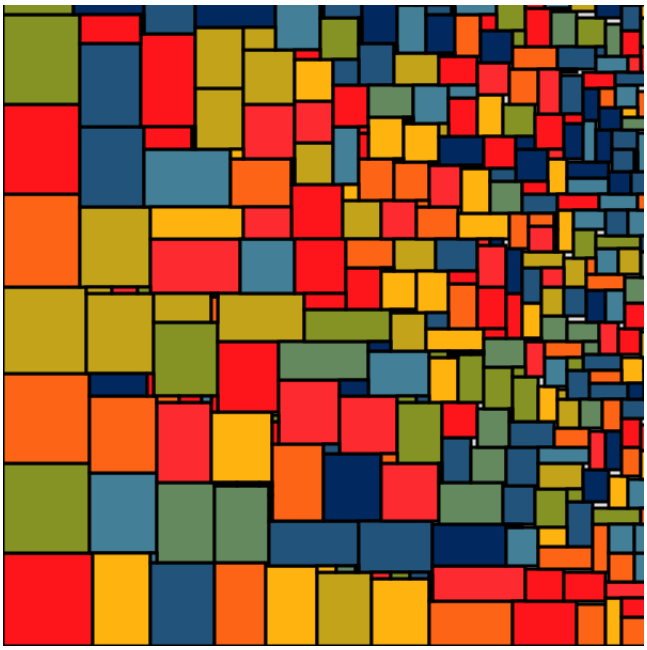
Alma Mater Studiorum
Università di Bologna

VLSI Design

LP Solution



Yuri Noviello yuri.noviello@studio.unibo.it
Francesco Olivo francesco.olivo2@studio.unibo.it

1 Introduction

VLSI (Very Large Scale Integration) refers to the trend of integrating circuits into silicon chips. A typical example is the smartphone. The modern trend of shrinking transistor sizes, allowing engineers to fit more and more transistors into the same area of silicon, has pushed the integration of more and more functions of cellphone circuitry into a single silicon die (i.e. plate). This enabled the modern cellphone to mature into a powerful tool that shrank from the size of a large brick-sized unit to a device small enough to comfortably carry in a pocket or purse, with a video camera, touchscreen, and other advanced features. As the combinatorial decision and optimization expert, the student is assigned to design the VLSI of the circuits defining their electrical device: given a fixed-width plate and a list of rectangular circuits, decide how to place them on the plate so that the length of the final device is minimized (improving its portability). Consider two variants of the problem. In the first, each circuit must be placed in a fixed orientation with respect to the others. This means that, an $n \times m$ circuit cannot be positioned as an $m \times n$ circuit in the silicon plate. In the second case, the rotation is allowed, which means that an $n \times m$ circuit can be positioned either as it is or as $m \times n$.

2 Format of the instances

This section describes the format in which the VLSI instances are written, as well as the expected format of the corresponding solutions.

2.1 Input format

An instance of VLSI is a text file consisting of lines of integer values. The first line gives w, which is the width of the silicon plate. The following line gives n, which is the number of necessary circuits to place inside the plate. Then n lines follow, each with x_i and y_i, representing the horizontal and vertical dimensions of the i-th circuit.
For example, a file with the following lines:

```
9
5
3 3
2 4
2 8
3 9
4 12
```

describes an instance in which the silicon plate has the width 9, and we need to place 5 circuits, with the dimensions 3×3, 2×4, 2×8, 3×9, and 4×12.

2.2 Solution format

Where to place a circuit i can be described by the position of i in the silicon plate. The solution should indicate the length of the plate l, as well as the position of each i by its $\hat{x}_i$ and $\hat{y}_i$, which are the coordinates of the left-bottom corner i. This could be done by for instance adding l next to w, and adding $\hat{x}_i$ and $\hat{y}_i$ next to x_i and y_i in the instance file. To exemplify, the solution of the previous example looks like:

```
9  12
5
3  3  4  0
2  4  7  0
2  8  7  4
3  9  4  3
4  12  0  0
```

Which says for instance that the left-bottom corner of the 3×3 circuit is at (4, 0). For the model with rotation the solution contains the (eventually) inverted dimensions of the rotated circuits.

3 Model

3.1 Variables

The starting point requires finding all the necessary variables to model the problem. In particular, since the output format requires the left bottom corner coordinates of a given rectangle, we need two arrays to store the x and the y coordinates, respectively p_x and p_y. These arrays must be sorted coherently with the original arrays x and y that state the dimensions of the rectangles, respectively the width and the height. All said arrays have length n, i.e. the number of circuits to be placed on the board.
We want to model all of our variables to have the smallest possible domain, in order to reduce the search space. Thus, the coordinates p_x and p_y will be bounded within $[0, w - min(x)]$ and $[0, l_{max} - min(y)]$ respectively.

3.2 Bounds for the height

Since the goal is to minimize the height, it is crucial to find a good pair of bounds for the height variable l. For the lower bound of l we considered that if all the plates can be inserted in the plate without leaving empty space, the height is given by the division between the summation of the areas and the fixed width. At the same time we need to consider the case in which there is a plate with a very large height, in this case the lower bound is given by the larger

height.

$$l_{min} = max(max(y), \lfloor \frac{\sum_{i=1}^{n} y_i * x_i}{w} \rfloor)$$

For the upper bound we approximated an estimator that (largely) estimate the value of l. We also noticed that the upper bound does not influence particularly the performances since the solver searches firstly for values near to the lower bound, since we need to minimize it.

$$l_{max} = \lfloor l_{min} + \frac{\sum_{i=1}^{n} y_i}{2} \rfloor$$

3.3 Objective Function

The objective function (the function that we want to minimize) is basically represented by the variable l.

$$objective\_function := minimize(l_{goal})$$

4 Constraints

There are many constraints to impose in order to correctly solve the problem: we want to respect the width and the height constraint, while trying to minimize the height. On the other hand we do not want circuits to overlap, and we want to avoid symmetries in order to reduce the search space.

4.1 Limits Constraint

All the rectangles must be places within the board limits.
This can be expressed as:

$$\begin{aligned} x_i + p_{x_i} &\leq w, & i \in [1, n] \\ y_i + p_{y_i} &\leq l, & i \in [1, n] \end{aligned}$$

4.2 No overlapping

To make that there is no overlapping between all the rectangles, at least one of the following constraint must hold:

$$\begin{aligned} p_{x_i} &\geq p_{x_j} + x_j, & i, j \in [1, n], \ i < j \\ p_{x_i} + x_i &\leq x_j, & i, j \in [1, n], \ i < j \\ p_{y_i} &\geq p_{y_j} + y_j, & i, j \in [1, n], \ i < j \\ p_{y_i} + y_i &\leq y_j, & i, j \in [1, n], \ i < j \end{aligned}$$

Since in LP does not exist any explicit OR implementation we can write the previous constraints as follows (big-M formulation [1]):

$$\begin{aligned} p_{x_i} &\geq p_{x_j} + x_j - M_1\delta_{1,i,j}, & i,j \in [1,n],\ \ i<j \\ p_{x_i} + x_i &\leq x_j + M_1\delta_{2,i,j}, & i,j \in [1,n],\ \ i<j \\ p_{y_i} &\geq p_{y_j} + y_j - M_2\delta_{3,i,j}, & i,j \in [1,n],\ \ i<j \\ p_{y_i} + y_i &\leq y_j - M_2\delta_{4,i,j}, & i,j \in [1,n],\ \ i<j \end{aligned}$$

with $M = [w, l_e]$, where w represents the width of the plate and l_e is an estimator of the height and $\delta_{k,i,j}$ contains all binary variables s.t.

$$\sum_k \delta_{k,i,j} \leq 3$$

4.3 Symmetries

The handling of the symmetries will be explained in the implementation section 6.2.2 since it is extremely related to the solver.

5 Rotation

We are required to create another model, where it is possible to rotate the circuits on the board with respect to the given dimensions.
To represent rotation, we need 2 binary array r_t and r_f such that $\forall i$ in $1..n$ $r_{t_i} + r_{f_i} = 1$. In this way r_{t_i} is equal to 1 if the i-th rectangle in the input is rotated, 0 otherwise.
Then, we can represent x_r and y_r (that represent the effective dimensions of a rectangle) as follows:

$$\begin{aligned} x_{r_i} &= x_i * r_{f_i} + y_i * r_{t_i} \\ y_{r_i} &= y_i * r_{f_i} + x_i * r_{t_i} \end{aligned}$$

6 Implementation choices

To solve this problem we decided to use `PuLP`[2], an LP modeler for Python. A very relevant feature of `PuLP` is that it is possible to define a problem and then try different different optimizers (Gurobi[3], CPLEX[4], etc.) to solve it without changing the syntax of the code.

6.1 GUROBI

`GUROBI` is a powerful mathematical programming solver available for LP, QP and MIP (MILP, MIQP, and MIQCP). Unfortunately due to technical issues

we could not get the full license of Gurobi but only the free one, for this reason our model with this solver can produce only the solution for the first 31 instances since the for the following ones there are too many variables. It failed only 4 instances with no rotation and 6 with rotation.
In the following tables we reported the results of the model. The **Error** columns contain the difference between the optimal solution and the found one.

No rotation model:

Name	**Time**	**Error**
ins-01	0.03673	0
ins-02	0.00396	0
ins-03	0.00545	0
ins-04	0.00645	0
ins-05	0.01174	0
ins-06	0.02171	0
ins-07	0.01401	0
ins-08	0.00498	0
ins-09	0.01821	0
ins-10	0.10539	0
ins-11	0.45203	0
ins-12	0.19703	0
ins-13	0.22079	0
ins-14	0.32051	0
ins-15	0.32468	0
ins-16	300	1
ins-17	1.21487	0
ins-18	37.50279	0
ins-19	7.78832	0
ins-20	4.81069	0
ins-21	300	1
ins-22	161.18236	0
ins-23	3.02602	0
ins-24	1.25084	0
ins-25	300	1
ins-26	198.67611	0
ins-27	8.53881	0
ins-28	2.51877	0
ins-29	1.33688	0
ins-30	300	1
ins-31	1.09829	0

Rotation model:

Name	**Time**	**Error**
ins-01	0.00334	0
ins-02	0.00545	0
ins-03	0.00512	0
ins-04	0.01415	0
ins-05	0.05520	0
ins-06	0.06069	0
ins-07	0.04004	0
ins-08	0.03005	0
ins-09	0.14633	0
ins-10	0.32638	0
ins-11	0.54734	0
ins-12	2.27934	0
ins-13	0.61387	0
ins-14	3.22381	0
ins-15	1.12315	0
ins-16	14.66644	0
ins-17	5.45880	0
ins-18	50.48112	0
ins-19	300	1
ins-20	300	1
ins-21	300	1
ins-22	300	1
ins-23	14.30323	0
ins-24	1.29263	0
ins-25	300	1
ins-26	84.46380	0
ins-27	250.22625	0
ins-28	13.44658	0
ins-29	13.20913	0
ins-30	300	1
ins-31	1.94747	0

6.2 CPLEX

IBM ILOG CPLEX Optimization Studio is a prescriptive analytics solution that enables rapid development and deployment of decision optimization models using mathematical and constraint programming.
We obtained the academic license of CPLEX and we obtained good results. `PuLP` offers two different solvers for CPLEX:

- `CPLEX_PY`: The CPLEX LP/MIP solver (via a Python Binding) ;
- `CPLEX_CMD`: The CPLEX LP solver (requires the full path of the executable).

The main difference between them is that only `CPLEX_CMD` supports additional options to pass to solver. In the CPLEX documentation we found the *symmetry breaking* option that allows CPLEX to detect symmetries in the model. So we used `CPLEX_PY` to find a solution without any symmetry breaking detection and `CPLEX_CMD` to add this feature to the model.

6.2.1 CPLEX_PY

With the solver `CPLEX_PY` we reached good results. We solved 29 instances with the model without rotation and 23 instances with rotation. Anyway, when we cannot find the optimal solution the model returns a solution that is very close to the optimal one. For example, in the following images we can observe that in the instance 39 our model returns $l = 61$ without rotation and $l = 62$ with rotation, instead of the optimal value that is 60.

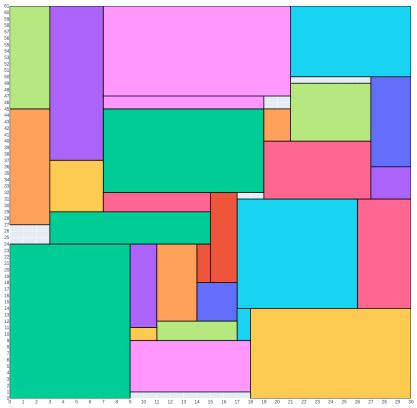


Figure 1: ins-39 with no rotation

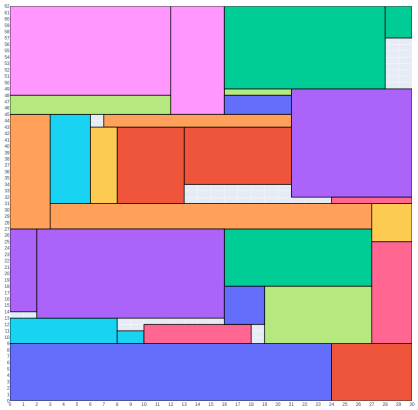


Figure 2: ins-39 with rotation

In the following tables we reported the results of the model with `CPLEX_PY`.

No rotation model:

Name	Time	Error
ins-01	0.02767	0
ins-02	0.01196	0
ins-03	0.01889	0
ins-04	0.01751	0
ins-05	0.05895	0
ins-06	0.04485	0
ins-07	0.02879	0
ins-08	0.15674	0
ins-09	0.03643	0
ins-10	0.16746	0
ins-11	70.11056	0
ins-12	0.16737	0
ins-13	0.58594	0
ins-14	0.22293	0
ins-15	4.22194	0
ins-16	20.41389	0
ins-17	13.80992	0
ins-18	4.37256	0
ins-19	207.6478	0
ins-20	80.2606	0
ins-21	300	1
ins-22	300	1
ins-23	11.8249	0
ins-24	1.93177	0
ins-25	300	1
ins-26	16.89525	0
ins-27	5.37518	0
ins-28	6.82104	0
ins-29	18.03386	0
ins-30	300	2
ins-31	5.15963	0
ins-32	300	2
ins-33	1.26341	0
ins-34	300	1
ins-35	33.77486	0
ins-36	300	1
ins-37	300	1
ins-38	300	1
ins-39	300	1
ins-40	300	18

Rotation model:

Name	Time	Error
ins-01	0.03355	0
ins-02	0.01868	0
ins-03	0.02505	0
ins-04	0.01991	0
ins-05	0.12191	0
ins-06	0.03137	0
ins-07	0.0314	0
ins-08	0.04968	0
ins-09	0.0615	0
ins-10	0.1672	0
ins-11	2.16002	0
ins-12	0.83146	0
ins-13	0.94543	0
ins-14	35.454	0
ins-15	130.86692	0
ins-16	300	1
ins-17	86.17698	0
ins-18	179.11631	0
ins-19	300	1
ins-20	124.95276	0
ins-21	300	1
ins-22	300	1
ins-23	300	1
ins-24	79.09807	0
ins-25	300	2
ins-26	300	1
ins-27	107.38966	0
ins-28	300	1
ins-29	300	1
ins-30	300	1
ins-31	300	1
ins-32	300	2
ins-33	300	1
ins-34	50.93315	0
ins-35	209.32795	0
ins-36	83.49632	0
ins-37	300	1
ins-38	300	1
ins-39	300	2
Ins-40	300	13

6.2.2 CPLEX_CMD

Also with the `CPLEX_CMD` solver we obtained good results. We solved 31 instances with the model without rotation and 21 instances with rotation. Also in this case when we cannot find the optimal solution we are very close to it.
Following there are some images that represent the solution found by our model:

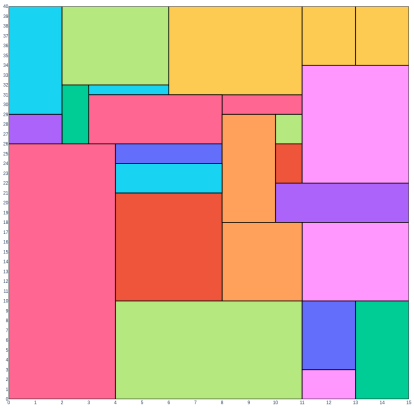


Figure 3: ins-35 with no rotation

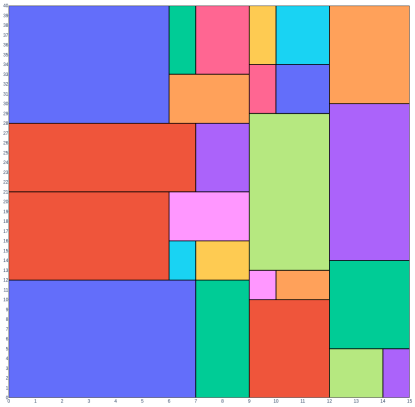


Figure 4: ins-36 with no rotation

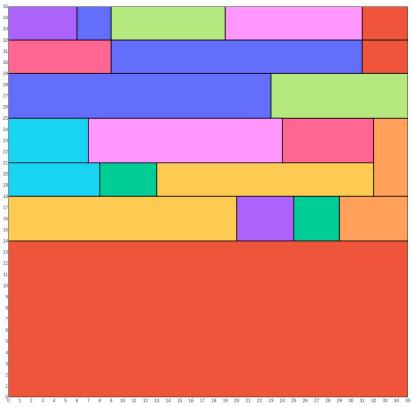


Figure 5: ins-28 with rotation

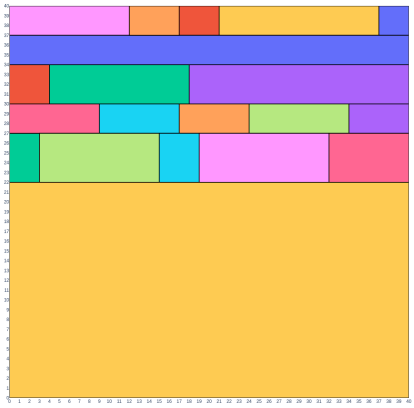


Figure 6: ins-33 with rotation

In the following tables we reported the results of the model with `CPLEX_CMD` using the option *(set preprocessing symmetry 5)*.

No rotation model:

Name	Time	Error
ins-01	0.04557	0
ins-02	0.01693	0
ins-03	0.04794	0
ins-04	0.04746	0
ins-05	0.07605	0
ins-06	0.09919	0
ins-07	0.07184	0
ins-08	0.08973	0
ins-09	0.09798	0
ins-10	0.24284	0
ins-11	57.77396	0
ins-12	0.79727	0
ins-13	0.7918	0
ins-14	1.52282	0
ins-15	0.99688	0
ins-16	11.97966	0
ins-17	5.13692	0
ins-18	6.90408	0
ins-19	41.62143	0
ins-20	199.93819	0
ins-21	107.61329	0
ins-22	300	1
ins-23	114.49582	0
ins-24	0.56408	0
ins-25	300	1
ins-26	300	1
ins-27	5.83222	0
ins-28	95.85853	0
ins-29	9.06915	0
ins-30	300	2
ins-31	261.72628	0
ins-32	300	2
ins-33	2.8326	0
ins-34	46.73709	0
ins-35	18.20331	0
ins-36	15.15149	0
ins-37	300	3
ins-38	300	2
ins-39	300	1
ins-40	300	19

Rotation model:

Name	Time	Error
ins-01	0.04018	0
ins-02	0.04561	0
ins-03	0.0572	0
ins-04	0.05923	0
ins-05	0.21187	0
ins-06	0.13125	0
ins-07	0.06396	0
ins-08	0.08356	0
ins-09	0.10474	0
ins-10	0.17665	0
ins-11	182.65107	0
ins-12	1.15443	0
ins-13	0.42483	0
ins-14	11.44502	0
ins-15	300	1
ins-16	300	1
ins-17	2.13807	0
ins-18	8.80146	0
ins-19	300	1
ins-20	31.23436	0
ins-21	300	1
ins-22	300	1
ins-23	300	1
ins-24	25.40178	0
ins-25	300	1
ins-26	300	1
ins-27	38.84125	0
ins-28	4.5719	0
ins-29	300	1
ins-30	300	2
ins-31	300	1
ins-32	300	2
ins-33	10.05414	0
ins-34	300	1
ins-35	300	1
ins-36	300	1
ins-37	300	1
ins-38	300	1
ins-39	300	1
ins-40	300	13

7 Final results

In our opinion the best model seems to be the one with `GUROBI` as solver, as we can see from the following data:

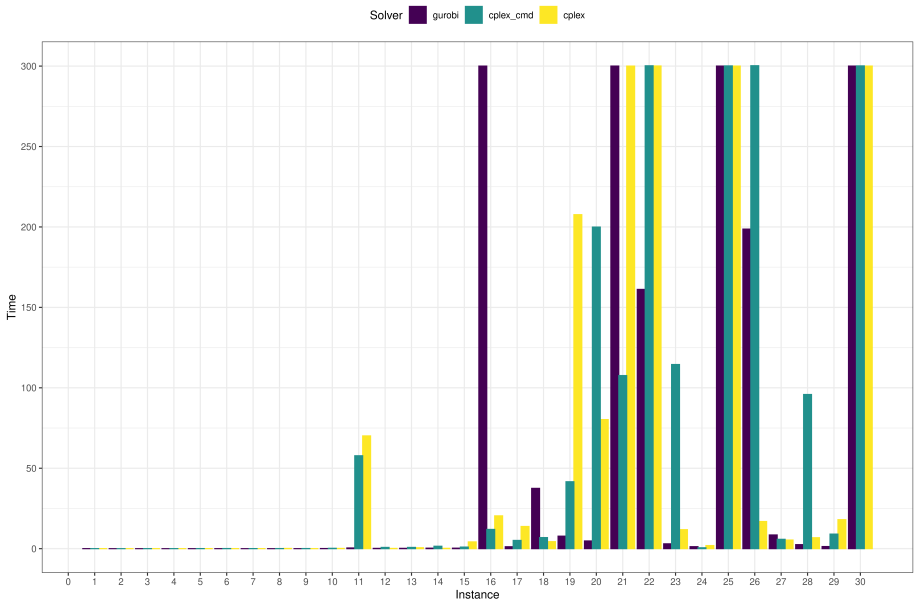


Figure 7: Execution time for different solvers

solver	avg_time	median_time	sd_time	found	optimal_found
GUROBI	54.323	0.833	108.049	30	26
CPLEX_PY	55.447	4.297	105.669	30	26
CPLEX_CMD	62.081	3.329	105.711	30	26

Unfortunately since we cannot run all the instances with our license we decided to use `CPLEX_PY` as final solver. Indeed, the `CPLEX` solvers are very similar but we preferred to solve more instances with rotation.

solver	model	avg_time	median_time	sd_time	found	optimal_found
CPLEX_CMD	basic	92.701	10.524	125.859	40	31
CPLEX_PY	basic	95.102	9.323	132.599	40	29
CPLEX_CMD	rotation	150.514	110.746	146.966	40	21
CPLEX_PY	rotation	154.799	127.910	135.549	40	23

Finally, in the last images we can observe the performances of our final model:

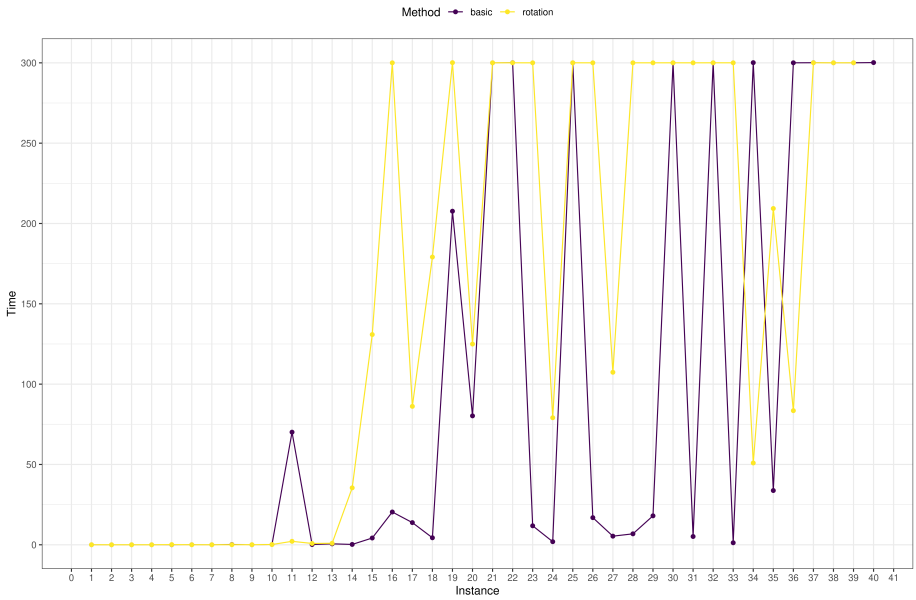


Figure 8: Comparison between basic and rotation models

References

[1] Erwin Kalvelagen. "Tiling Squares". In: (2022). URL: http://amsterdamoptimization.com/pdf/tiling.pdf.

[2] Stuart Mitchell, Stuart Mitchell Consulting, and Iain Dunning. *PuLP: A Linear Programming Toolkit for Python*. 2011.

[3] Gurobi Optimization, LLC. *Gurobi Optimizer Reference Manual*. 2022. URL: https://www.gurobi.com.

[4] IBM. "IBM ILOG CPLEX Optimization Studio". In: (). URL: https://www.ibm.com/products/ilog-cplex-optimization-studio.