Gordian

Ismael Youssef
Pranay Jaggi
Spring 2025

Problem Formulation

- **Problem:** Modern circuit design utilizes millions of cells. This makes manual placement of circuits impossible.

- **Solution:** An automated mathematical based placement of circuitry.

- **Method:** Using the gordian placement algorithm for automatic placement.

Algorithm Discussion

- **Gordian Algorithm (1991):**
  - Analytical Placement solution for VLSI placement problem.
  - Named after Gordian Knot to shows it simplicity to solve a complex problem.

- **Core Methodology:**
  - Transforms placement into a quadratic optimization problem.
  - Begins with a global solution that allows initial overlaps
  - Progressively refines placement through repeated partitioning,
  - Each iteration creates smaller, more manageable sub-regions, until solution is reached

Implementation

- **Programming language:** Python
- **Libraries Used:**
  - `numpy, pandas: For file loading and data management`
  - `matplotlib: For GUI visualization`
  - `quadprog, osqp,cvxopt: Different Quadratic Solvers`
  - `scipy: For sparse matrix handling`

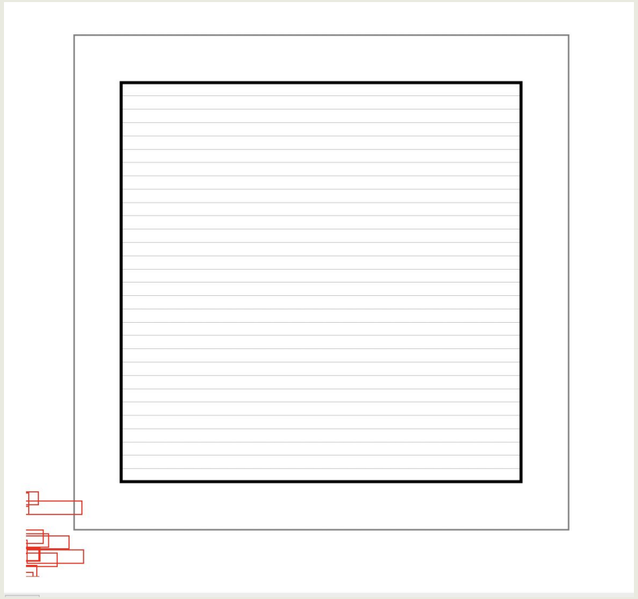
Implementation (Cont.)

- **`quadprog` Python library used as base QP solver**
  - No sparse matrix optimizations applied at this level
  - Runtime is very high so decided to explore other solvers (Explained later).
- **Recursive Bipartitioning**
  - At odd levels a vertical cut is made
  - At even levels a horizontal cut is made
  - Max levels is $\log_2(N)$ where N is number of cells
  - This ensures each partition has roughly half the cells and improves placement locality
  - Partitioning goes down until each partition has at most 2 nodes
  - After each cut, center of gravity constraint is computed for each new partition

Sparse Matrix Handling

- Base solver (quadprog) is very slow (see results)
  - To solve this a sparse matrix solution was used.
- Implemented **two** sparse QP solvers:
  - `osqp library` solver
    - Easy syntax, handles sparse matrices directly, flexible for additional constraints
    - To use in code: “-s cvxpy” (calls cvxpy with osqp as the solver)
  - `cvxopt library` solver
    - More accurate for dense problems, control over constraint matrix formats
    - To use in code: “-s cvxopt”

Sparse Matrix Handling Note / Issue Encountered



- These two sparse solvers give interesting results for Level 0.
  - Placement was out of bounds (see right)
- To counteract this, used `quadprog` for level 0.
  - Results looked more correct, however this increases runtime as shown in results table later on
- We have shown the placement progression through each level and the wire connections at each level
  - This increases runtime by a decent amount
  - Without these visualizations, the tool runs much faster

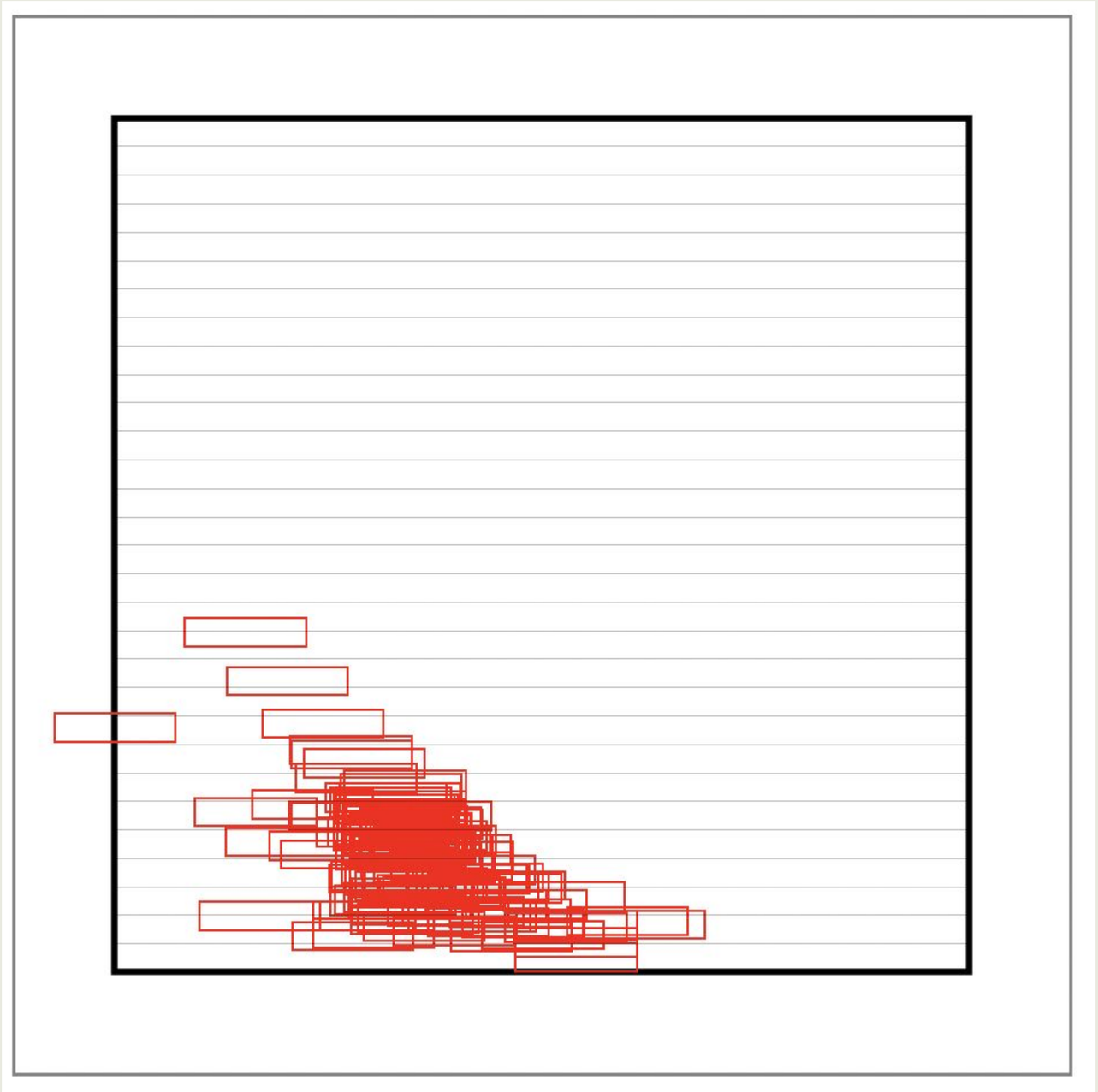
Experimental Results

structP Animation

- https://youtu.be/2Te6mHDalIs

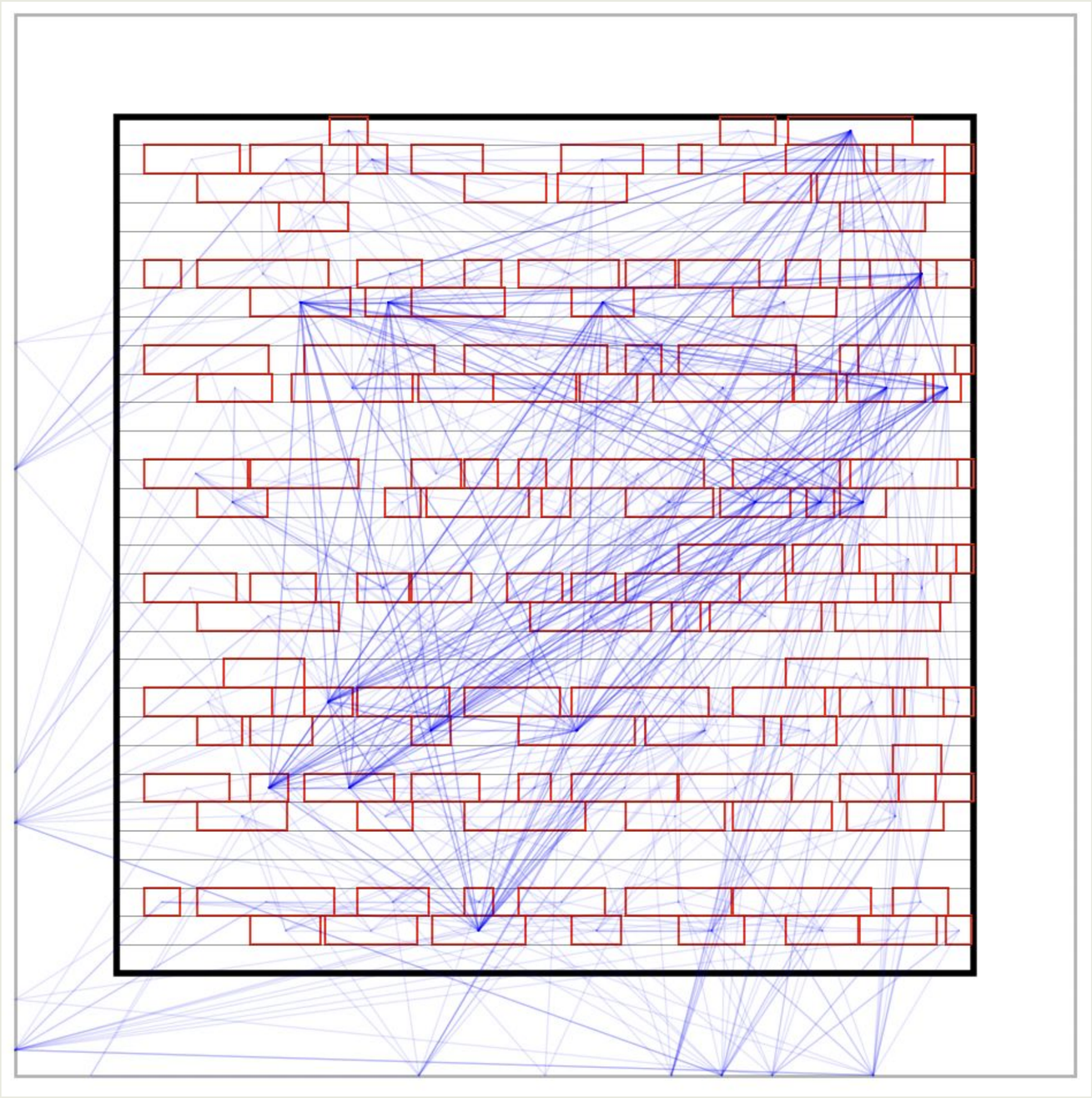
fract

- Level 0



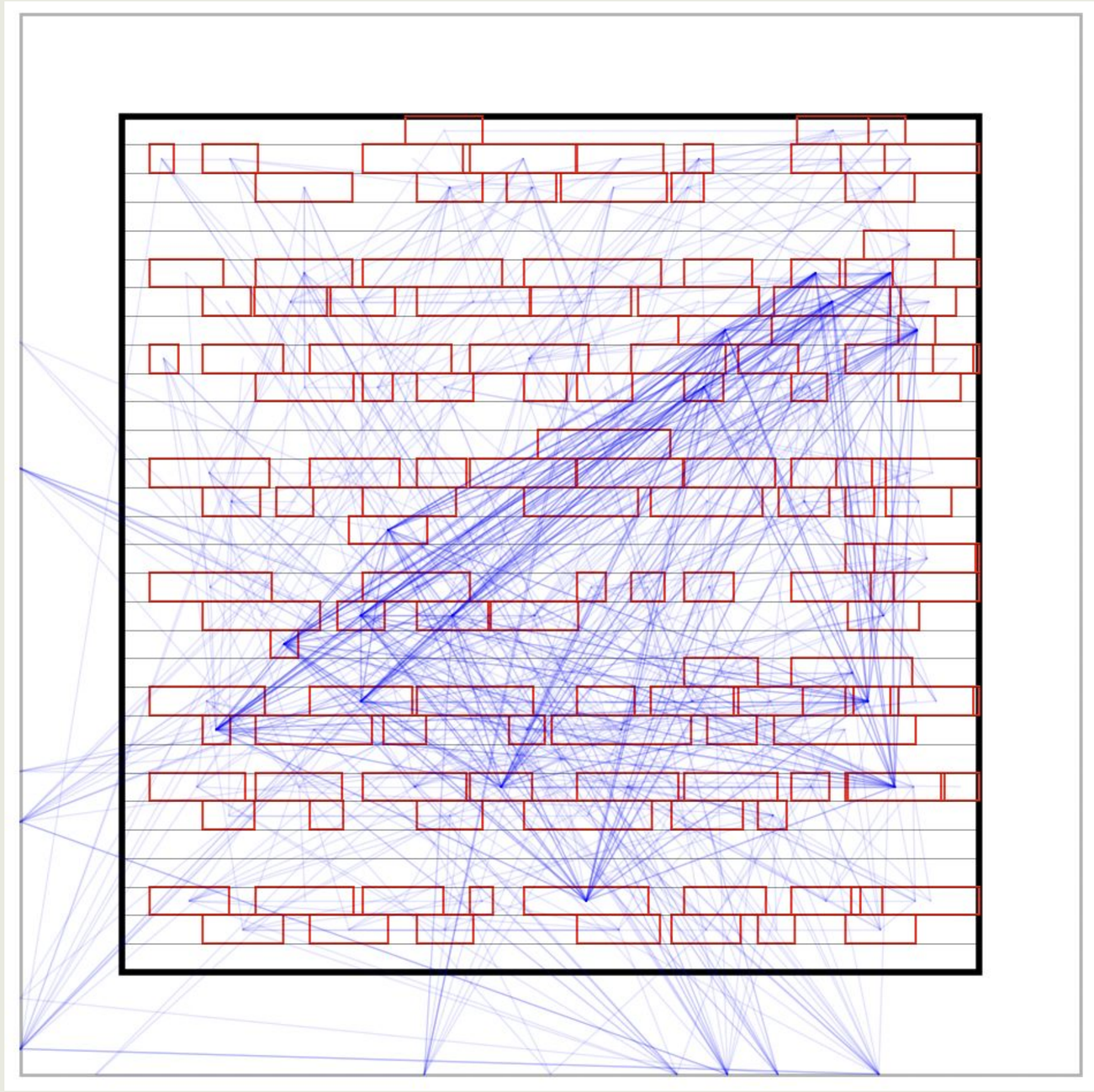
fract (quadprog)

- WL: 3884.82 um



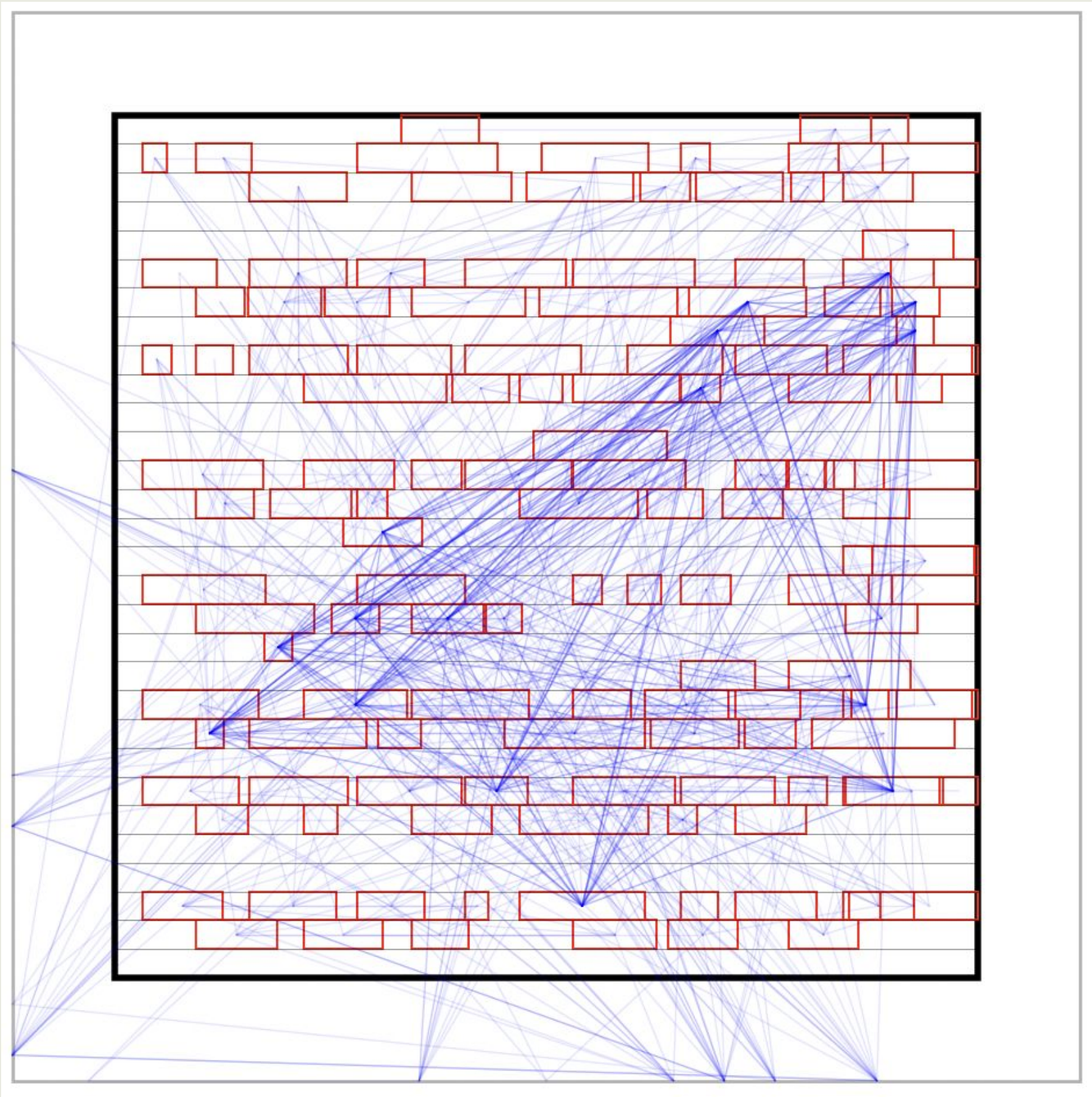
fract (cvxopt)

- WL: 5147.32 um



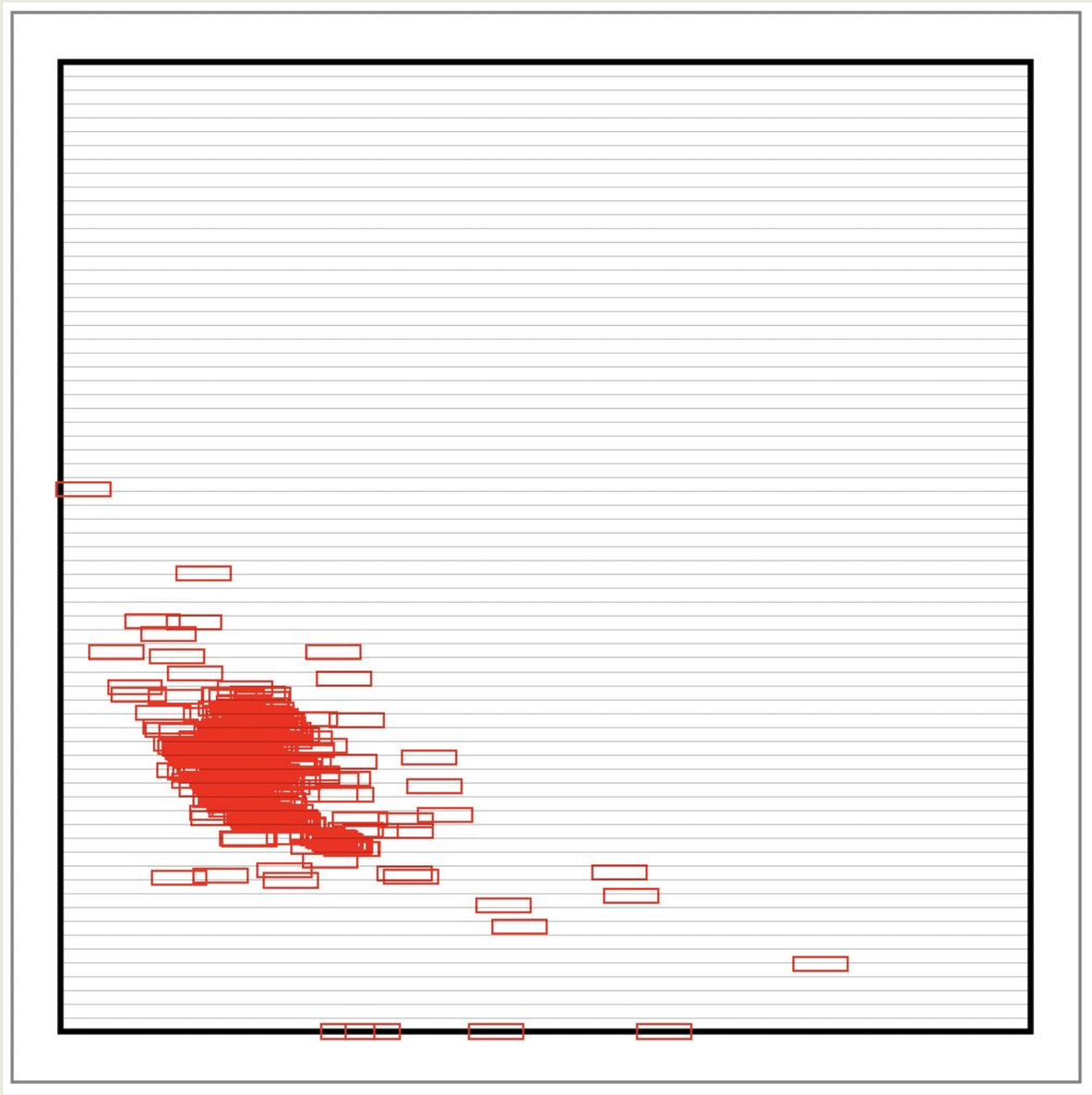
fract (cvxpy)

- WL: 5128.24 um



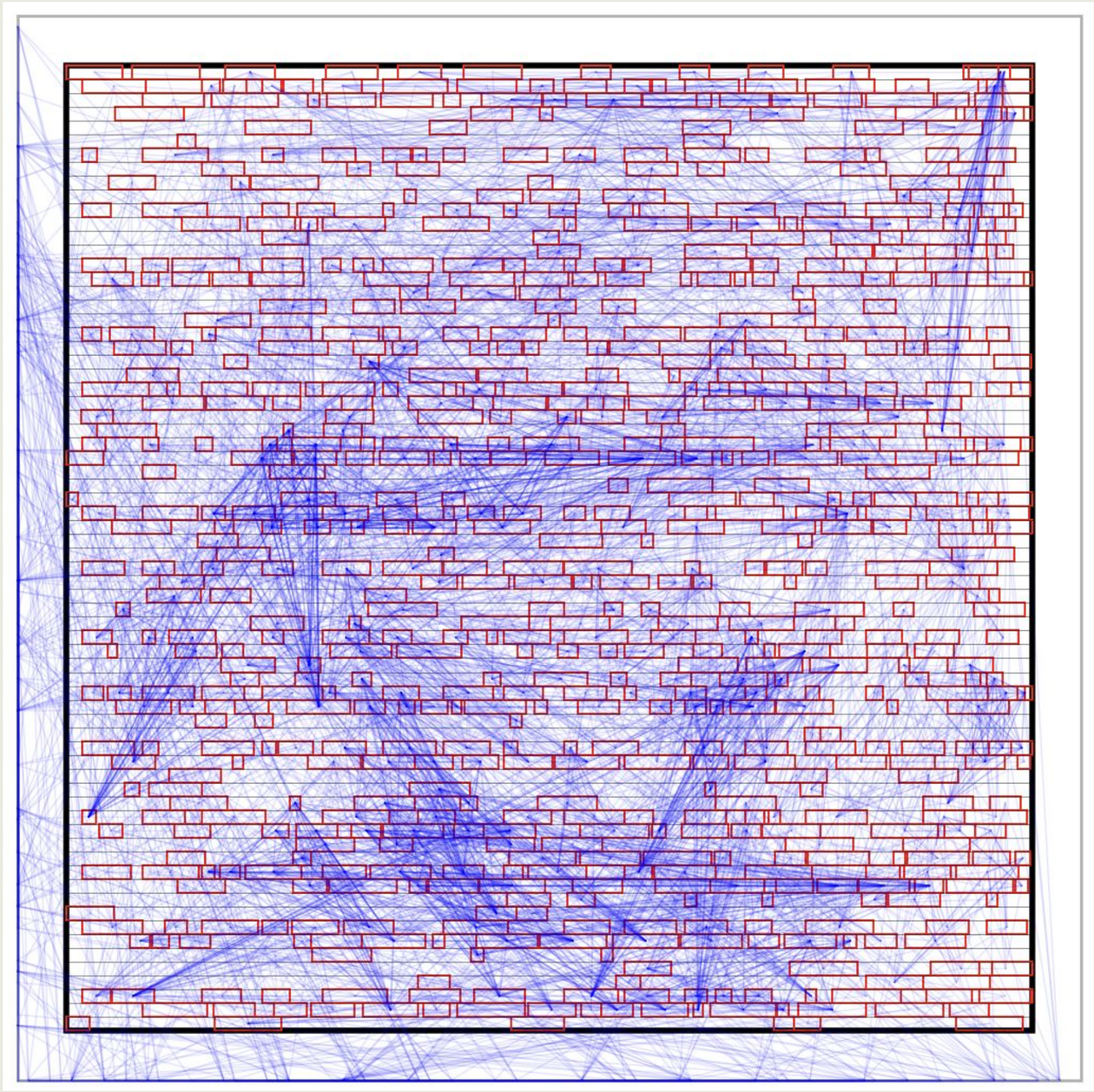
p1

- Level 0



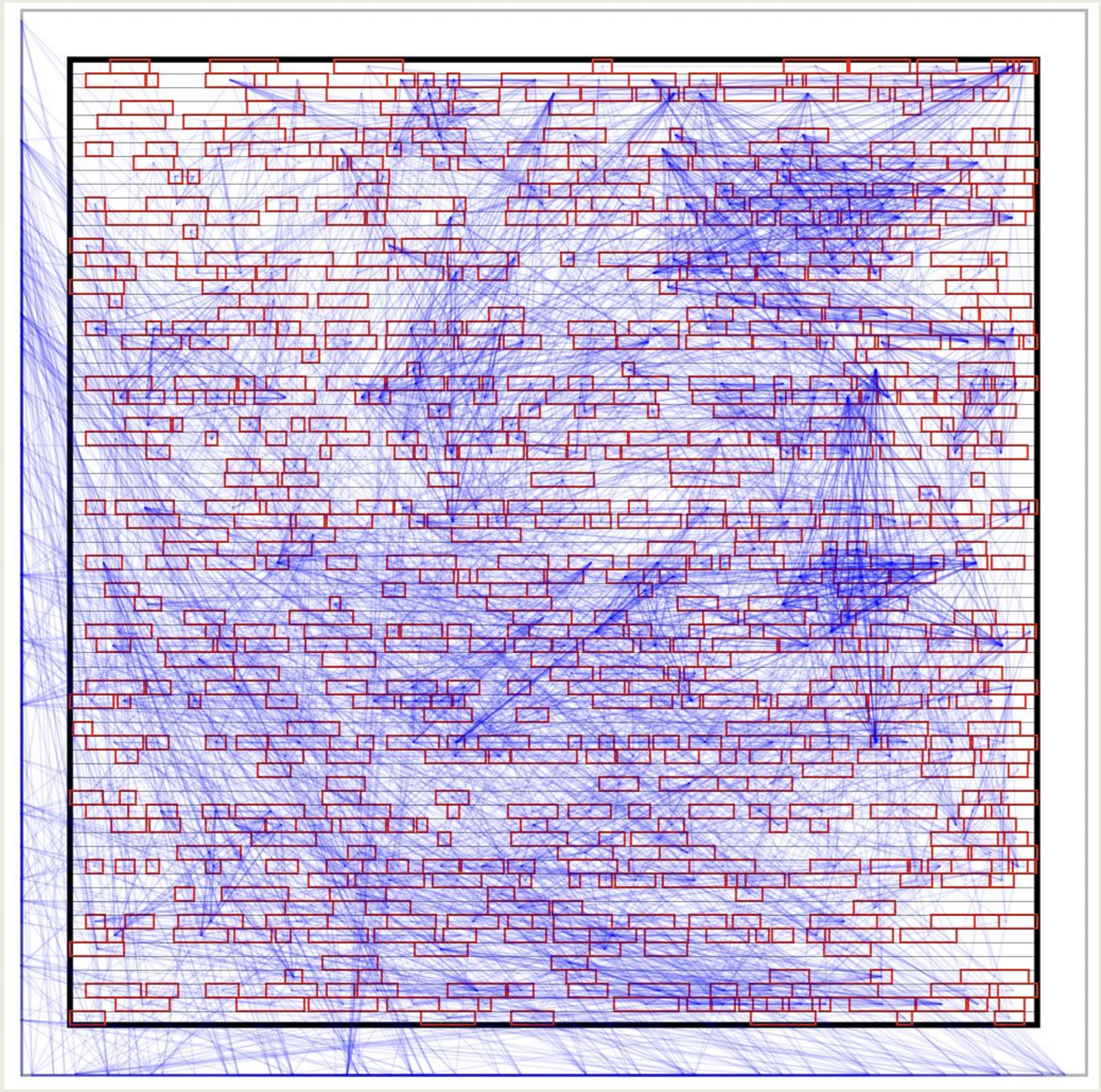
p1 (quadprog)

- WL: 38047.31 um



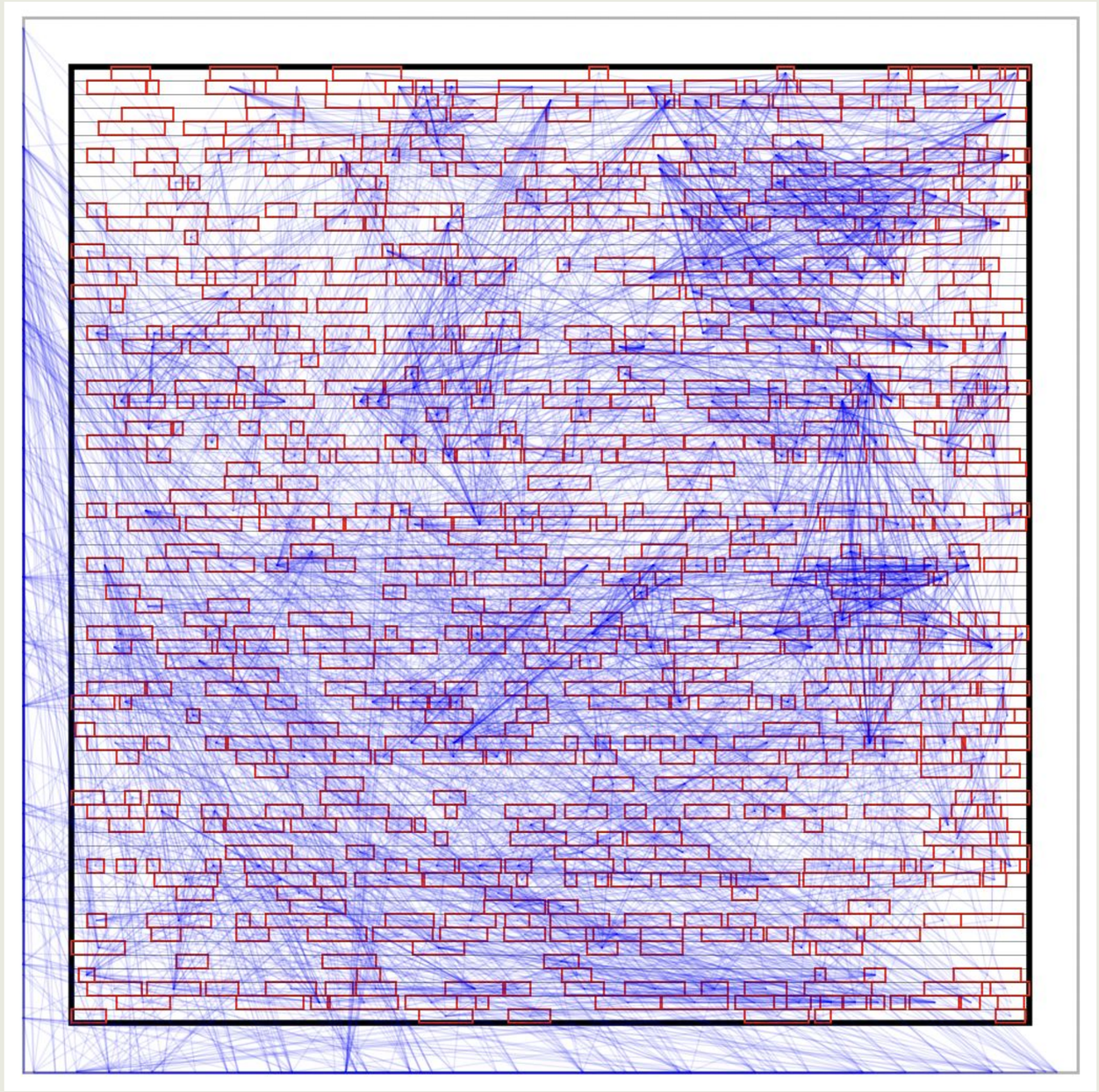
p1 (cvxopt)

- WL: 41523.63 um



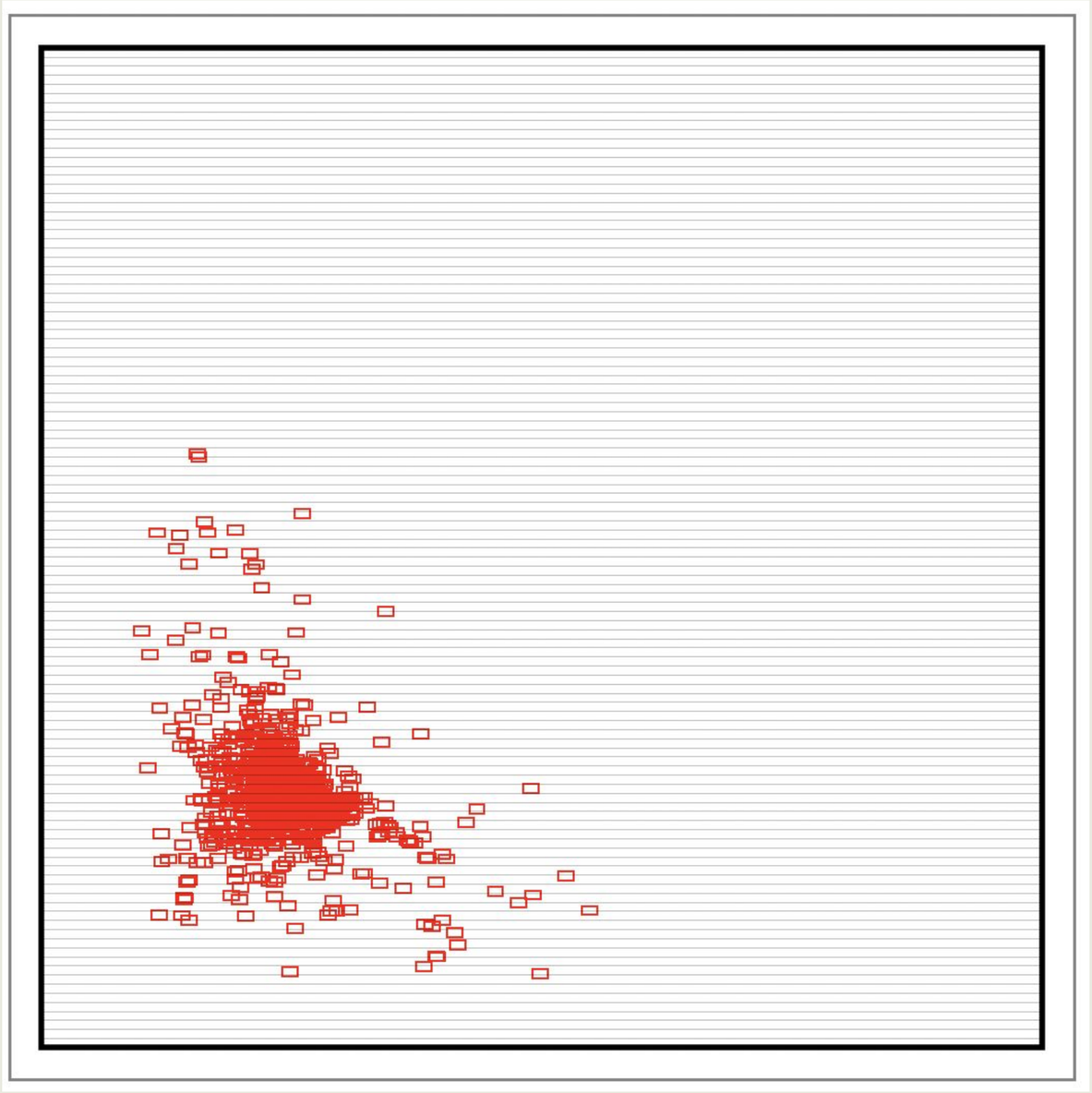
p1 (cvxpy)

- WL: 40939.63 um



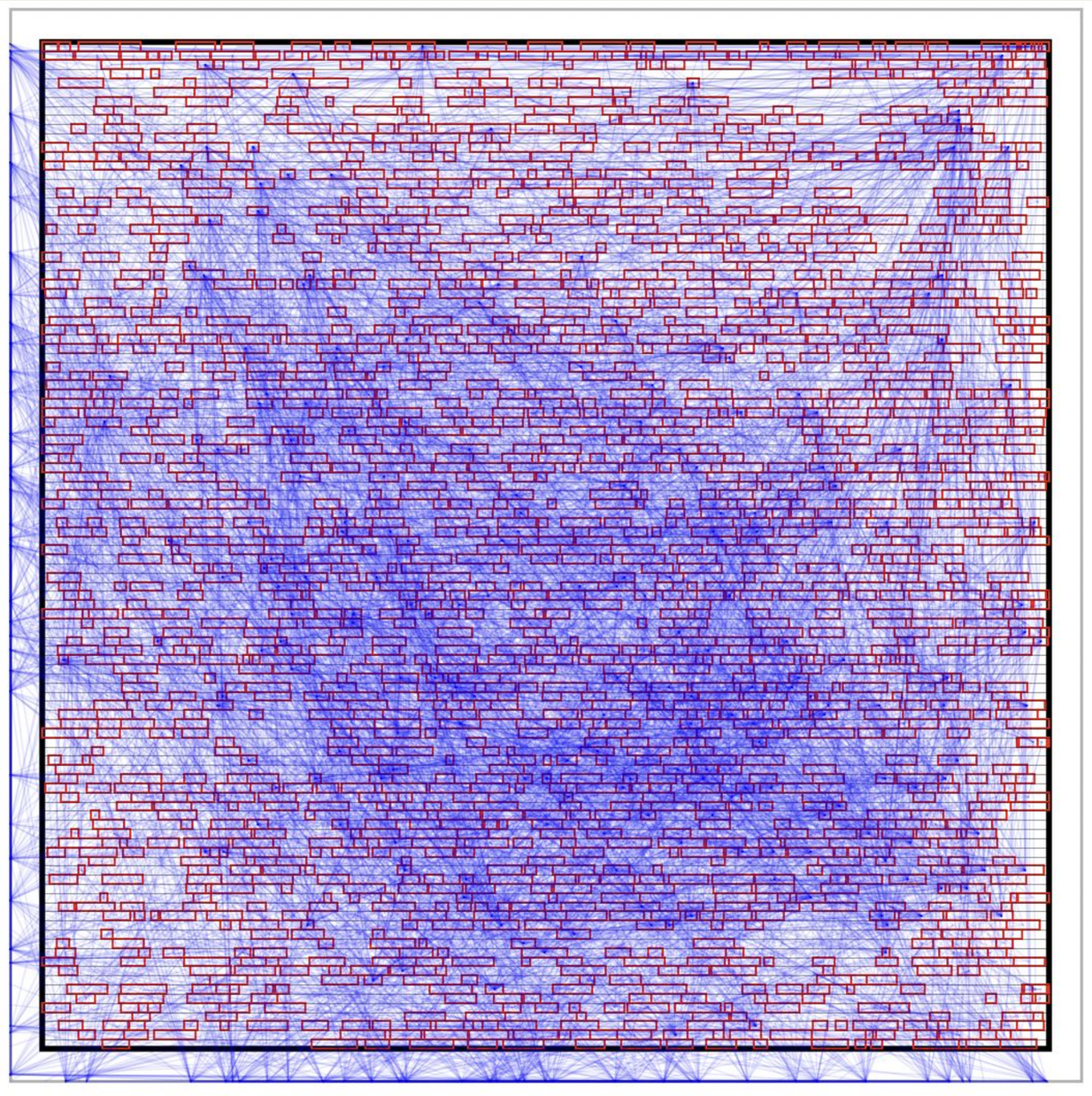
structP

- Level 0



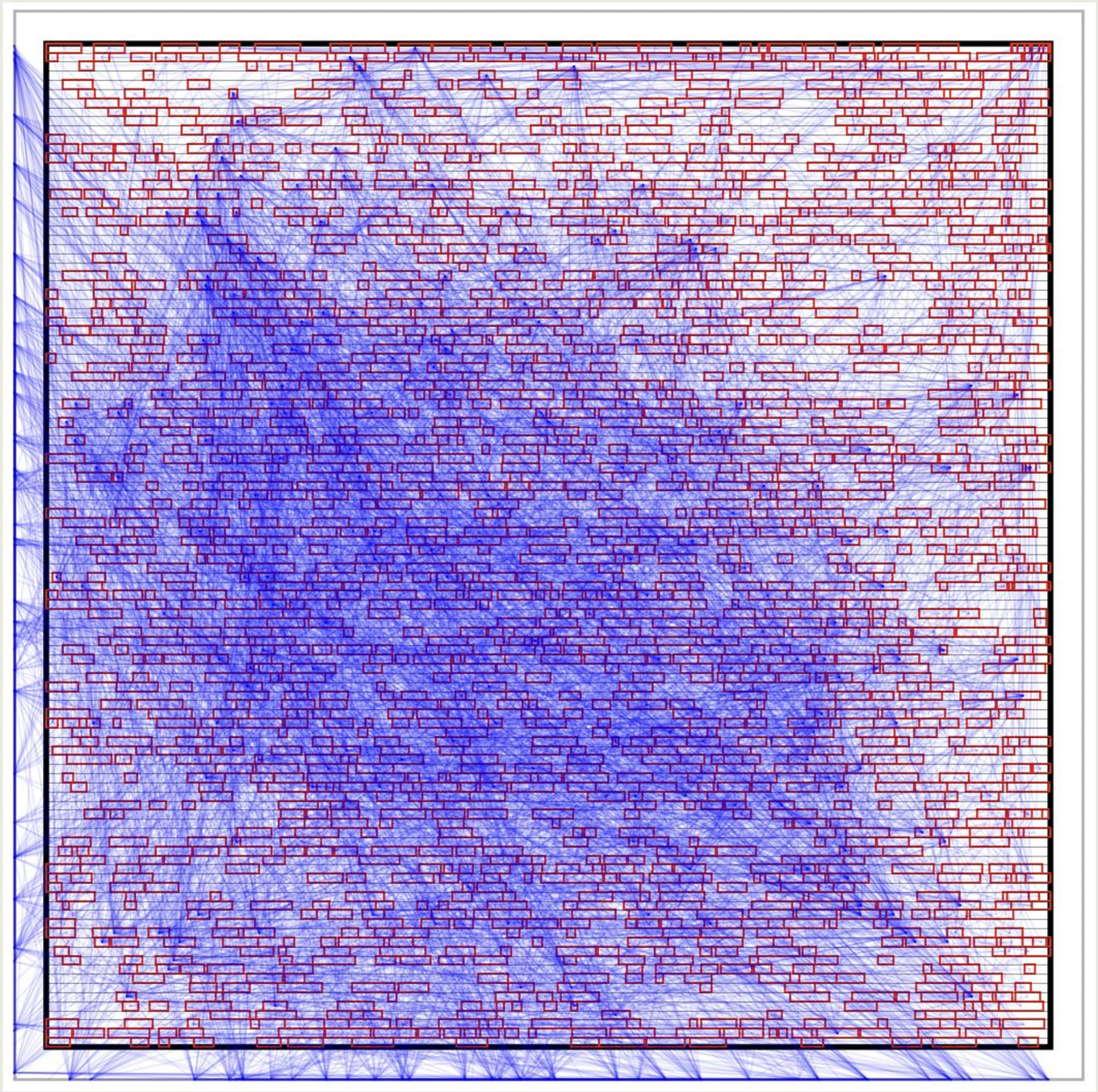
structP (quadprog)

- WL: 87918.99 um



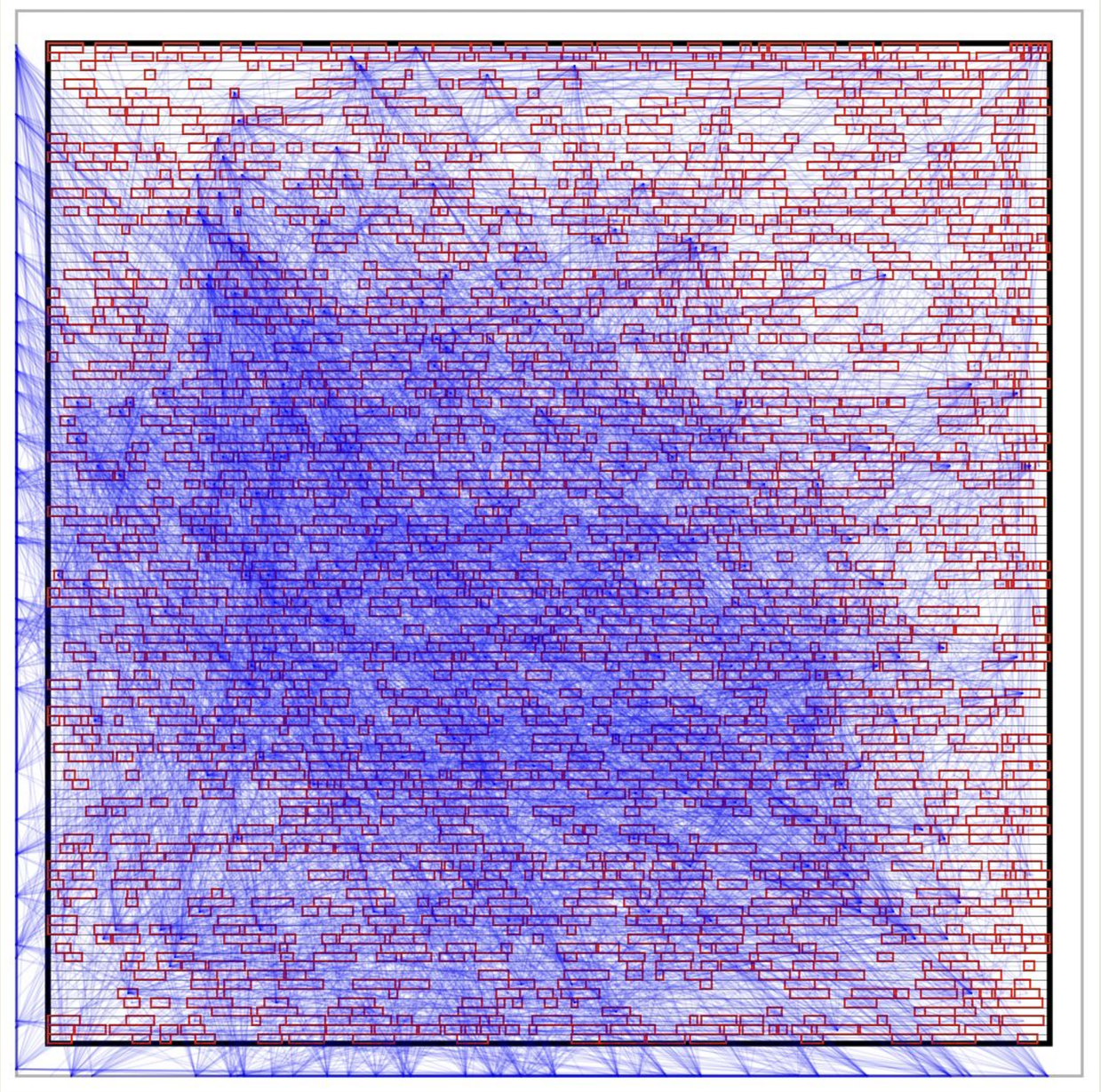
structP (cvxopt)

- WL: 97021.02 um



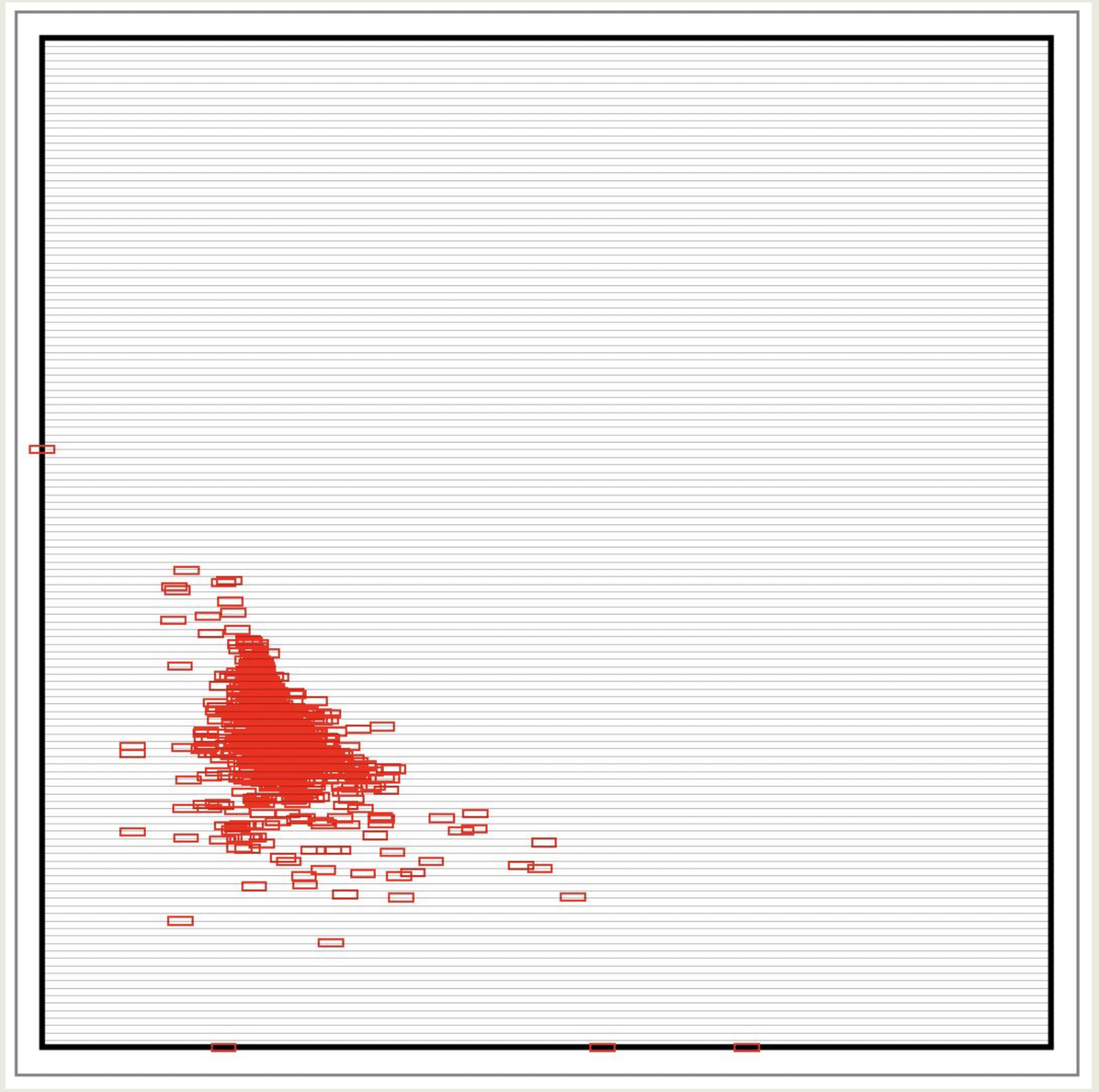
structP (cvxpy)

- WL: 97021.02 um



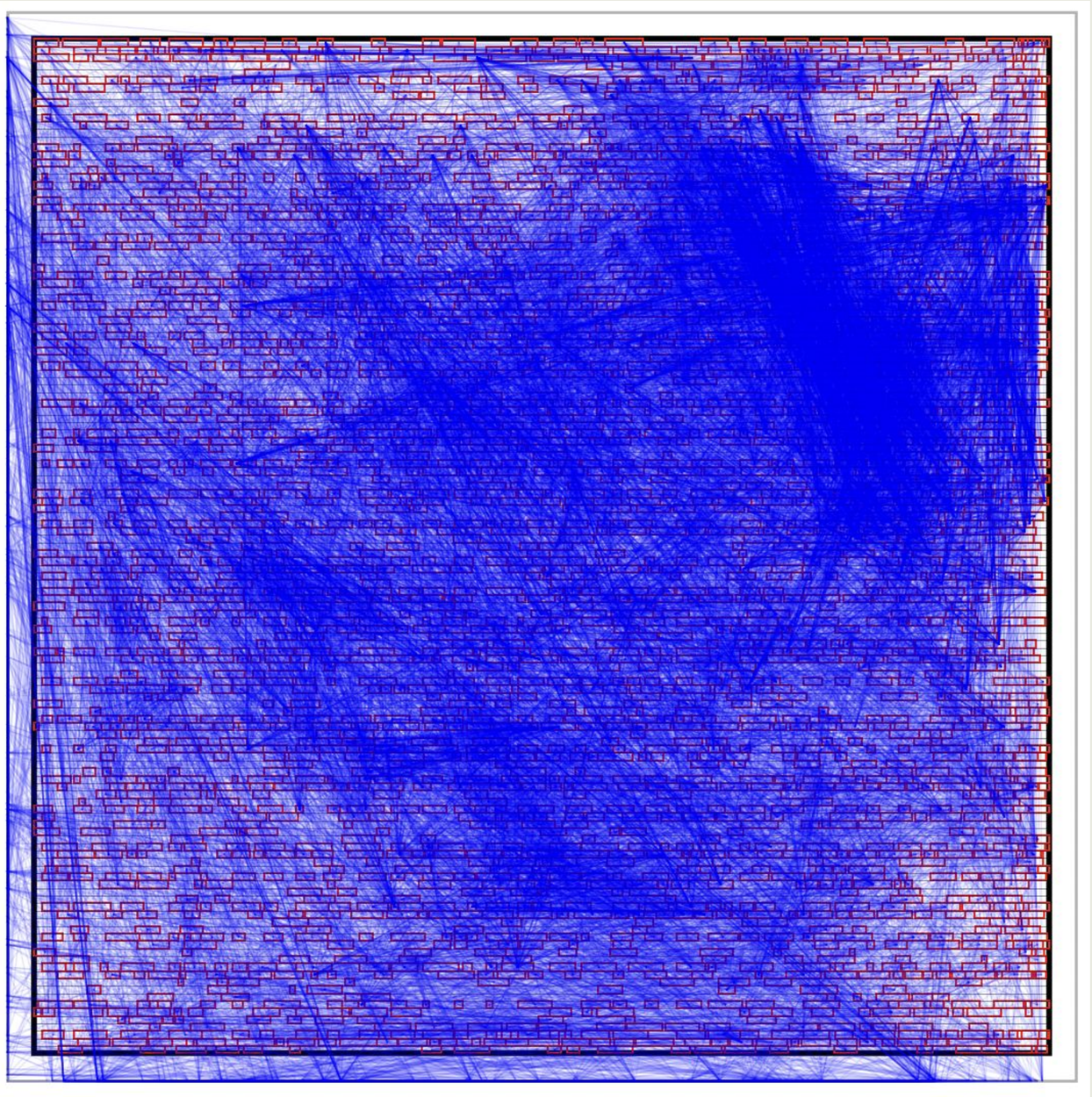
p2

- Level 0



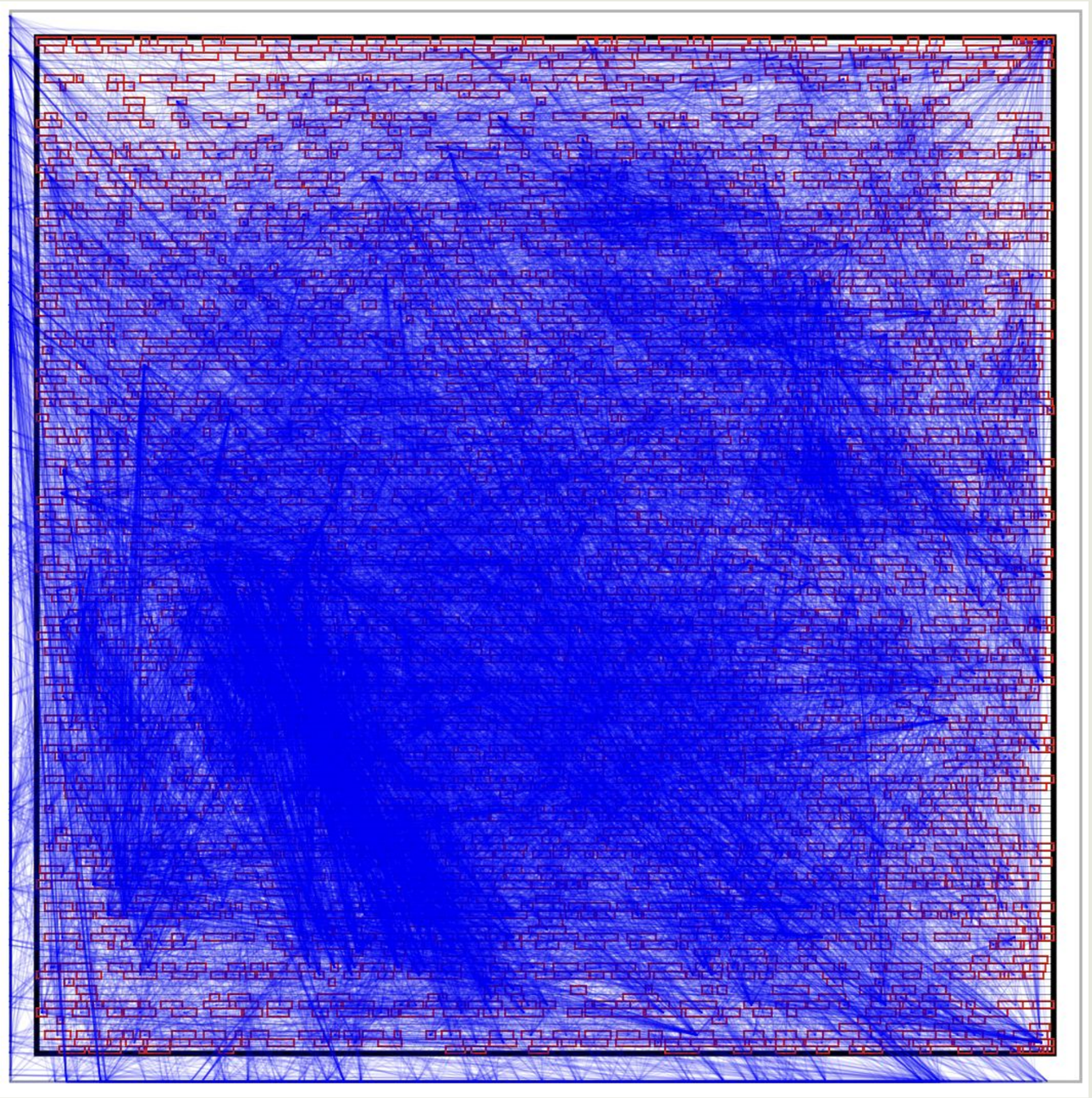
p2 (quadprog)

- WL: 251128.15 um
- Sparse quadprog with solve quadprog to get proper level 0



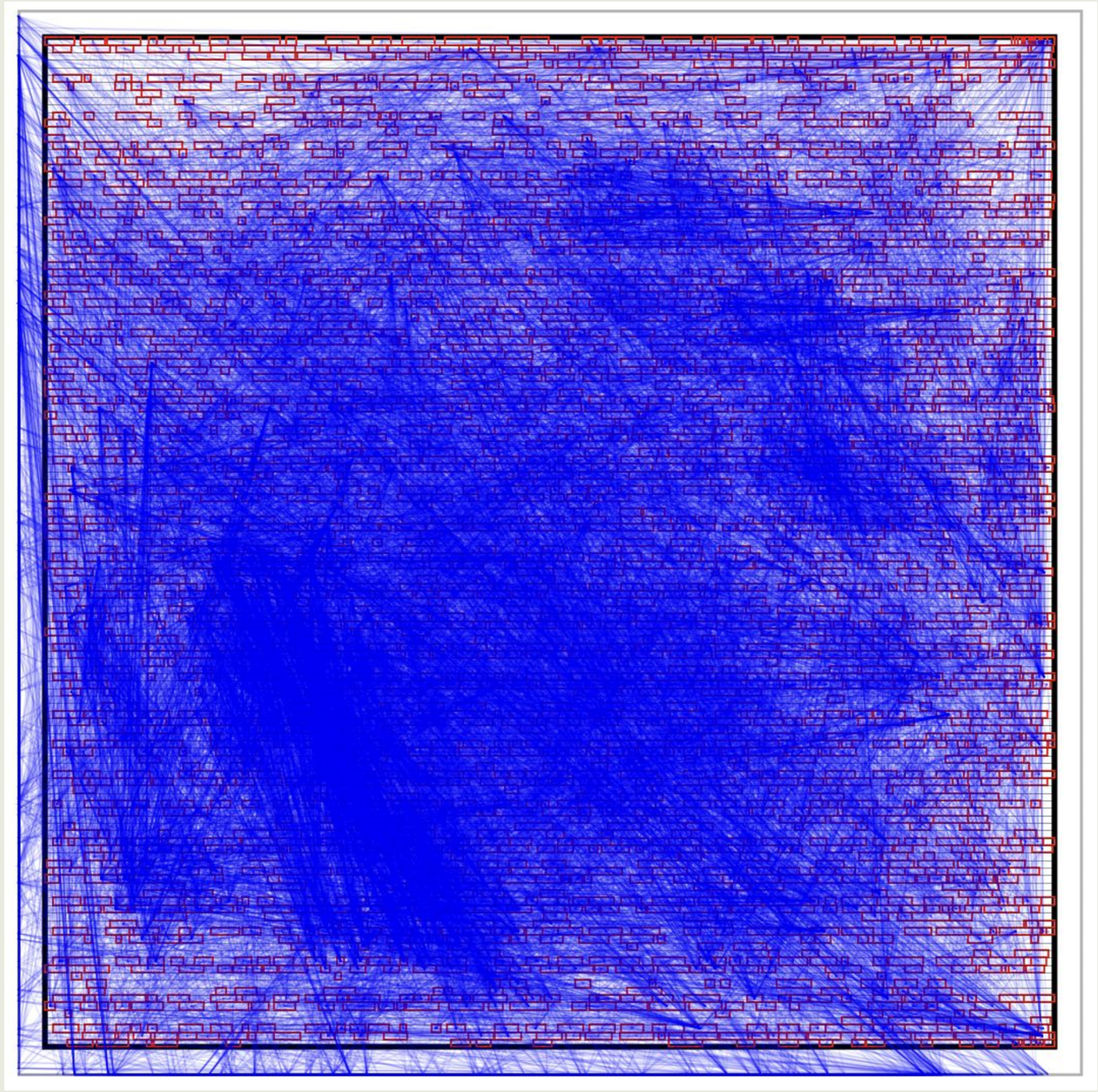
p2 (cvxopt)

- WL: 278495.38 um



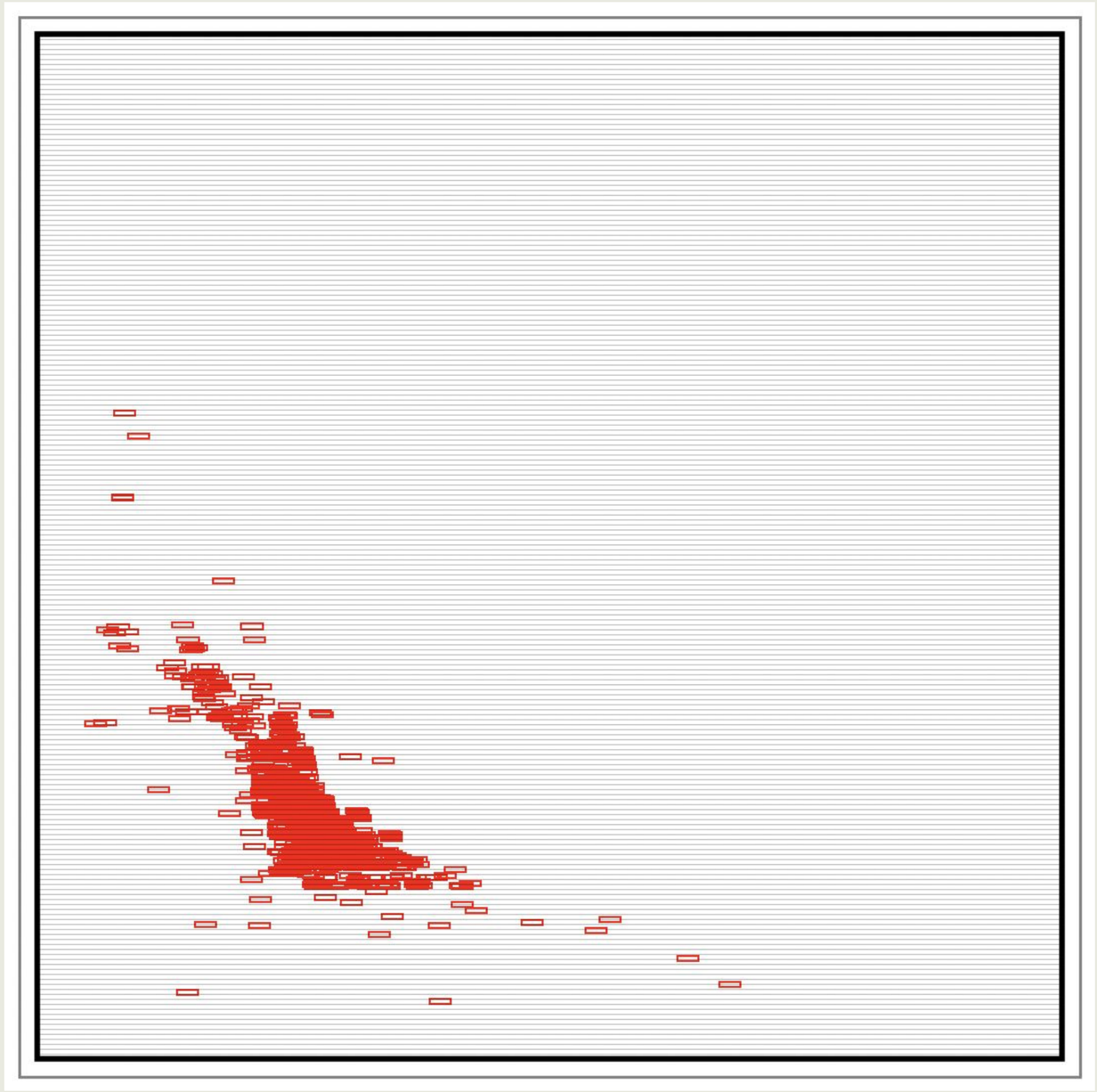
p2 (cvxpy)

- WL: 278492.12 um



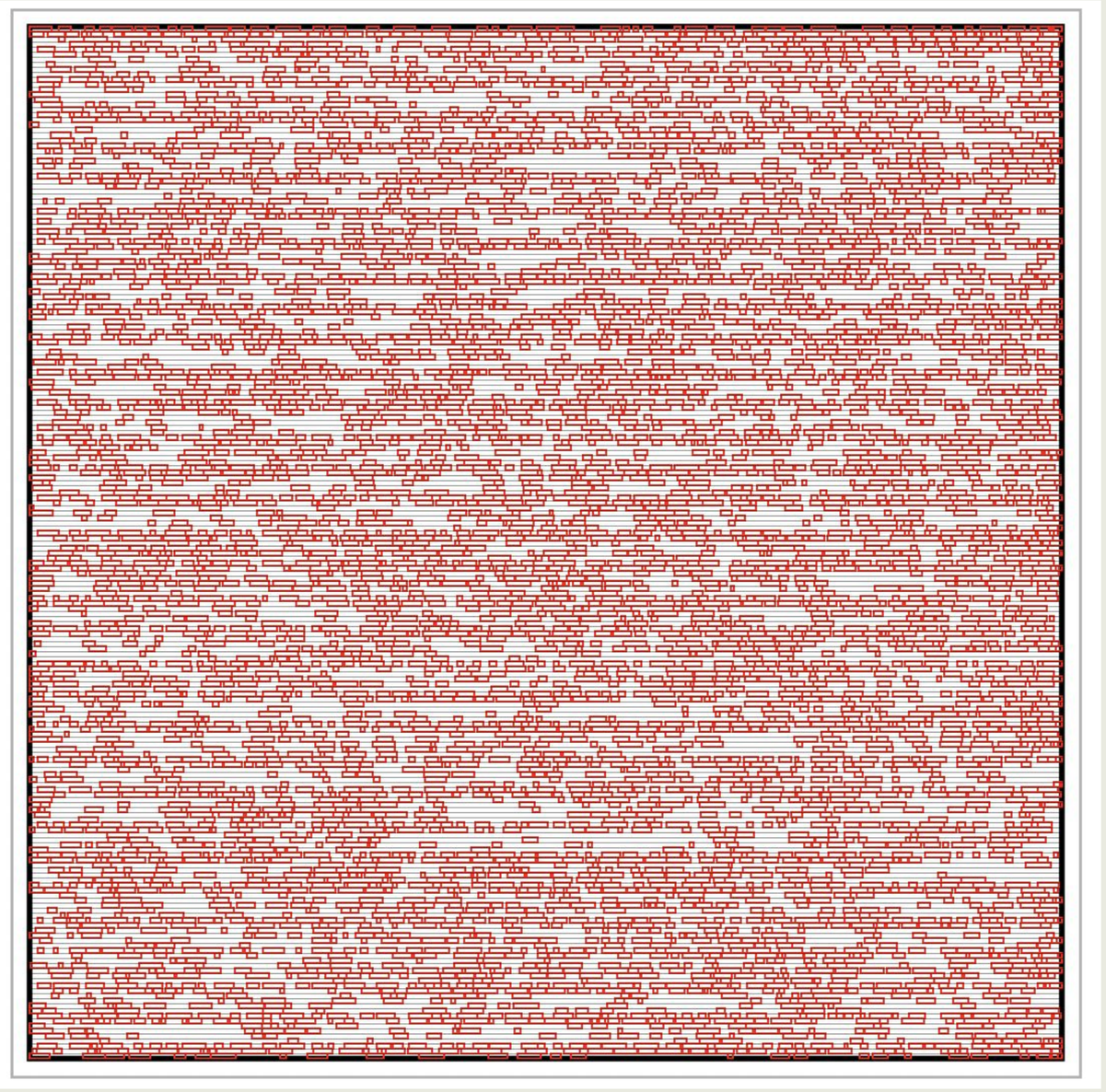
biomedP

- Level 0



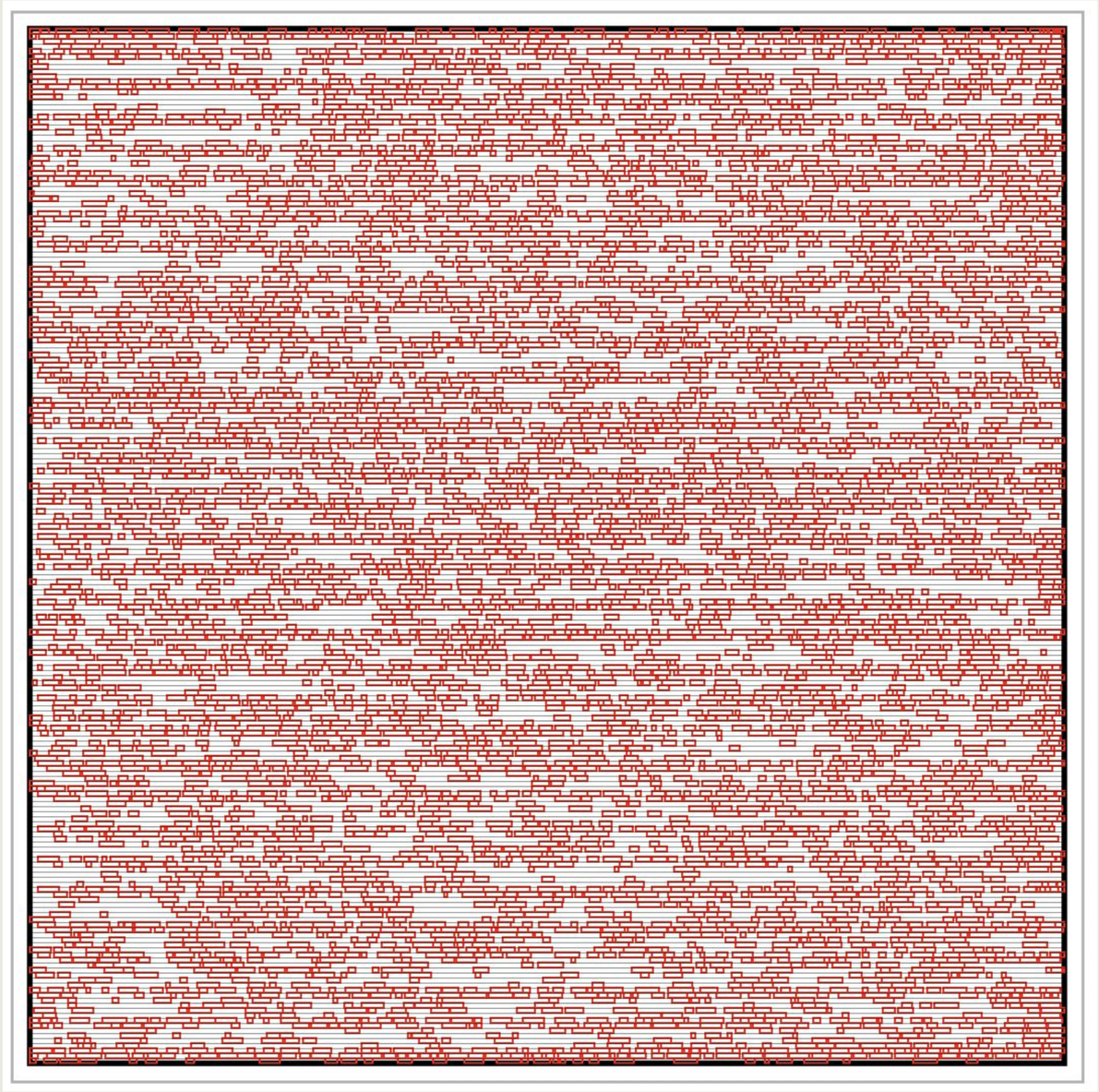
biomedP (quadprog)

- WL: 593121.68 um
- Wires omitted for visualization



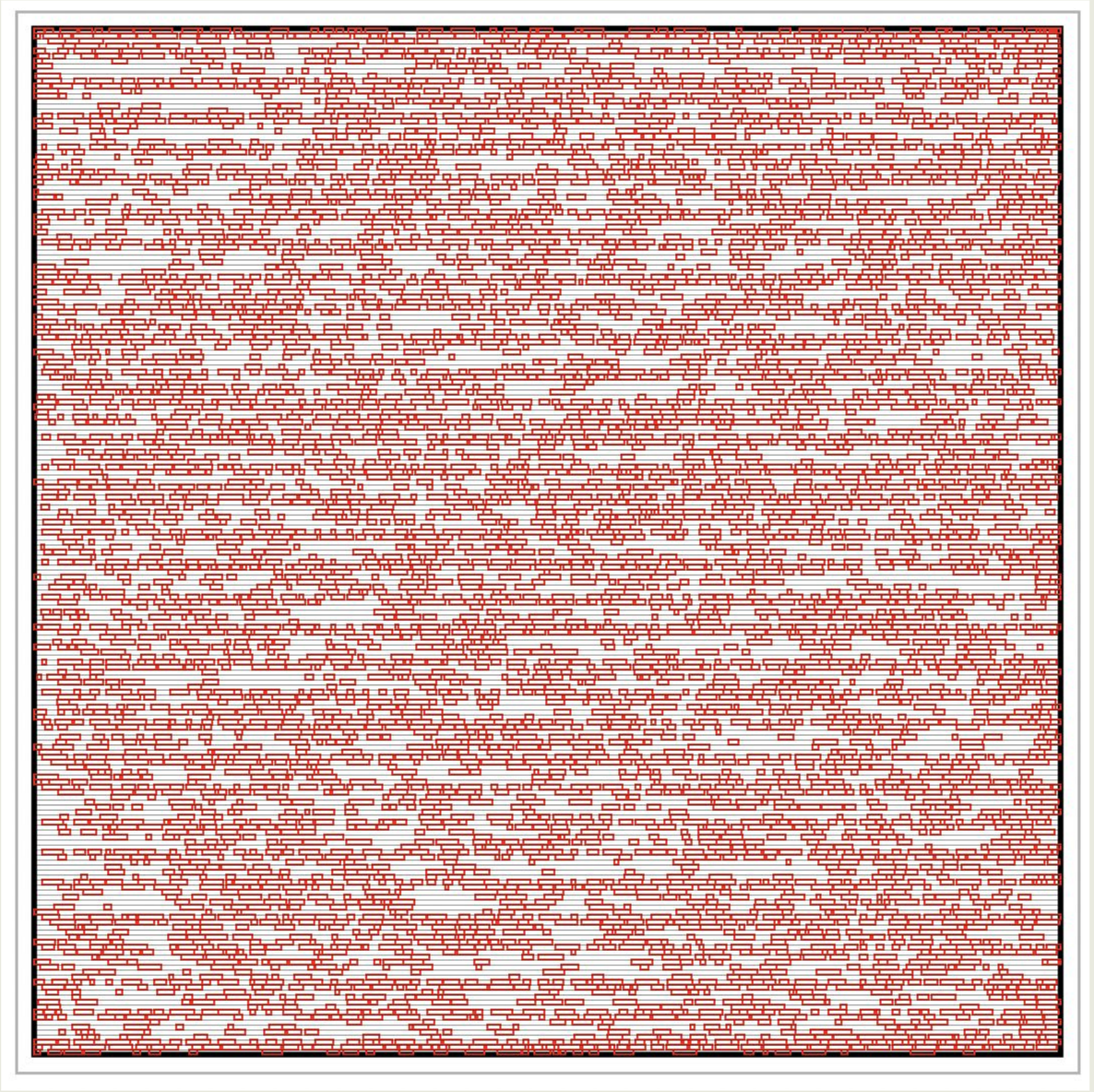
biomedP (cvxopt)

- WL: 633216.28 um
- Wires omitted for visualization



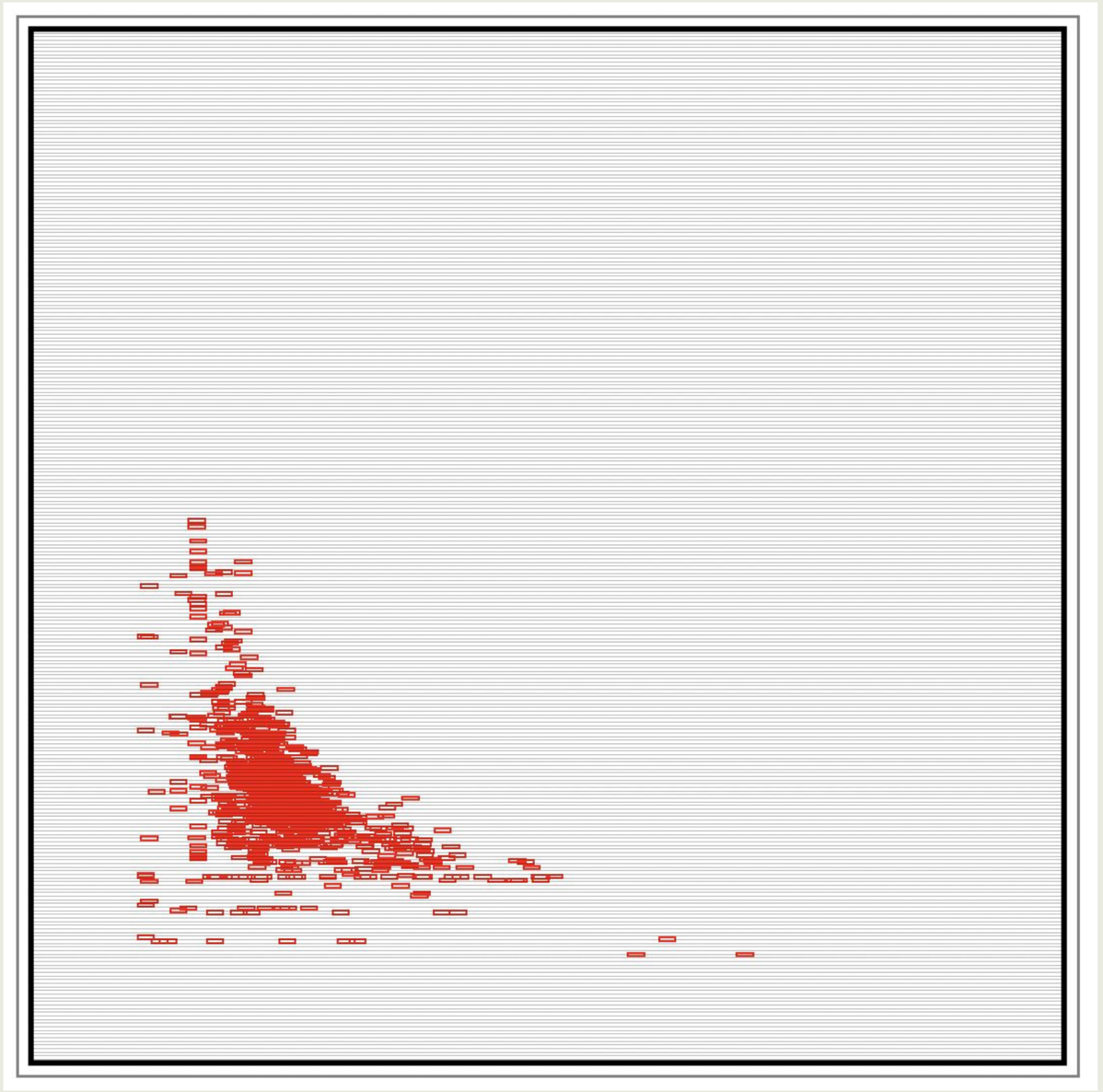
biomedP (cvxpy)

- WL: 633216.28 um
- Wires omitted for visualization



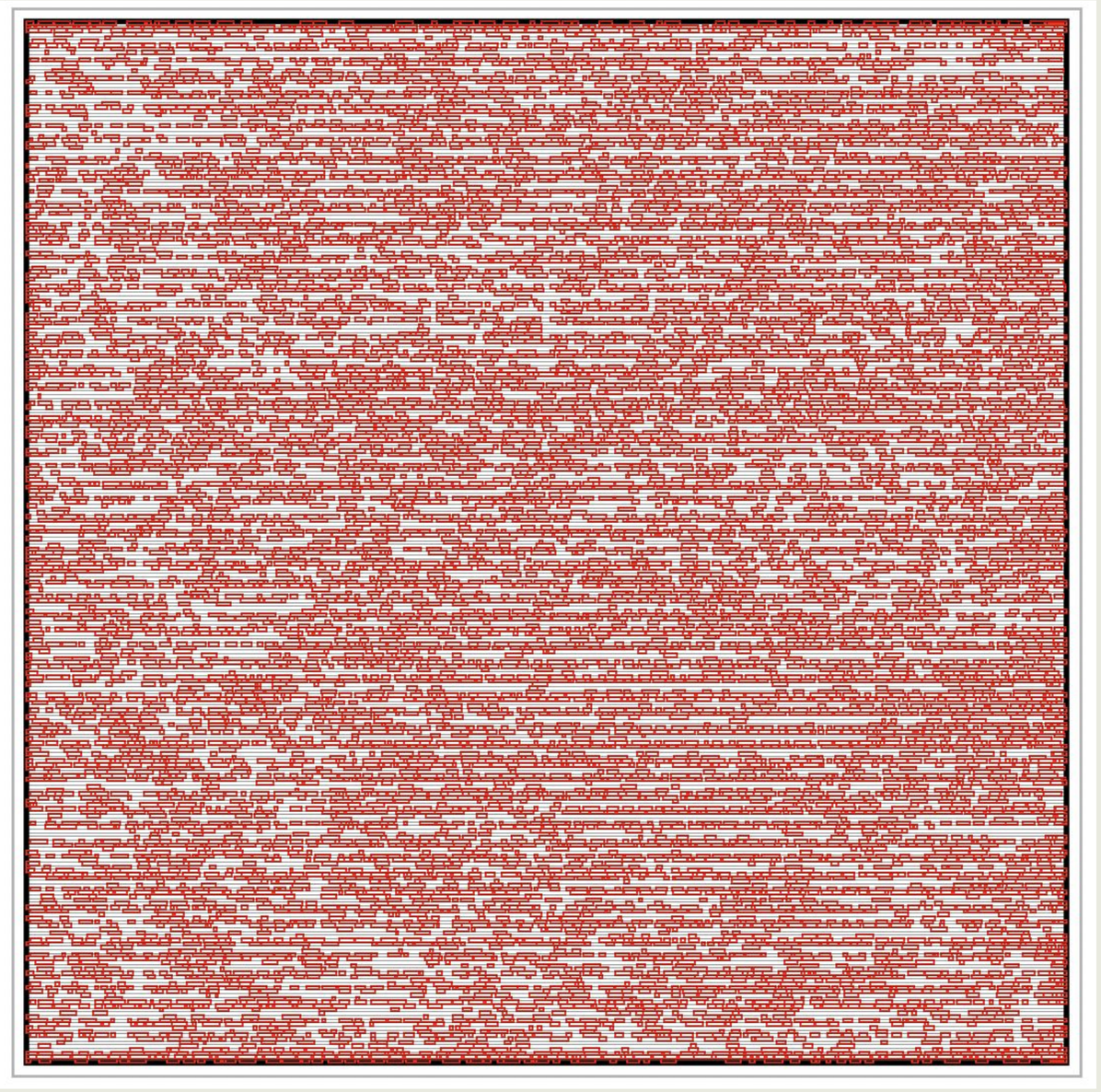
industry2

- Level 0



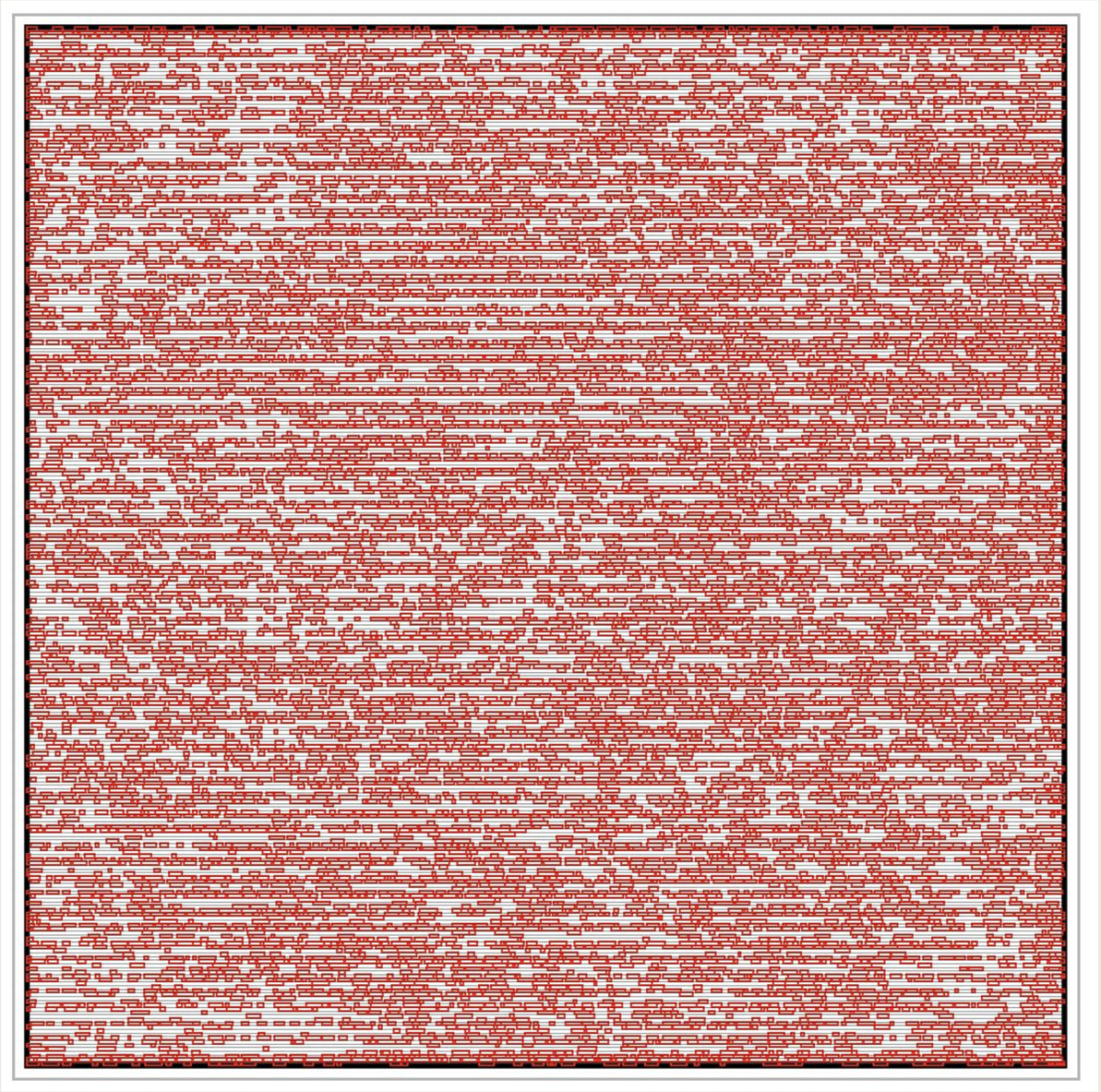
industry2 (quadprog)

- WL: 1494033.27 um
- Wires omitted for visualization



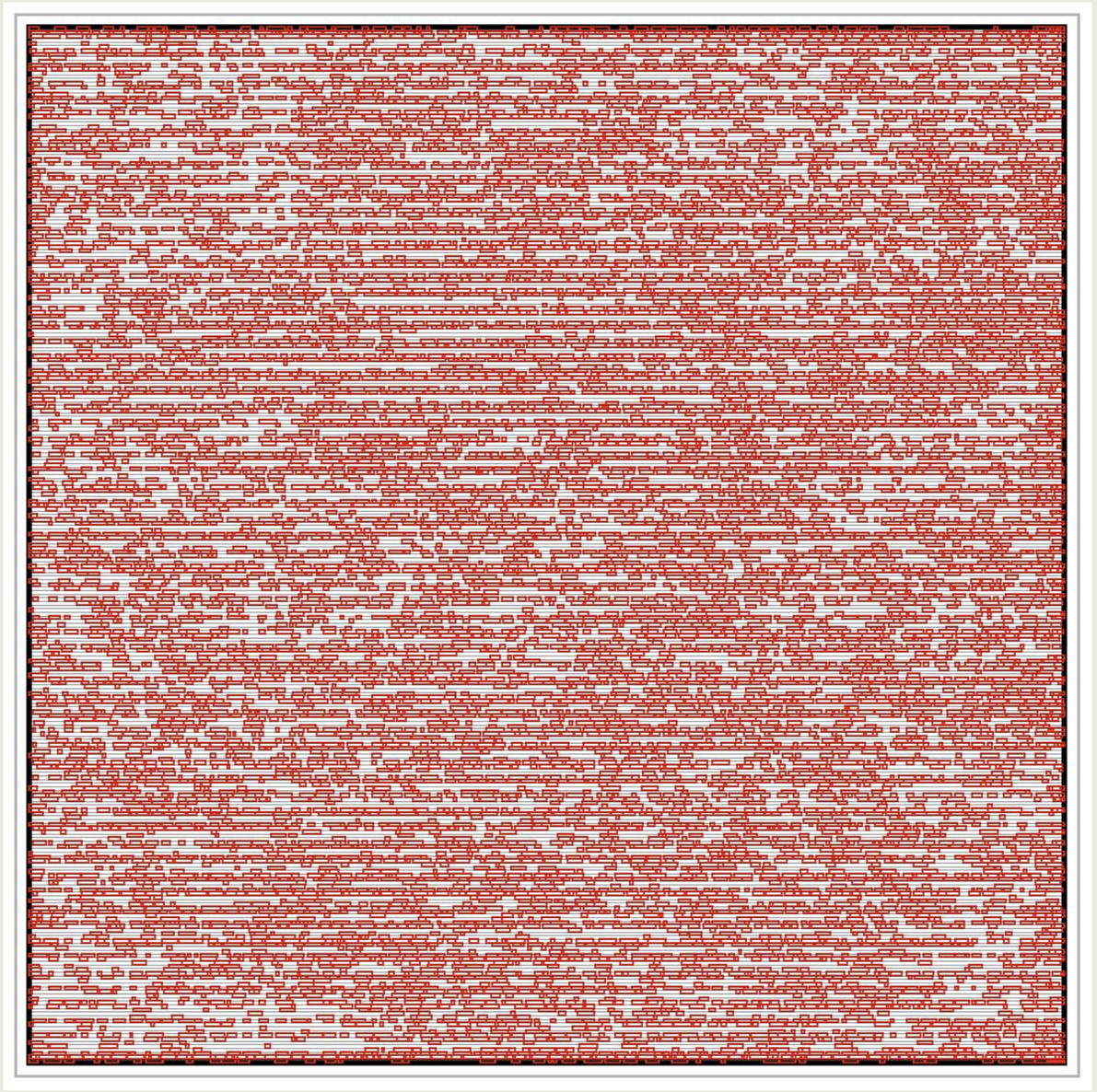
industry2 (cvxopt)

- WL: 1709006.47 um
- Wires omitted for visualization



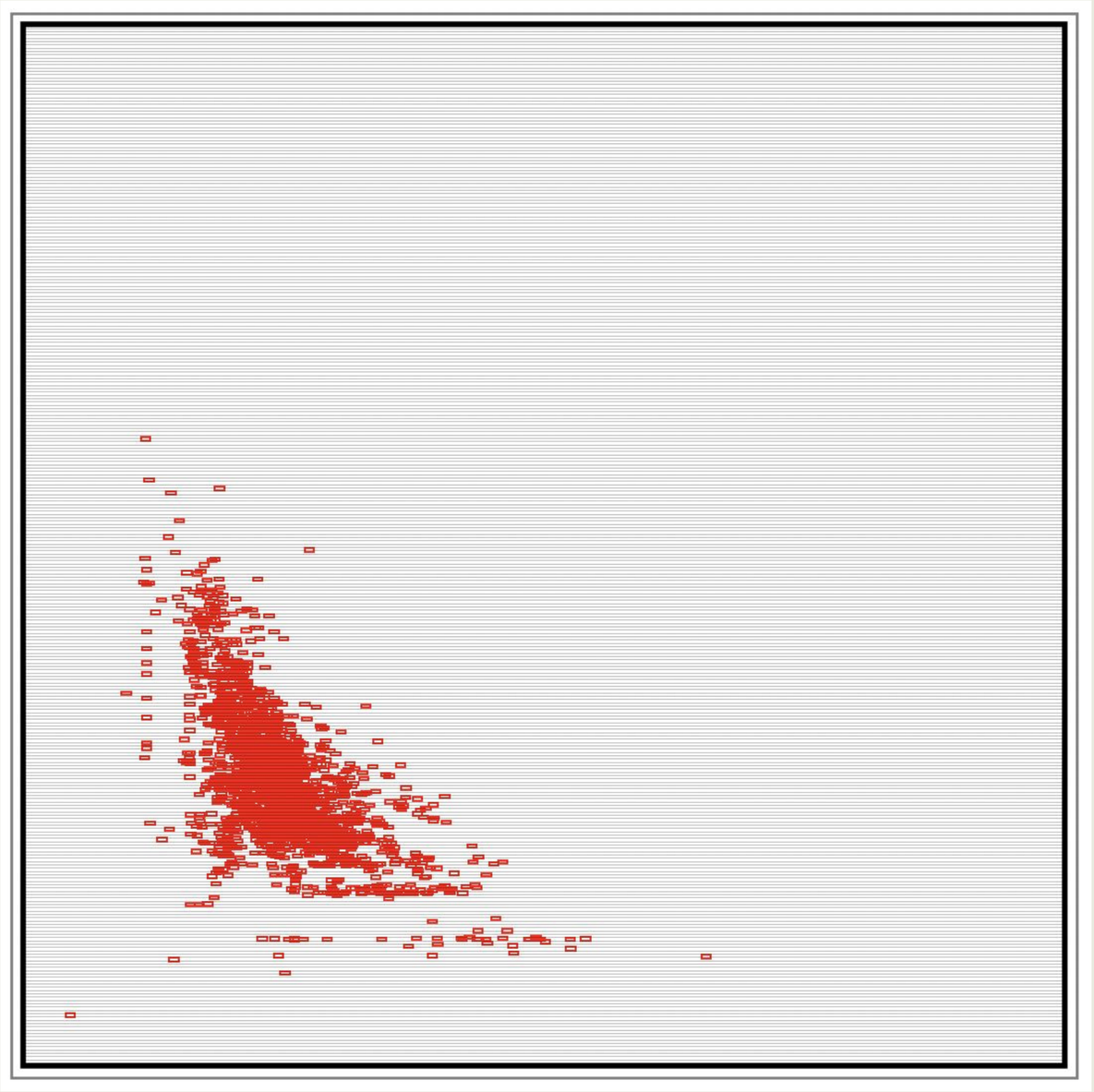
industry2 (cvxpy)

- WL: 1705421.41 um
- Wires omitted for visualization



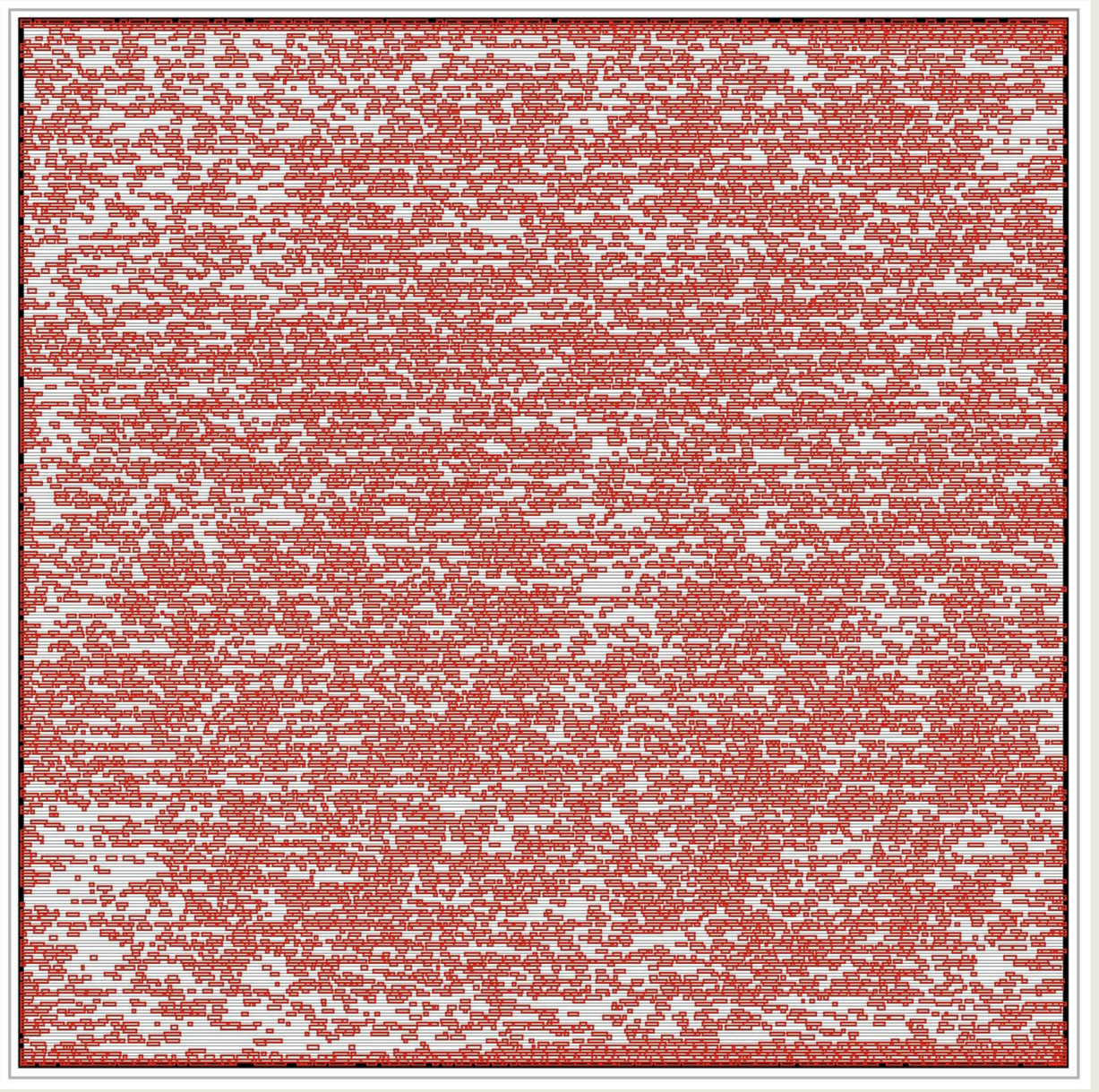
industry3

- Level 0



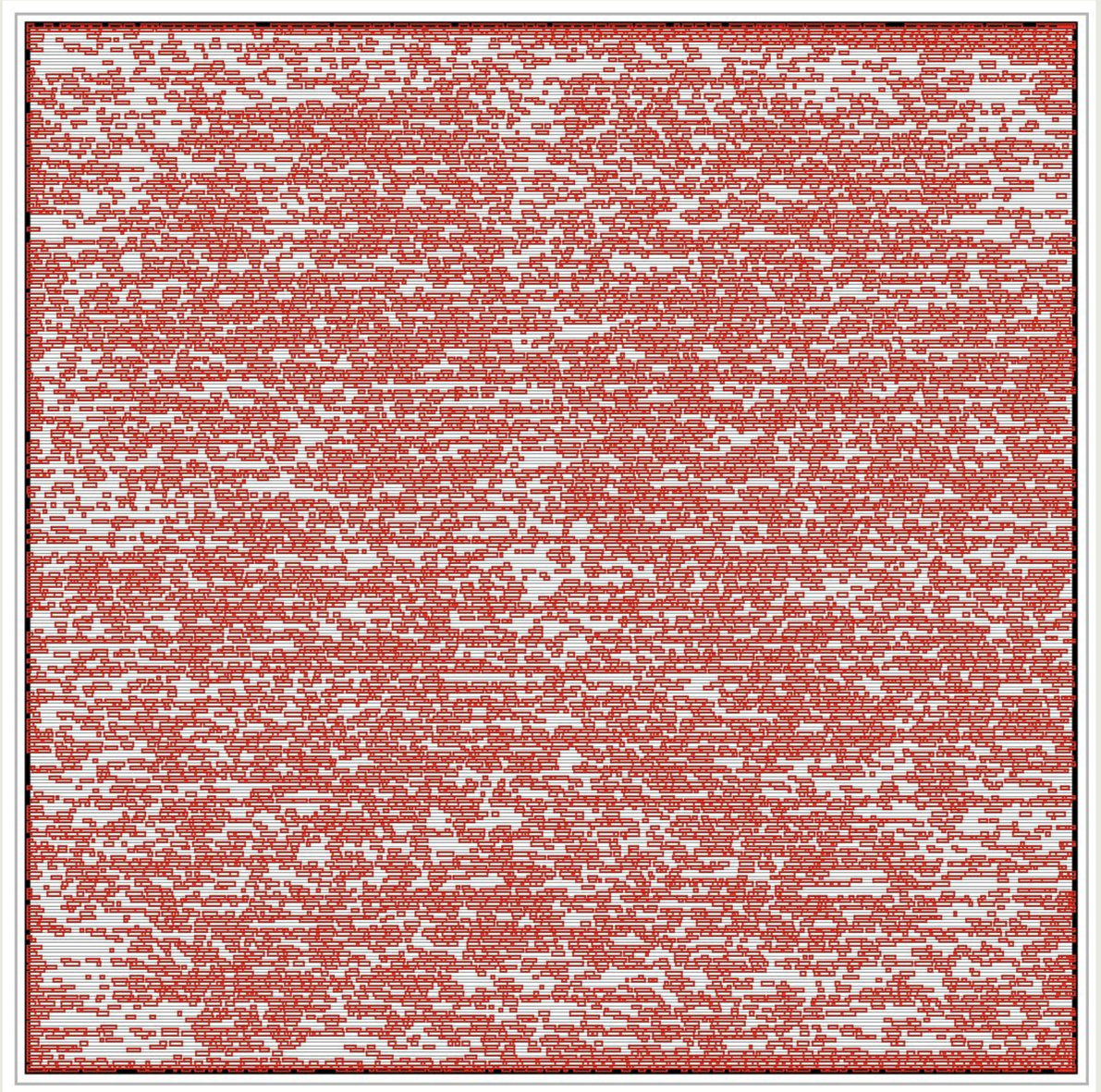
industry3 (quadprog)

- WL: 2640718.47 um
- Wires omitted for visualization



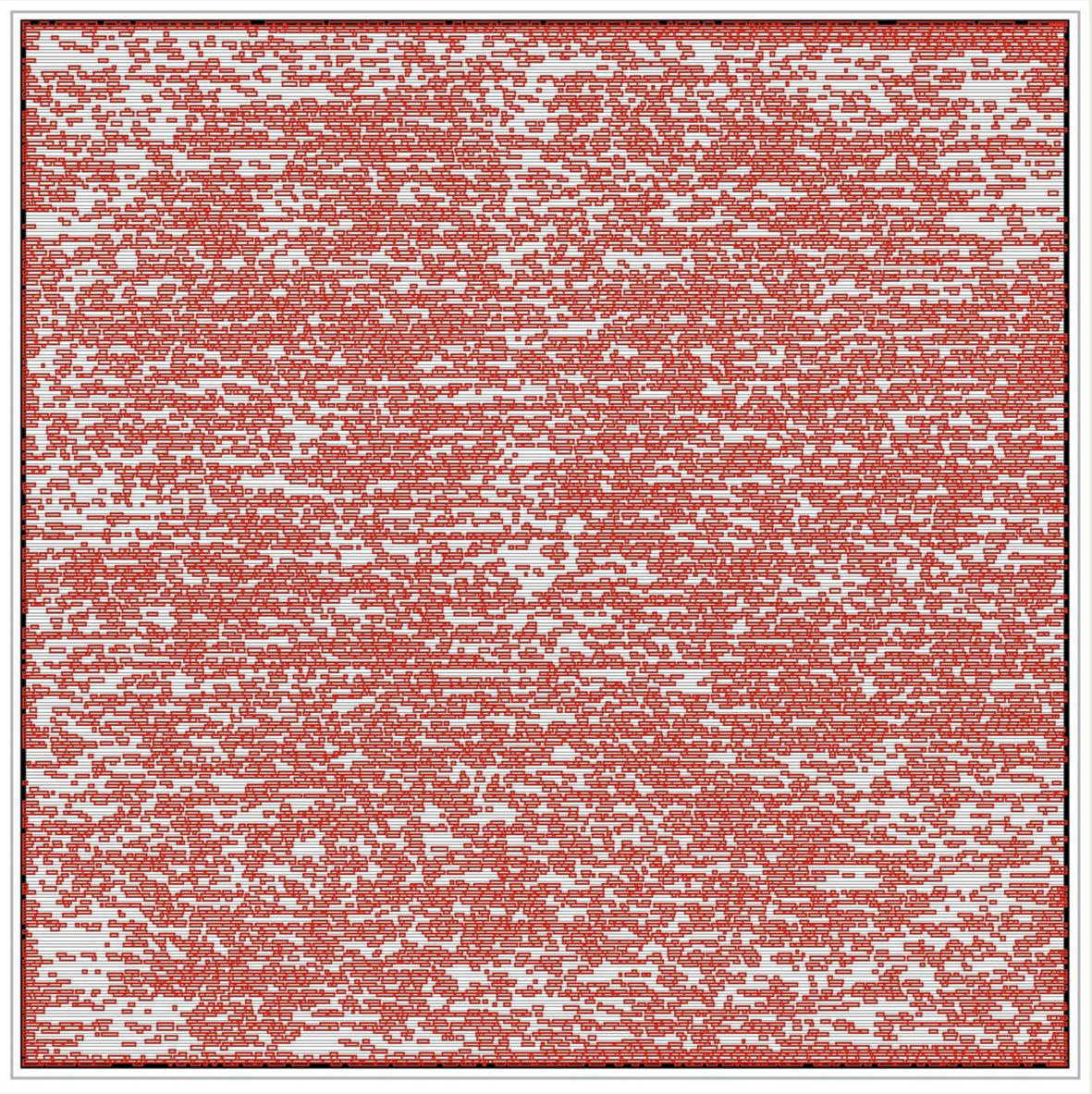
industry3 (cvxopt)

- WL: 2801792.99 um
- Wires omitted for visualization



industry3 (cvxpy)

- WL: 2796820.28 um
- Wires omitted for visualization



Results

Circuit	Standard Cells	Run Time (quadprog)	Run Time (osqp)	Run Time (cvxpy)	Wirelength (quadprog)	Wirelength (cvxopt)	Wirelength (cvxpy)	Lowest Level
fract	136	~2.5 - 3 s	~2 s	~2.5 - 3 s	3884.82	5147.32	5128.24	7
p1	735	~3 s	~2 s	~4 s	38047.13	41523.63	40939.63	9
structP	1,850	~5 s	~5 s	~4 s	87918.99	97021.02	97021.02	10
p2	2,826	~2 mins	~9 s	~9.5 s	251128.15	278495.38	278492.12	11
biomedP	6,228	~3.5 mins	~1.5 mins	~2 mins	593121.68	633216.28	633216.28	12
industry2	12,237	~15 mins	~5 mins	~5 mins	1494033.27	1709006.47	1705421.41	13
industry3	14,968	~24 mins	~3.5 mins	~4 mins	2640718.47	2801792.99	2796820.28	13

Conclusion

- Developed global placement tool using quadratic programming and greedy detailed placement
- Switched from slow base solver (quadprog) to sparse solvers (cvxopt and cvxpy) improving speed and flexibility
- Integrated recursive partitioning using center of gravity constraints to enhance global placement quality
- Cells snapped to legal rows with reduced overlaps
- Framework is modular scalable, and supports further enhancements
- Better placement means shorter nets which means fewer routing detours
    - Reduces congestion and via count
    - Improves timing closure

Thank you.