g(n,m) decide which two nets(n,m) are to be merged
- u(x) = longest path of x from s
- d(x) = longest path of x from t
- m is chosen in f(m) such that lies on the longest path(from source or sink) before merging
- n is chosen in g(n,m) such that the increase of the longest path after merging is minimum
- Tiebreaker is chosen on which pair (n,m) satisfies the 2 heuristic conditions set by h(n,m)

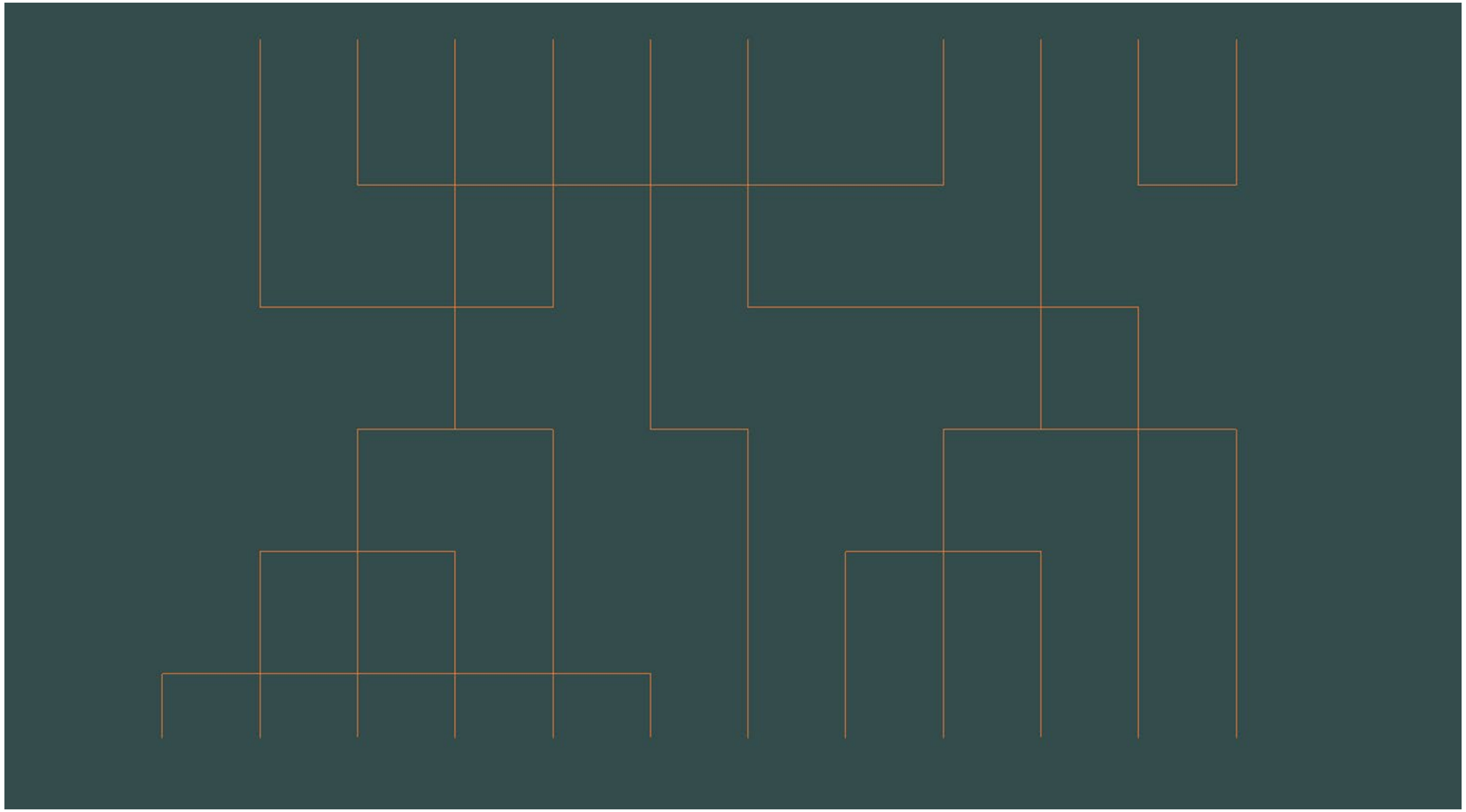
Netlist	NS	DT	TD	Merged nets	Total Height	Avg Program Runtime (ms)
DR1	10	0	0	9	5	96
DR1(DL)	10	0	12	11	6	144
DR2	20	1	1	4	19	210
DR2(DL)	20	1	46	41	19	1563
DR3	30	2	2	14	24	2023
DR3(DL)	30	2	86	85	28	23394
DR4	60	7	7	36	47	171169
DR4(DL)	60	7	156	148	49	2545258
DR5	60	7	7	36	47	172969
DR5(DL)	60	7	156	154	49	2552258
DR7	20	0	0	0	20	81
DR7(DL)	20	0	50	41	23	987

DL: Doglegged DT: Doglegs added to remove cyclic conflicts, TD: Total Doglegs added NS: Netlist Size

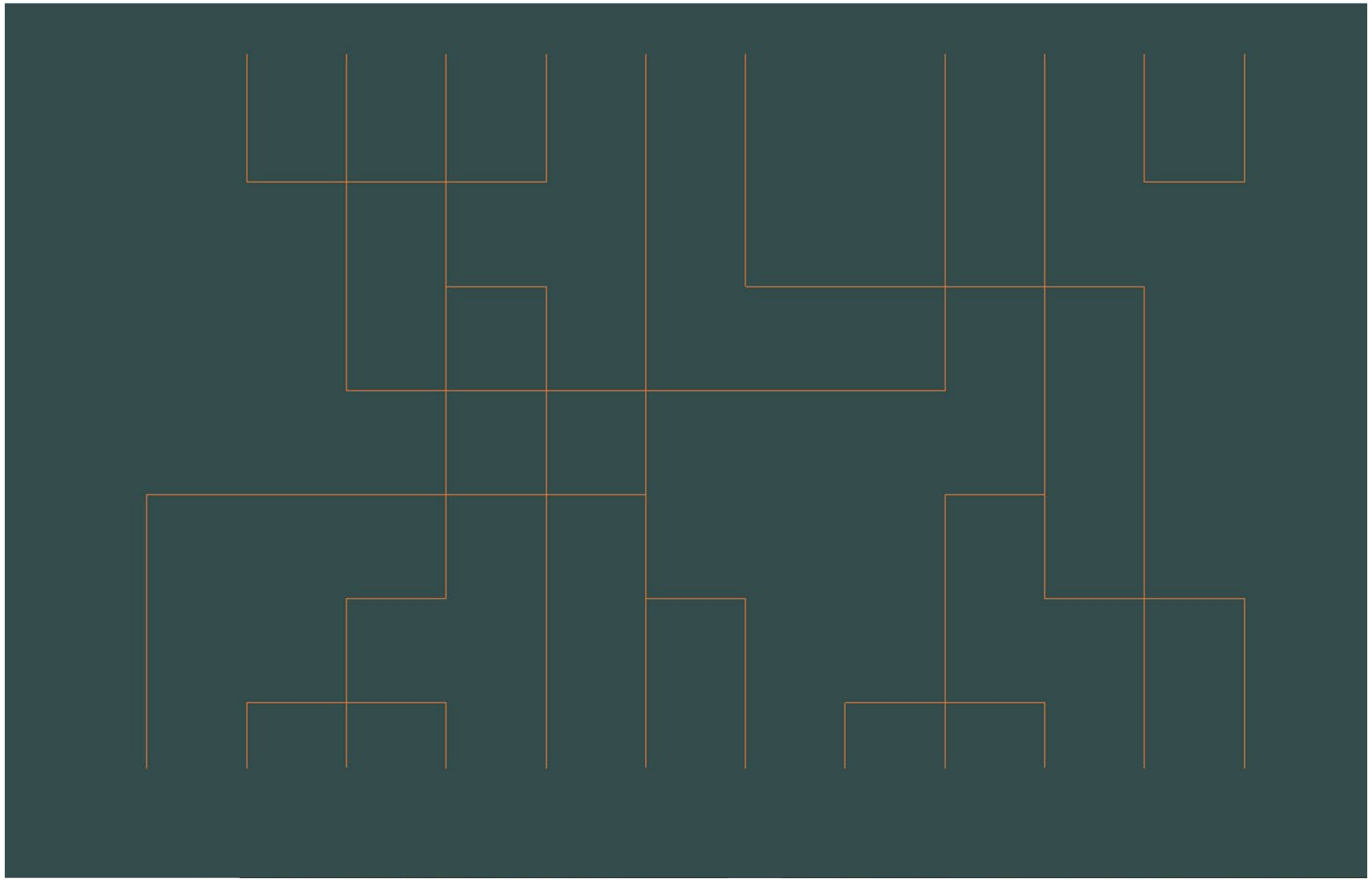
Netlist	NS	DT	TD	Merged nets	Total Height	Avg Program Runtime (ms)
DR8	40	0	0	15	31	375
DR8(DL)	40	0	68	49	36	1314
DR9	80	0	0	40	57	1432
DR9(DL)	80	0	136	84	84	5063
DR10	78	0	0	13	71	833
DR10(DL)	78	0	135	84	81	5666

DL: Doglegged DT: Doglegs added to remove cyclic conflicts, TD: Total Doglegs added NS: Netlist Size

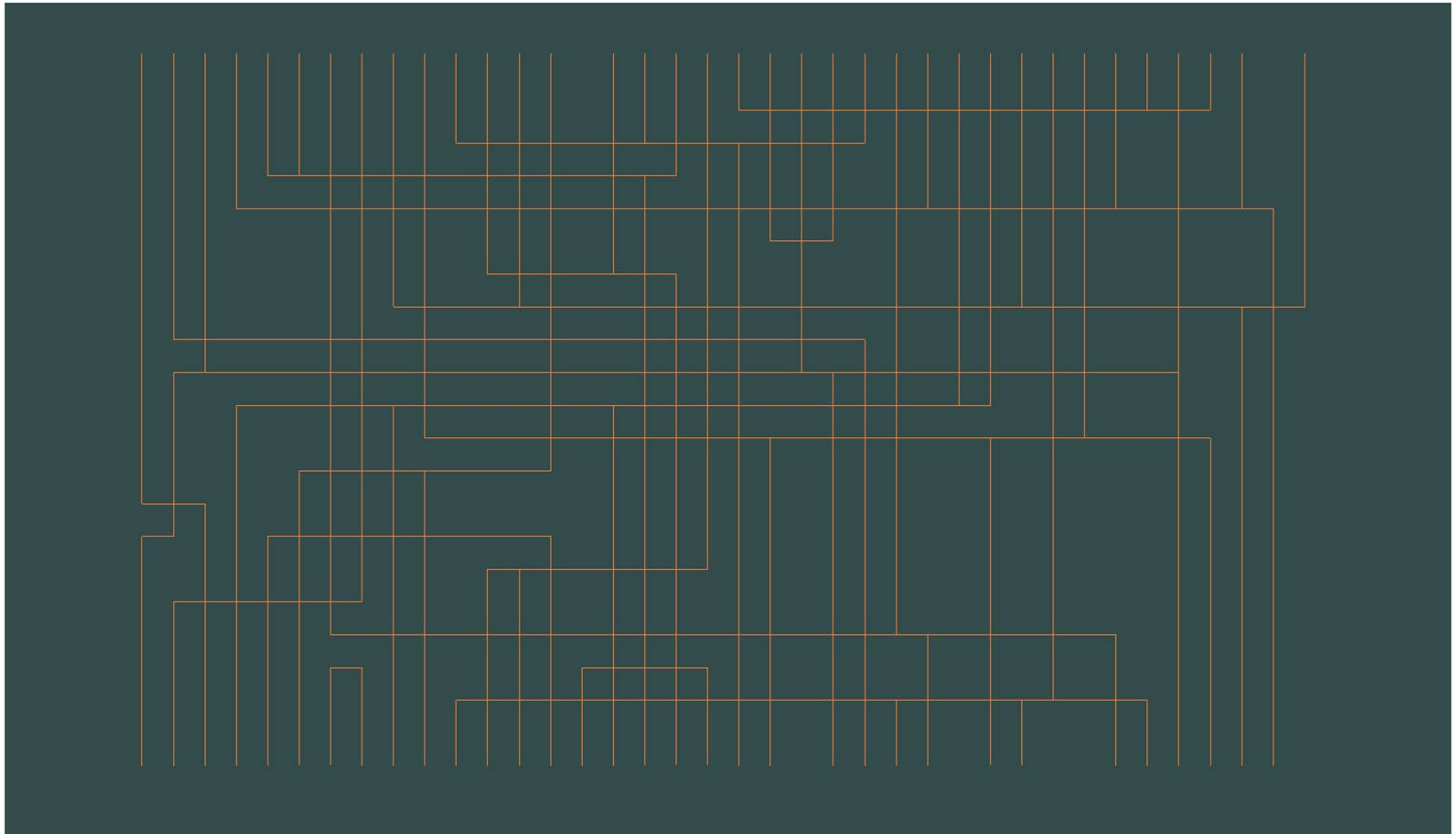
DR1 with minimum Doglegs



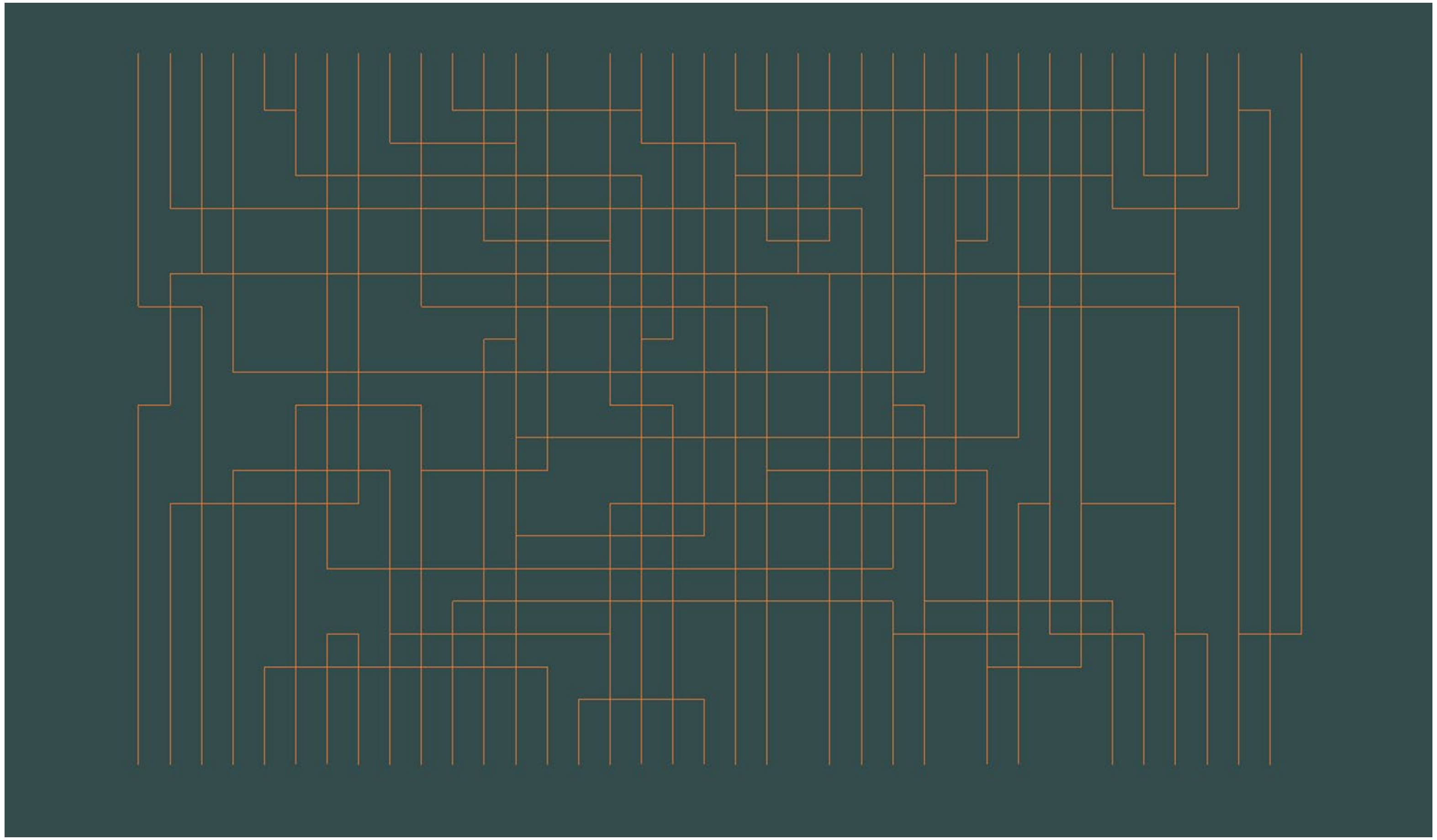
DR1 with Doglegs at terminal indexes



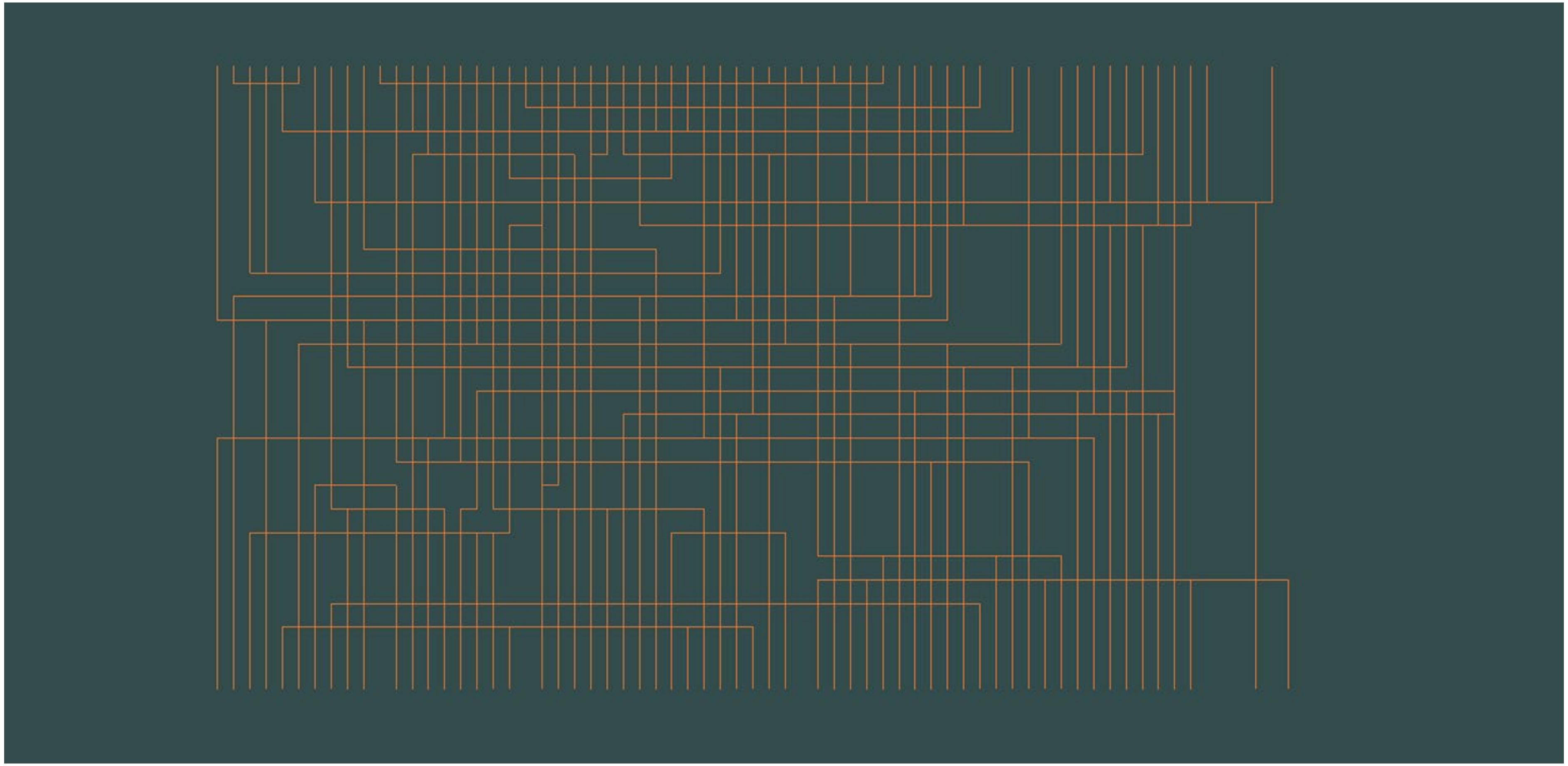
DR2 with minimum Doglegs



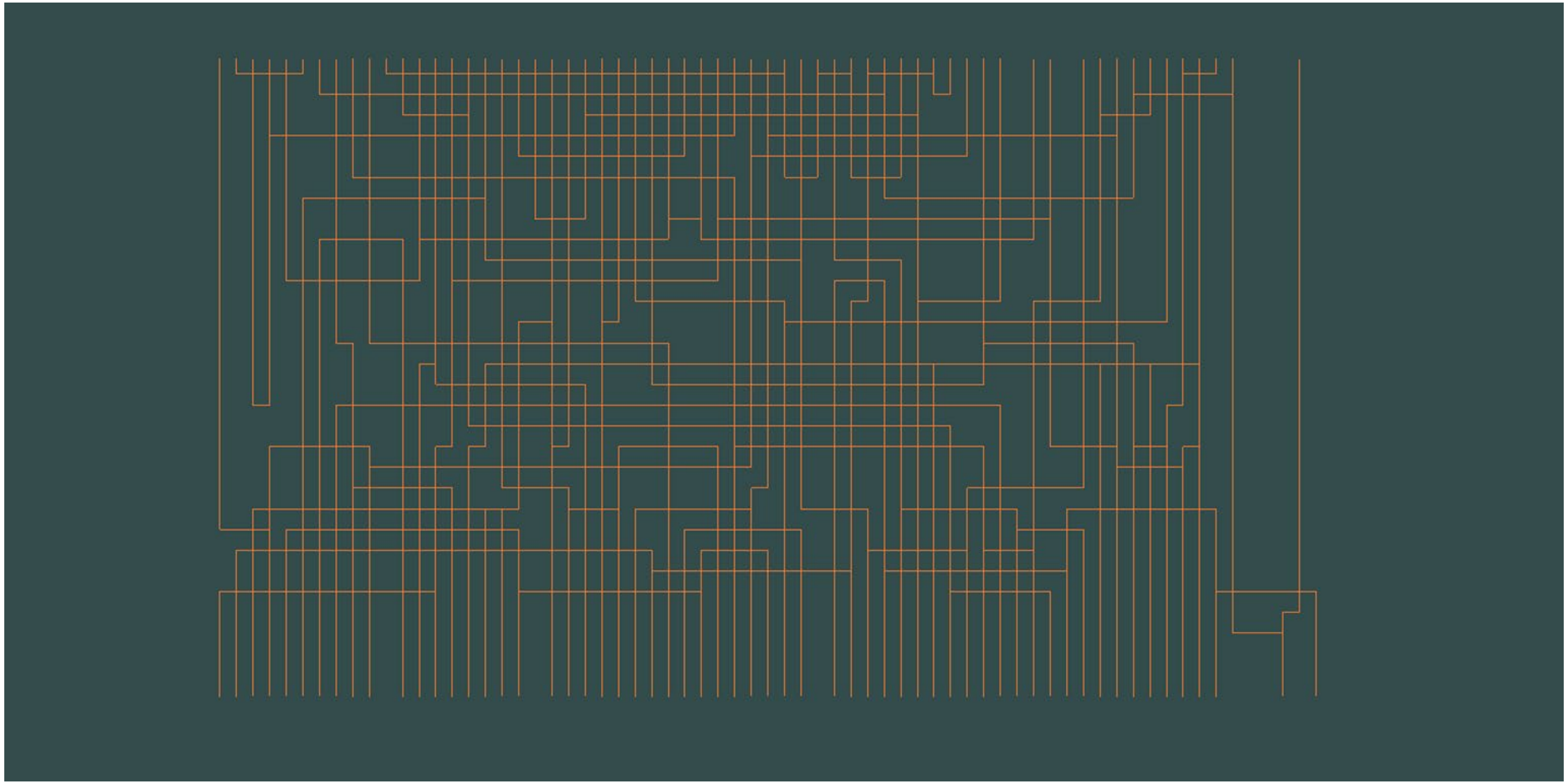
DR2 with Doglegs at terminal indexes



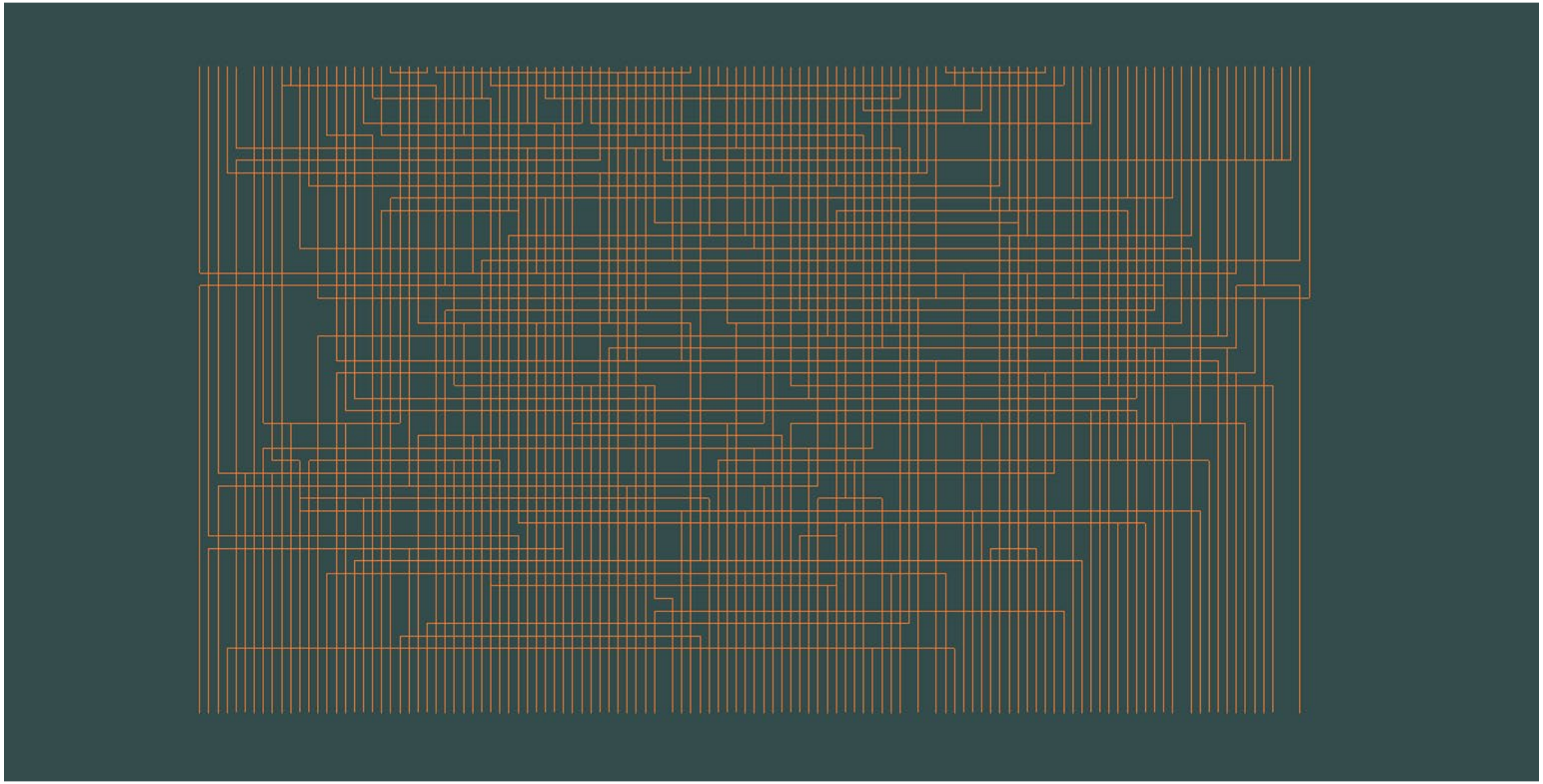
DR3 with minimum Doglegs



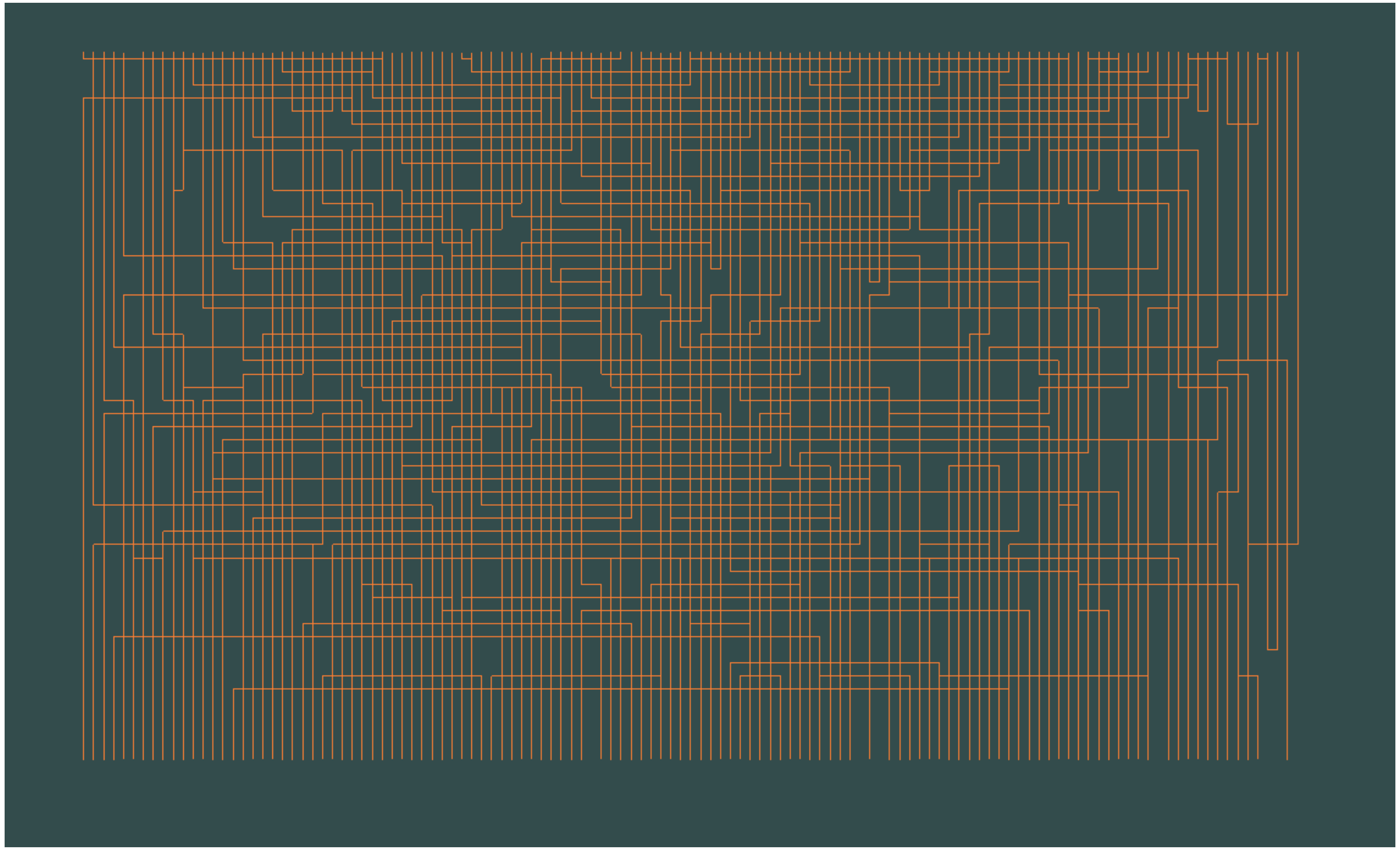
DR3 with Doglegs at terminal indexes



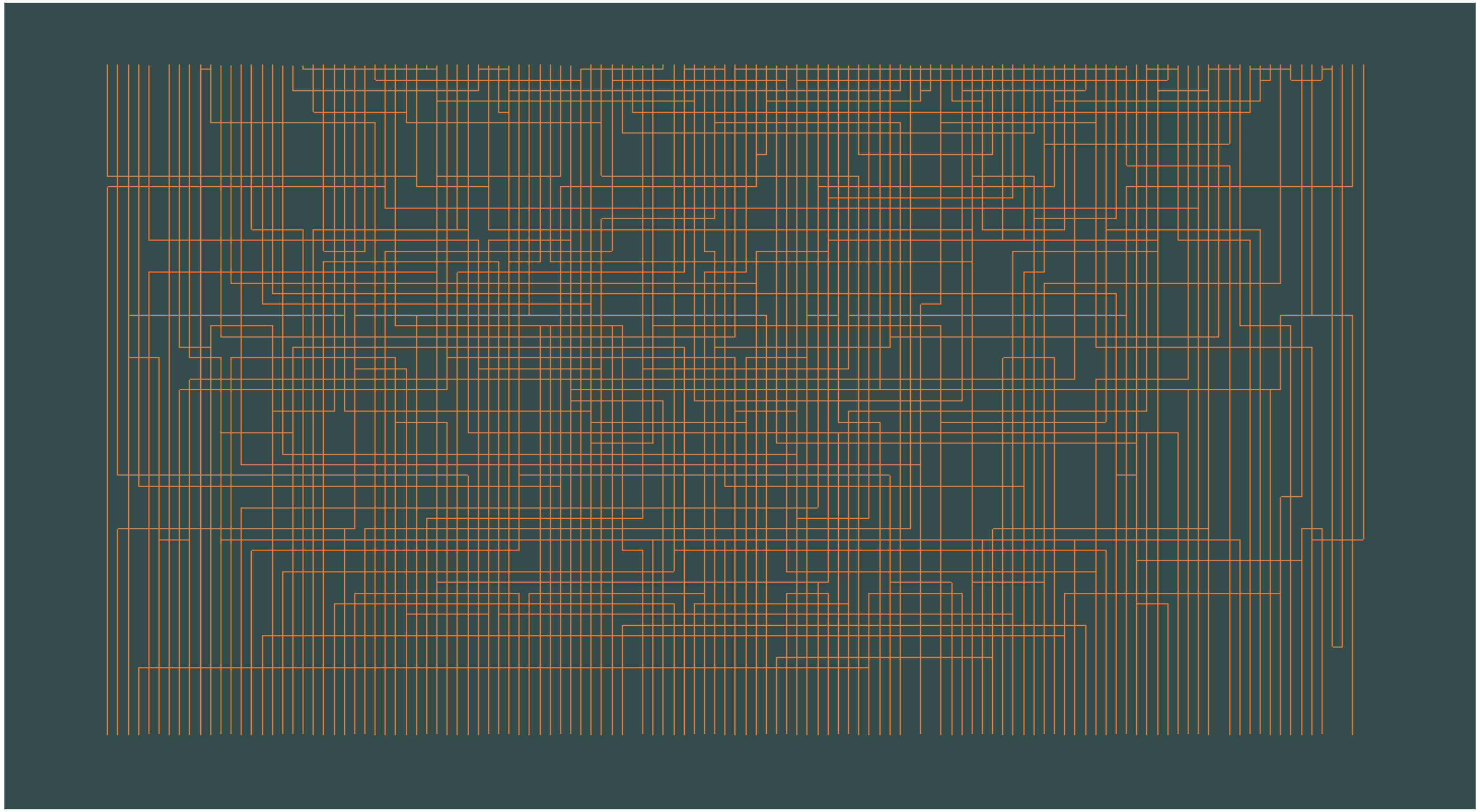
DR4 with minimum Doglegs



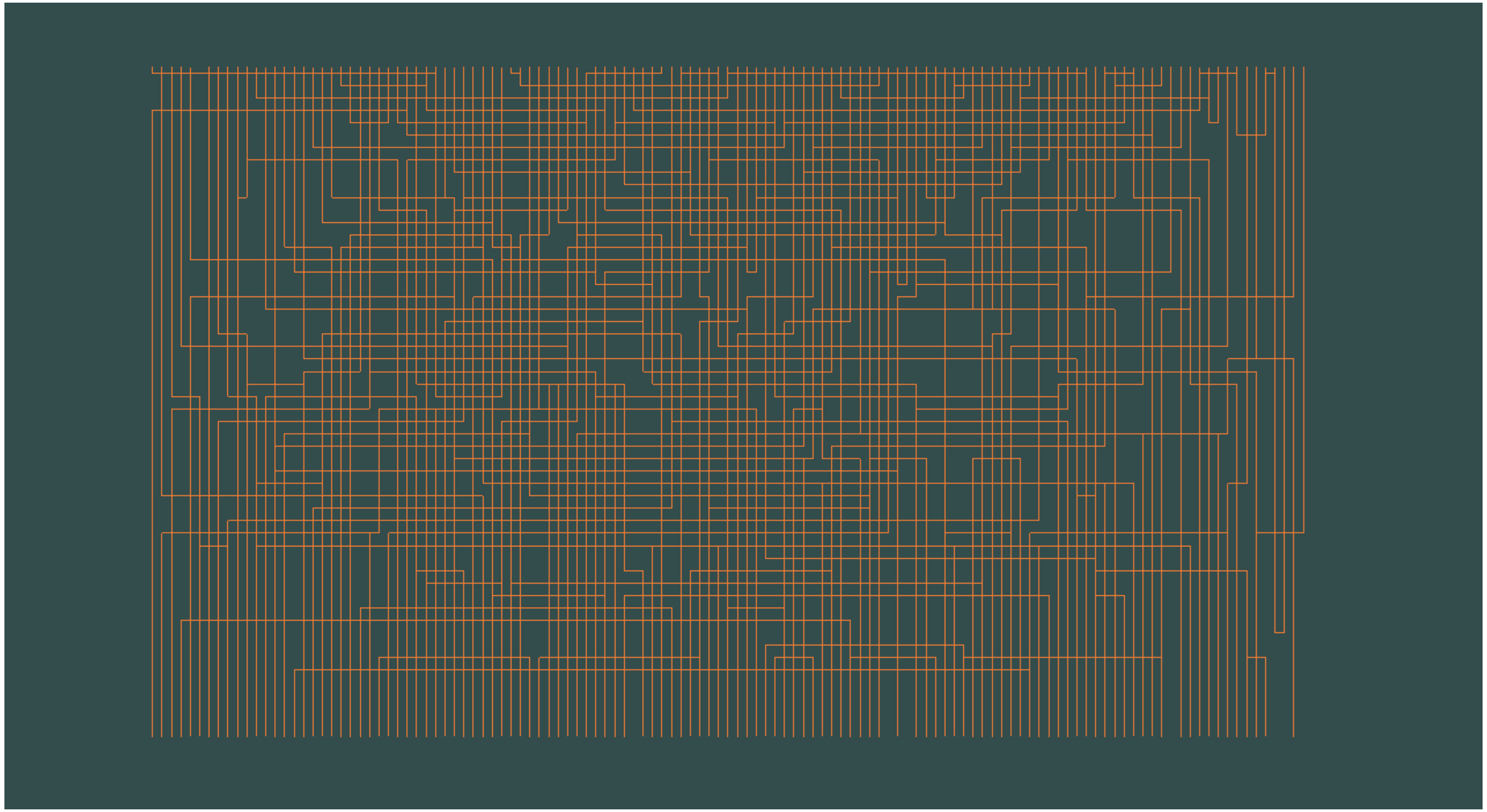
DR4 with Doglegs at terminal indexes



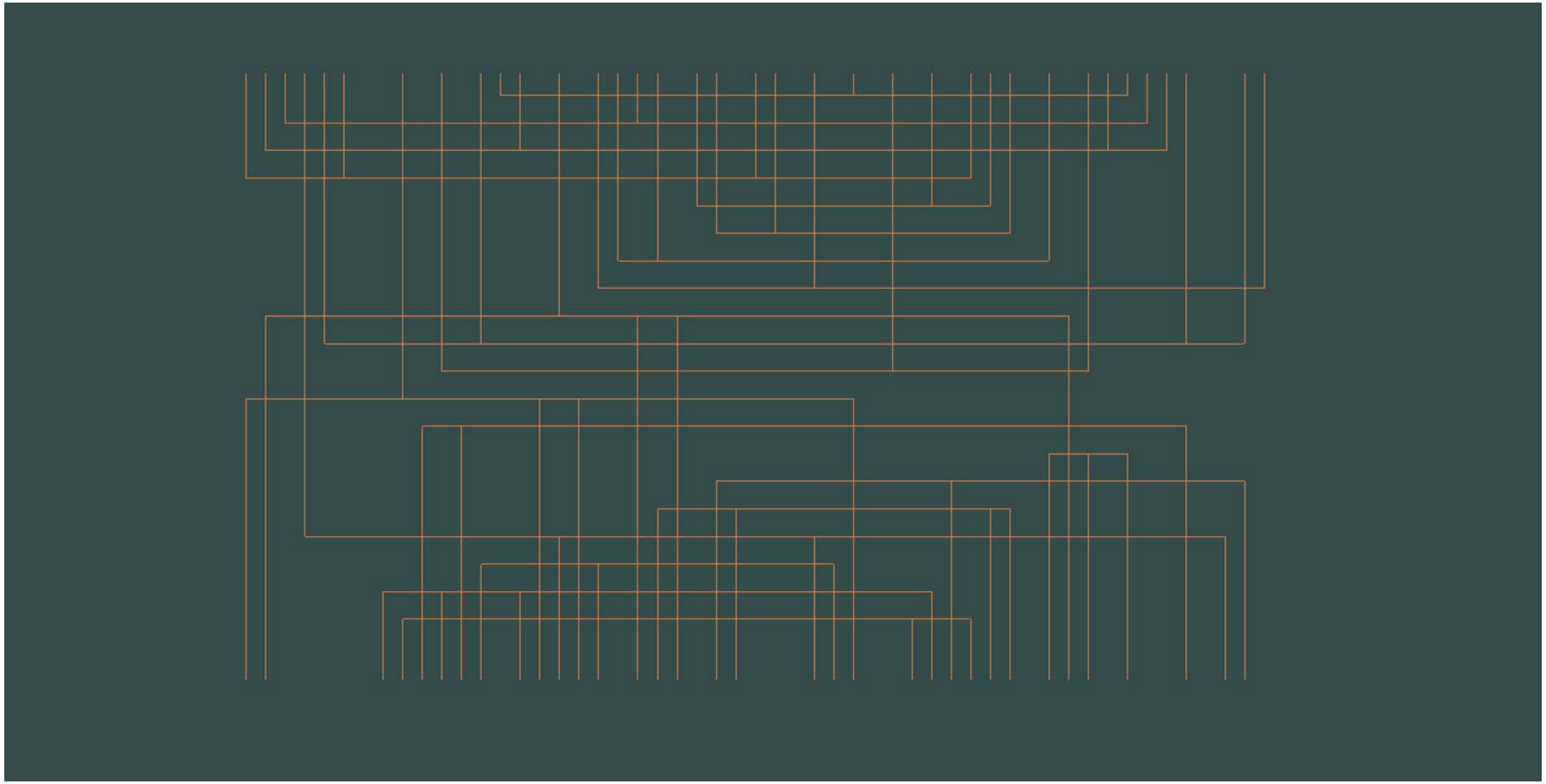
DR5 with minimum Doglegs



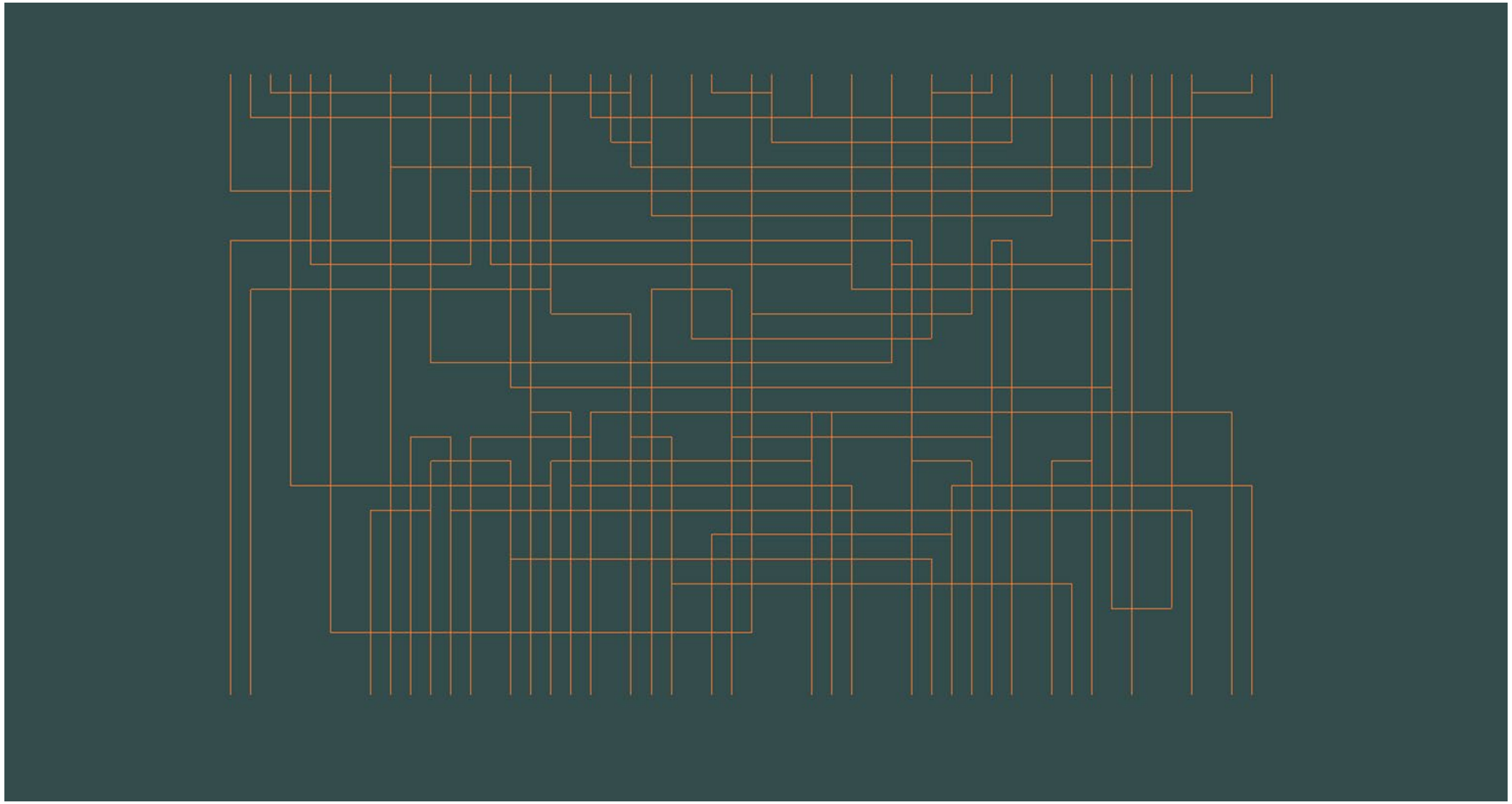
DR5 with Doglegs at terminal indexes



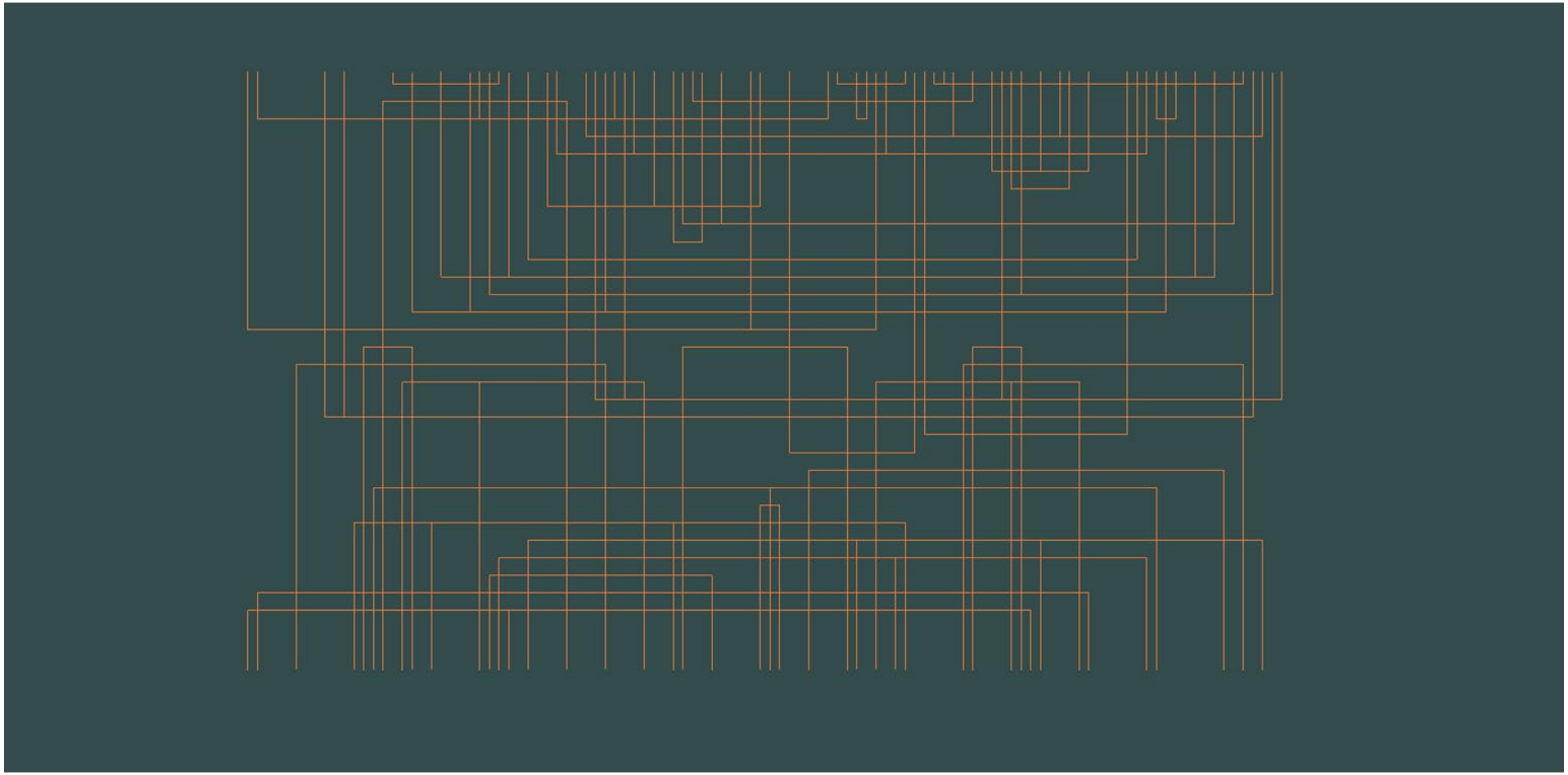
DR7 with minimum Doglegs



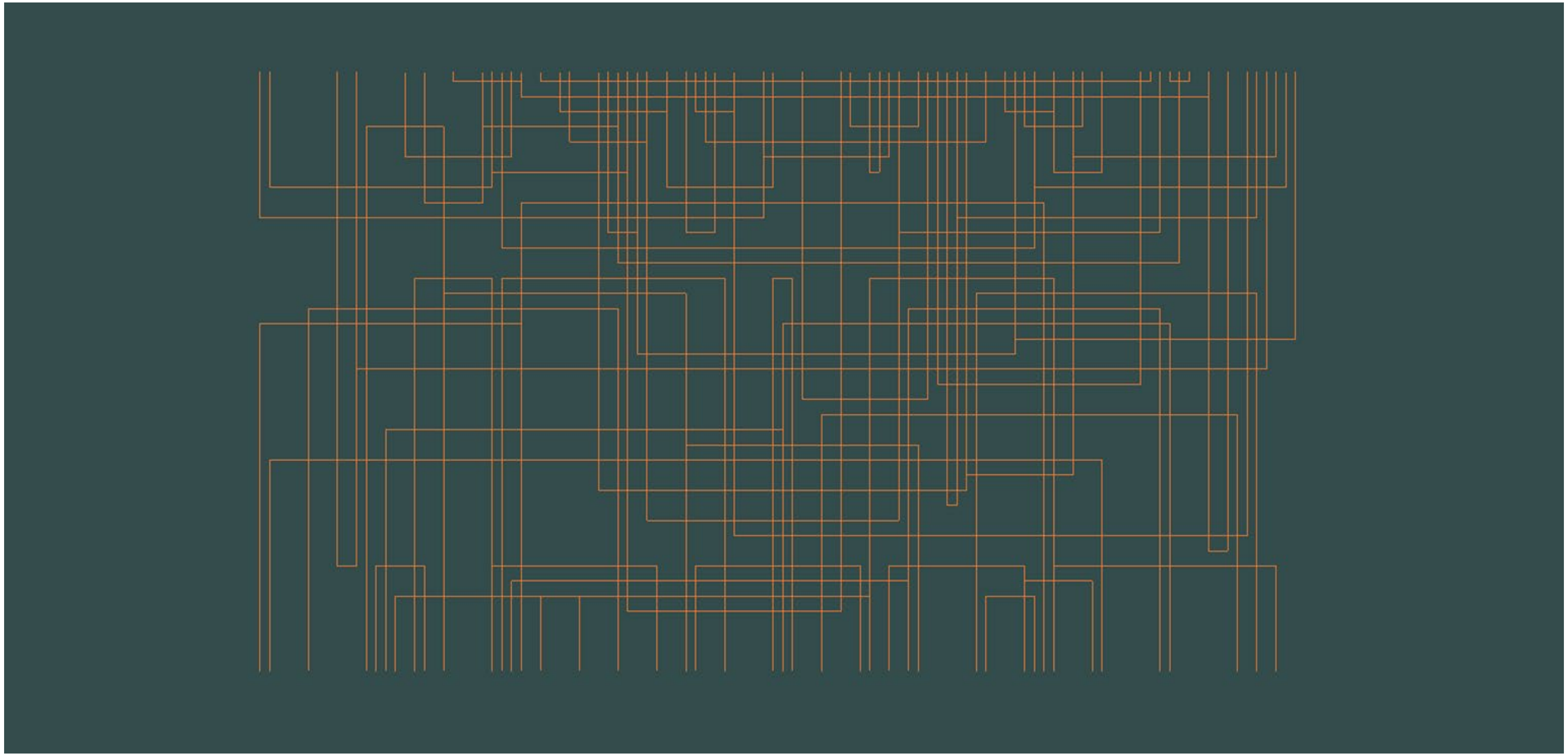
DR7 with Doglegs at terminal indexes



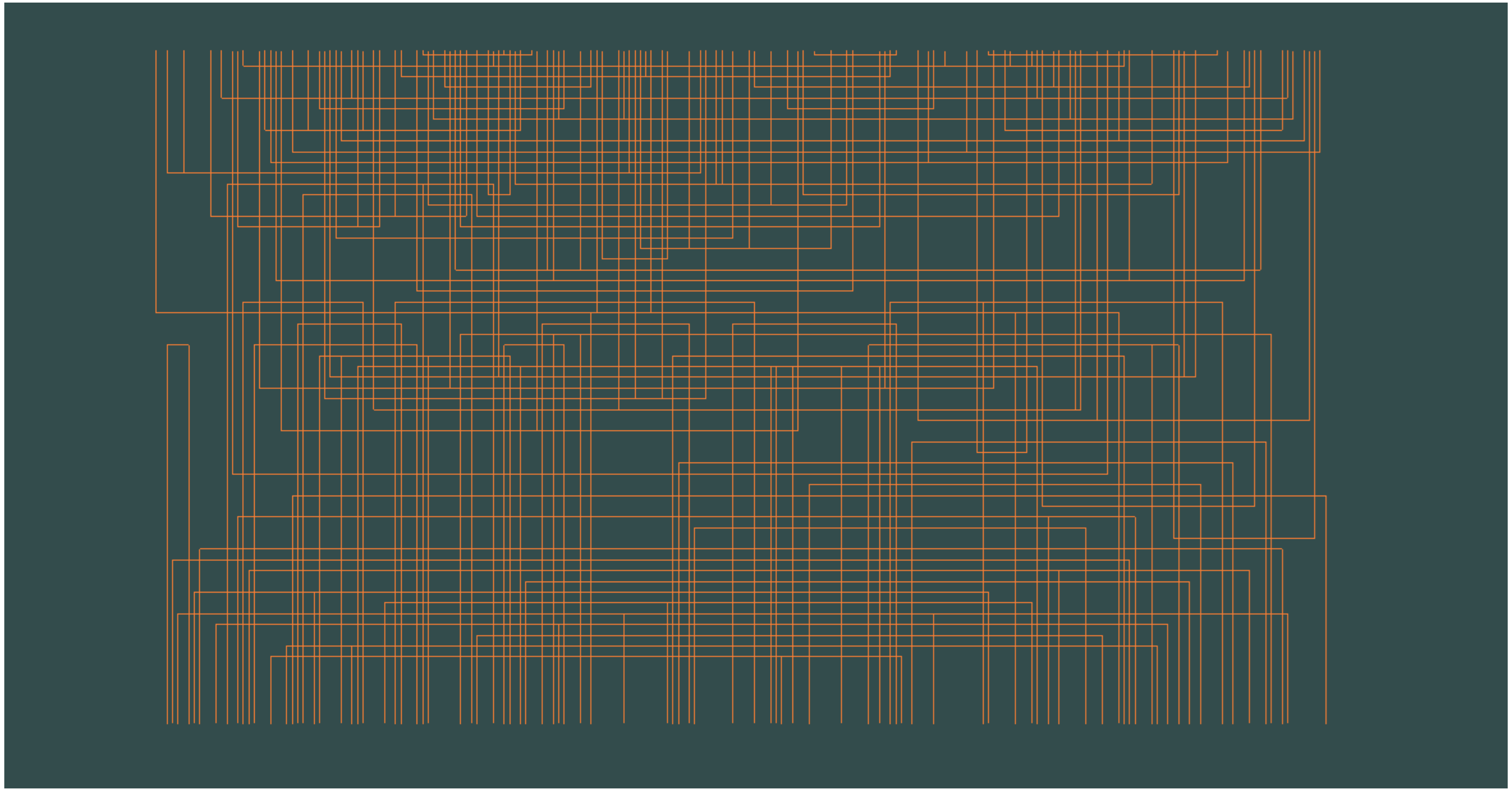
DR8 with minimum Doglegs



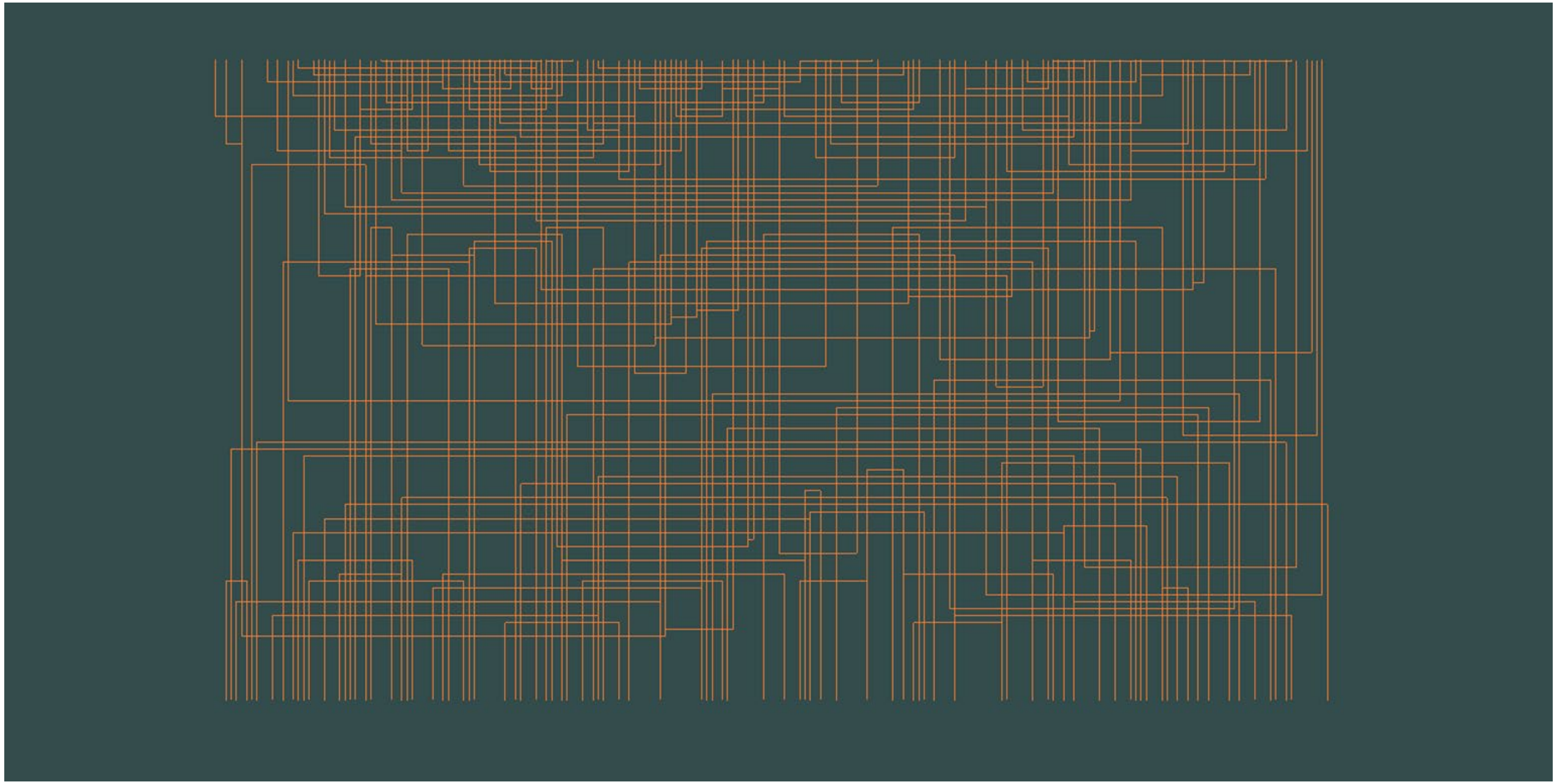
DR8 with Doglegs at terminal indexes



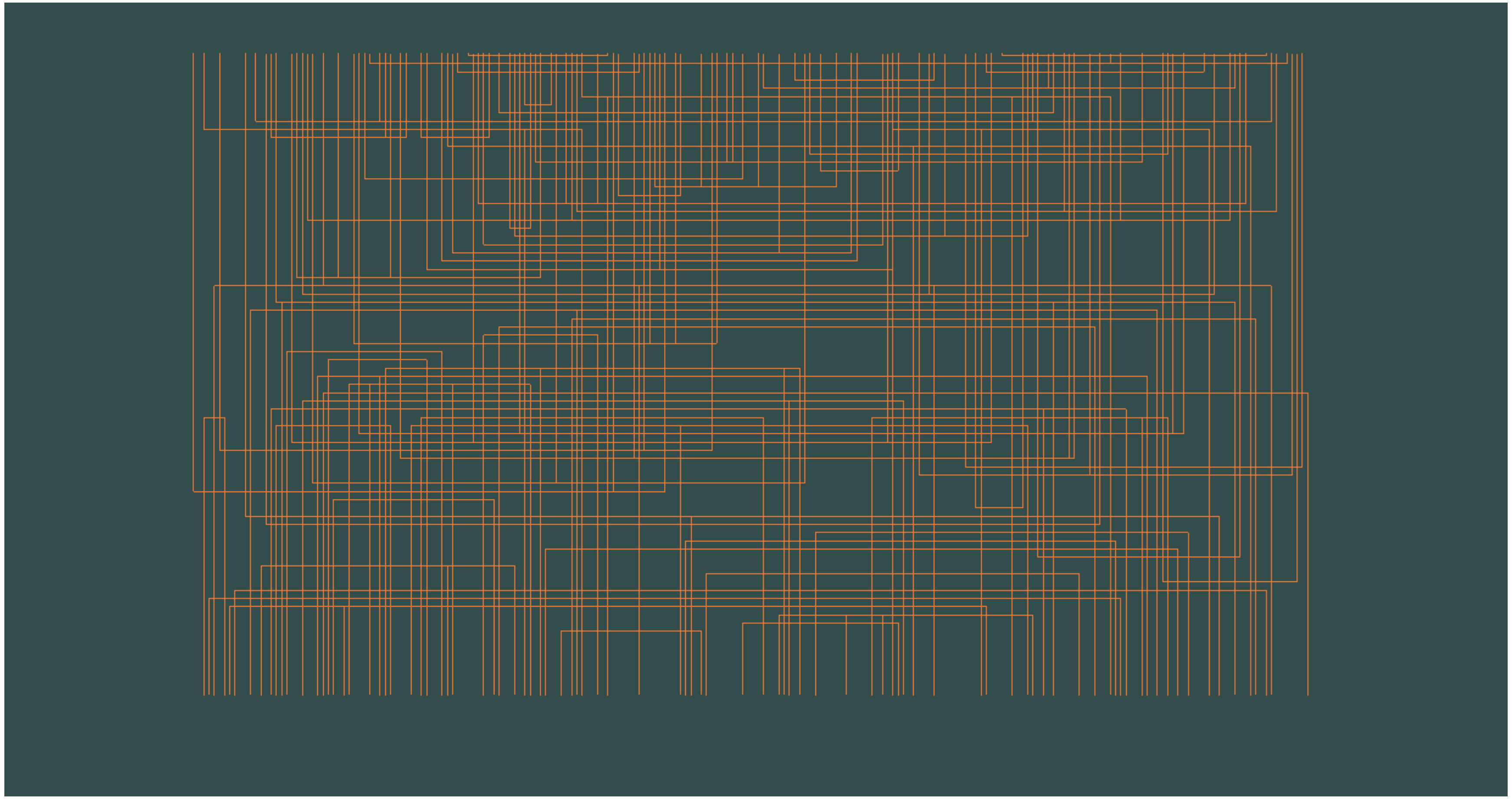
DR9 with minimum Doglegs



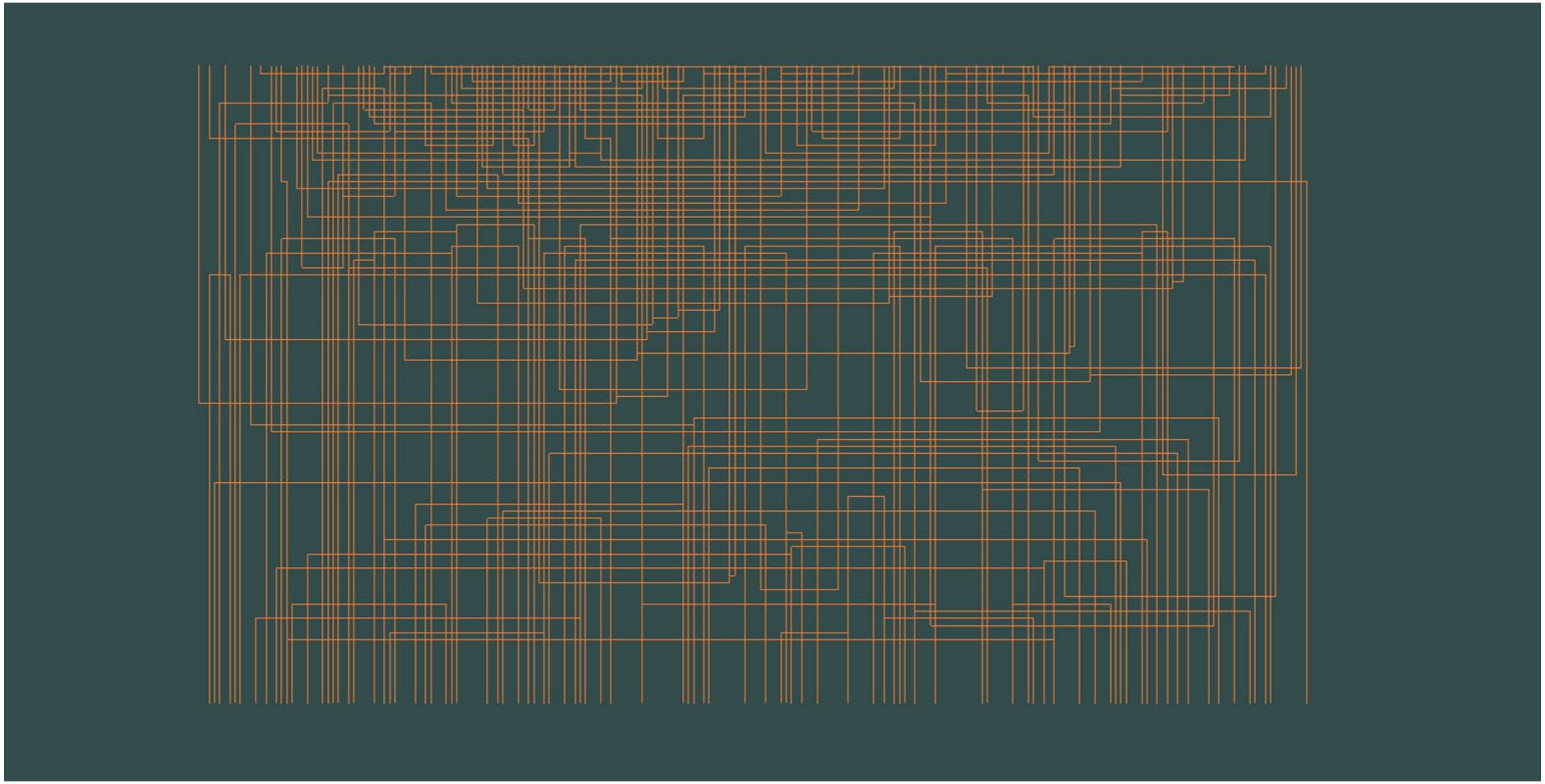
DR9 with Doglegs at terminal indexes



DR10 with minimum Doglegs



DR10 with Doglegs at terminal indexes



Concluding Thoughts and Improvements

- A more intelligent way to dogleg would be to look for the best decrease in the distance to source and sink, as opposed to our method of doglegging at every possible terminal.
- Excessive doglegging, as a result of terminal doglegging, decreases solution quality drastically while increasing run-time exponentially.
- There are some cases where +1 doglegging fails to remove a cycle in the VCG and causes a crash, this can be improved by adding conditions to check if the dogleg insertion will actually remove the cycle.