GEQO_SAIO_report

June 26, 2015

1 SAIO vs GEQO comparison - report

1.1 Introduction

This is an aggregated report on comparison of two alternative join order optimizers:

- GEQO - Genetic Query Optimizer (current default optimizer in Postgres)
- SAIO - Simulated Annealing based Join Order optimizer (PGXN extension for Postgres)

The tests compare query costs, optimizing time and memory used.

1.2 Methodology

GEQO and SAIO are tested on many different schemas and queries. To ensure proper representation of each join order optimizer abilities, each query has been tested on a range of parameters:

- GEQO has been tested with GEQO effort in range 0 to 10
- SAIO has been tested with different combinations of:
  - equilibrium factor (determines the number of equilibrium loops)
  - temperature reduction (temperature is multiplied by this fraction in each iteration until the system freezes)

Some parameters of SAIO were fixed to reduce testing time:

- initial temperature (affects initial randomness of the algorithm and the total number of iterations)
- steps before freeze

FROM_COLLAPSE_LIMIT and JOIN_COLLAPSE_LIMIT have been set to 200.

1.3 Testing data

The testing data (DB schemas and queries) has been divided into following sections:

1.3.1 Provided data

SQL scripts that were written by hand. Mostly for the previous iteration of SAIO.

1.3.2 Programatically generated data

SQL scripts automatically generated by Python scripts
Schemas:

- 20-150 tables without constraints
- 20-150 tables with constraints

Flat queries:

- 15-20 JOINS (JOINS, LEFT JOINS, RIGHT JOINS, mixed)
- 20 JOINS (. . .)
- 30 JOINS
- 50 JOINS
- 70 JOINS
- 100 JOINS
- 130 JOINS

Nested queries:

- 15-20 JOINS (LEFT, RIGHT, mix)
- 20-25 JOINS
- 30-35 JOINS
- 50-60 JOINS
- 70-80 JOINS

1.3.3 Industrial examples

This would be awesome to include such examples. No examples yet.

The biggest query that I found during the search was on StackOverflow: http://stackoverflow.com/questions/15769643/postgresql-query-optimization-with-many-left-joins?rq=1, which had 13 joins.

2 Test summary

3 Conclusions

This report presents the behaviour of SAIO and GEQO for a number of queries. It doesn't show superiority of SAIO however it indicates that SAIO is worth further development and can possibly replace GEQO for more complicated queries.

There is number of cases where SAIO outperforms GEQO in terms of estimated total query cost. Unfortunately SAIO often produces worse results in terms of total cost, used memory and optimizing time.

3.1 Cases where GEQO clearly wins:

In these cases GEQO constantly creates plans with lower cost, needs less time and memory.

- Queries with cartesian joins such as:

```
SELECT * FROM tab_1, tab_2 ... tab_N
    WHERE (tab_1.col_1 = tab_2.col_1) AND (..) AND (tab_1.col_1 = tab_N.col_1);
```

- Queries with many JOINS and LEFT JOINS such as:

```
SELECT * FROM tab_1
    [LEFT]JOIN tab_2 on tab_1.col_1 = tab_2.col_2
    ...
    [LEFT]JOIN tab_N on tab_N.col_X = tab_Y.col_Z;
```

3.2 Cases where SAIO wins

Here we can observe that SAIO constantly finds solutions with lower cost.

- Queries with many RIGHT JOINS such as:

```
SELECT * FROM tab_1
    RIGHT JOIN tab_2 on tab_1.col_1 = tab_2.col_2
    ...
    RIGHT JOIN tab_N on tab_N.col_X = tab_Y.col_Z;
```

- Nested queries:

```
    SELECT * FROM tab_1
    [LEFT/RIGHT/] JOIN (
        SELECT * FROM tab_2
            [LEFT/RIGHT/] JOIN tab_3.col_A = tab_2.col_B
            ...
            [LEFT/RIGHT/] JOIN tab_W.col_X = tab_Y.col_Z
    ) AS subquery_1 ON TRUE
    ...
    [LEFT/RIGHT/] JOIN (
    ) AS subquery_N ON TRUE;
```

This benchmark should be treated with a grain of salt as it doesn't cover all the possible cases (it can be improved with some suggestions). For a production system, join order optimizer should be chosen after checking it's performance on a few expected query types.

4 Test details

In this section one can find more detailed information about the testing process - schemas and queries used. Also graphs that highlight performance of each of the algorithms.

4.1 Provided queries

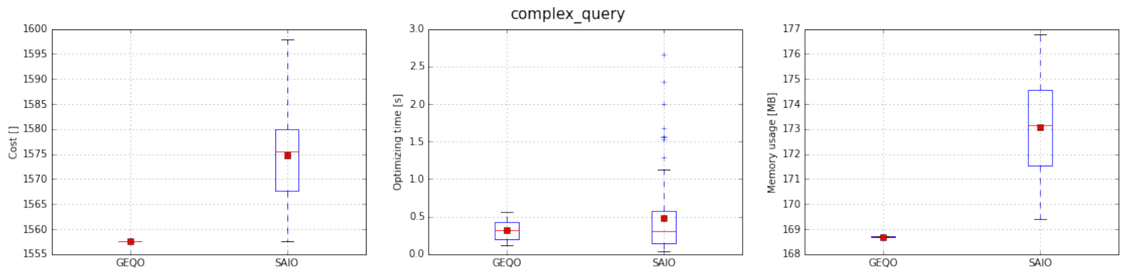
These queries were provided to test the previous iteration of SAIO back in 2010.

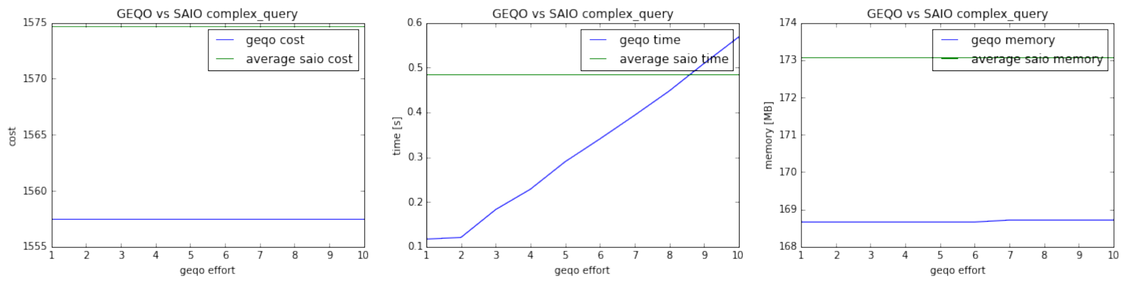
4.1.1 MODERATELY COMPLEX QUERY

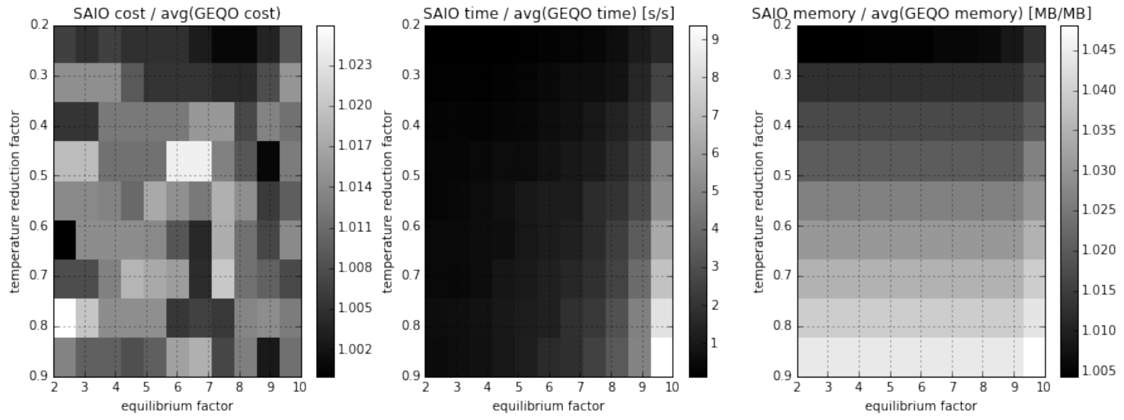
Here we can see that SAIO is is finding solutions comparable to GEQO. The optimizing time may be comparable to GEQO depending on the parameters of the algorithms. SAIO requires more memory than GEQO.

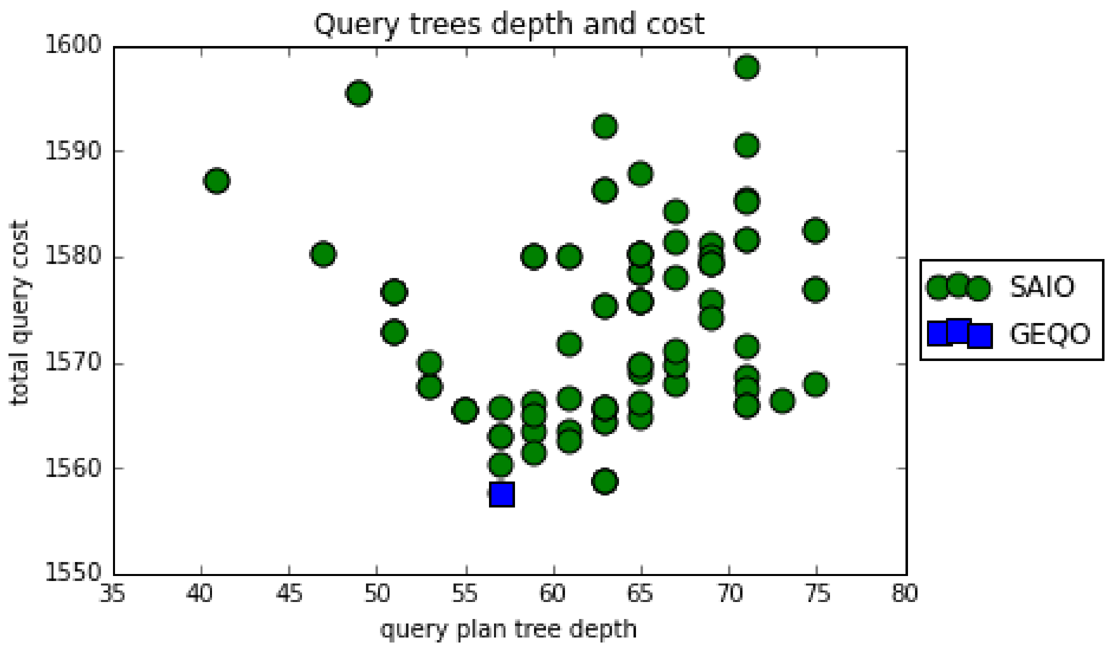
What's interesting - looking at 'query cost vs tree depth' plot, we can see that while SAIO explores a big number of different trees, GEQO is stuck near just one solution.

```
In [3]: display_info_for_test_case('complex_query')
```



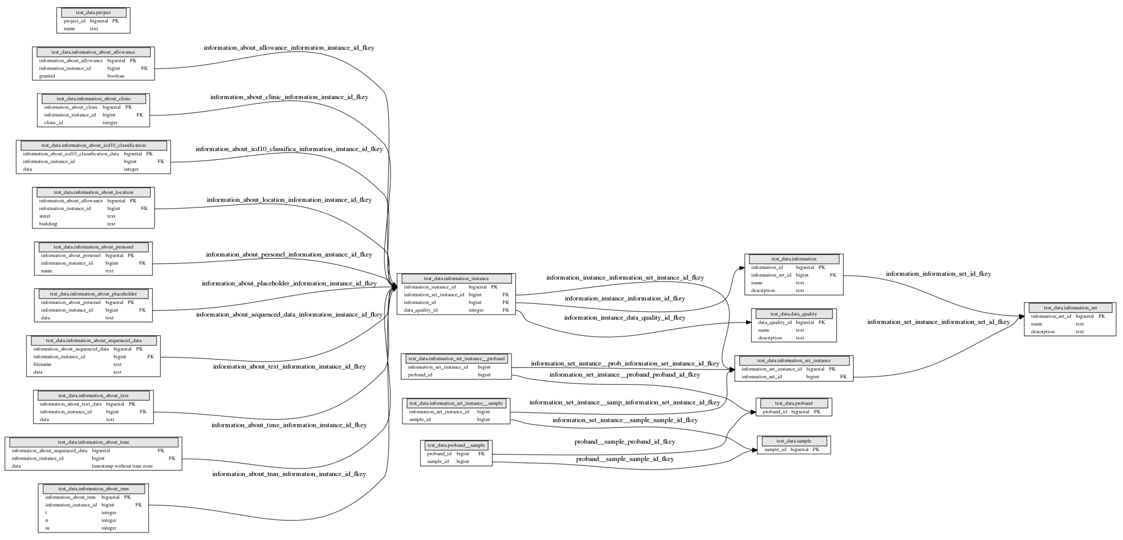

GEQO vs SAIO complex_query
geqo cost
average saio cost
cost
geqo effort
GEQO vs SAIO complex_query
geqo time
average saio time
time [s]
geqo effort
GEQO vs SAIO complex_query
geqo memory
average saio memory
memory [MB]
geqo effort


SAIO cost / avg(GEQO cost)
temperature reduction factor
equilibrium factor
SAIO time / avg(GEQO time) [s/s]
temperature reduction factor
equilibrium factor
SAIO memory / avg(GEQO memory) [MB/MB]
temperature reduction factor
equilibrium factor


Query trees depth and cost
total query cost
query plan tree depth
SAIO
GEQO

4.1.2 COMPLEX QUERY

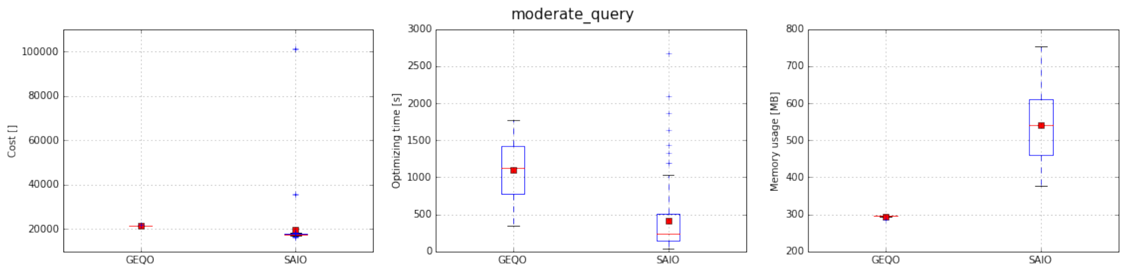
This query is based on a schema with multiple constraints.

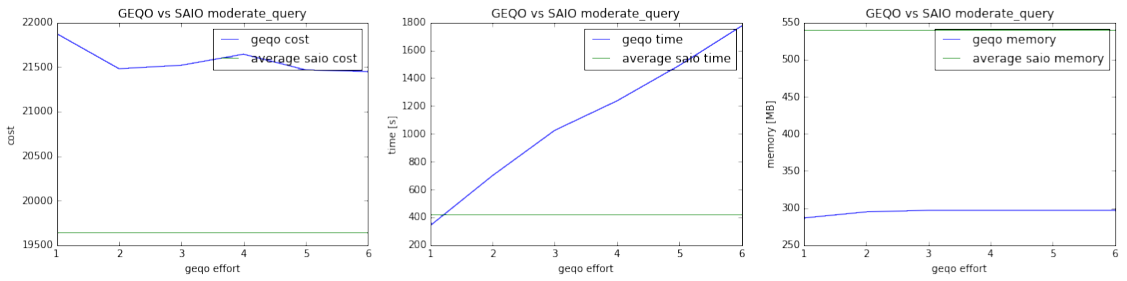


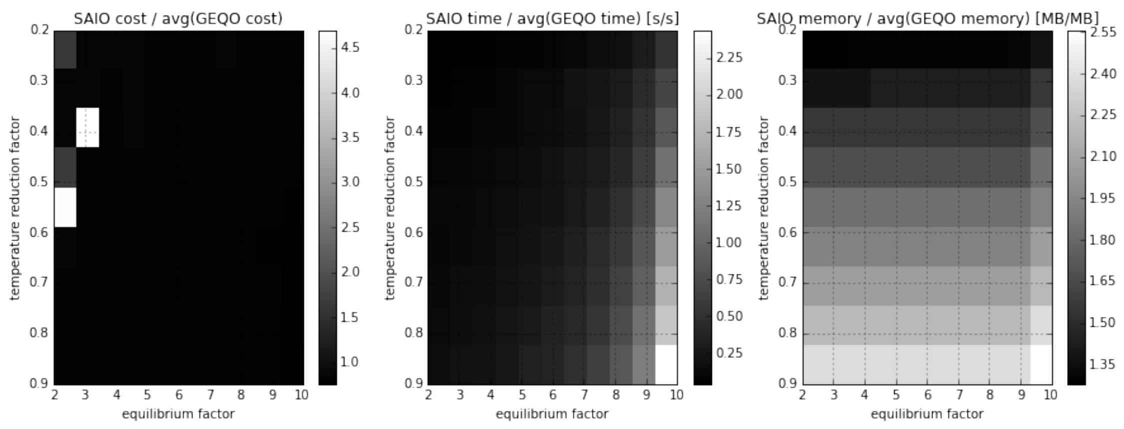
There are 5 views created on this schema. These views contain over 360 JOINS of different kind. The final query selects data from these views joining them with tables from the schema using 13 JOINS.

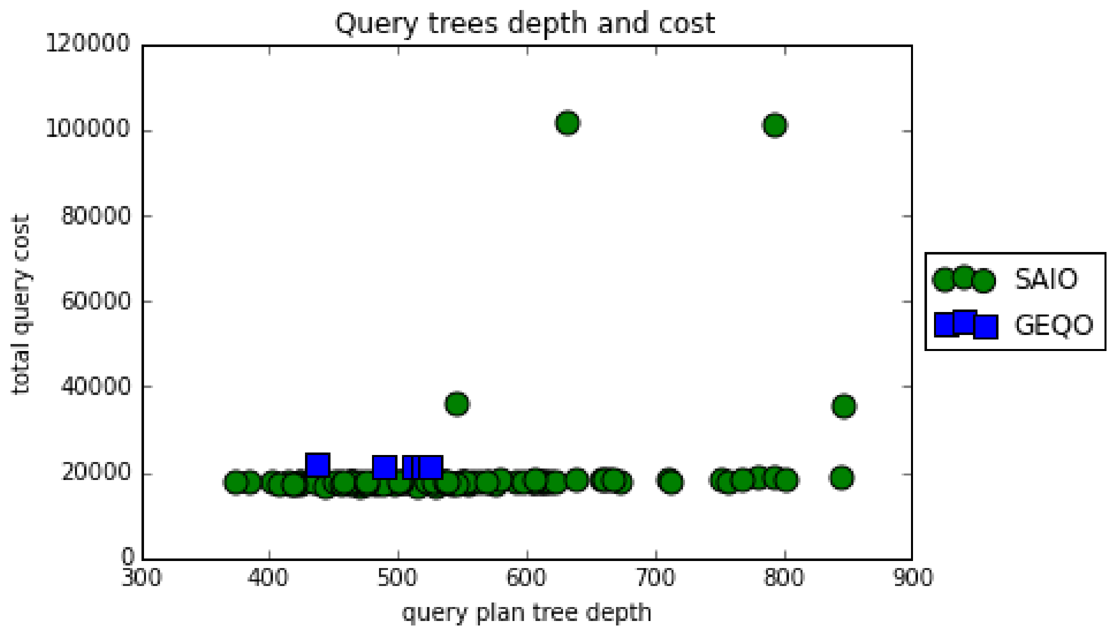
Here SAIO clearly outperforms GEQO finding join order with lower cost in shorter time.

```
In [4]: import sys
        sys.setrecursionlimit(2000) # for finding depth of a very deep python dict
        display_info_for_test_case('moderate_query')
```









4.2 Programatically generated queries

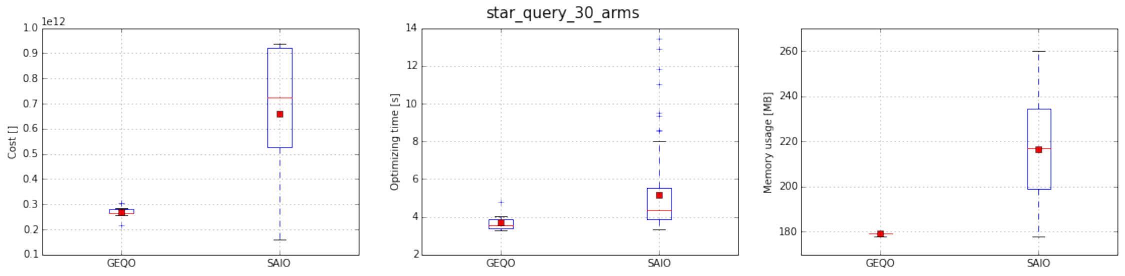
4.2.1 star query - cartesians

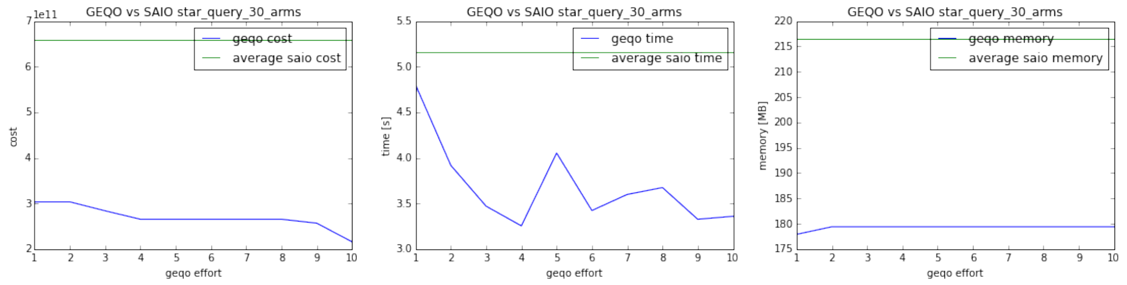
These queries look like this:

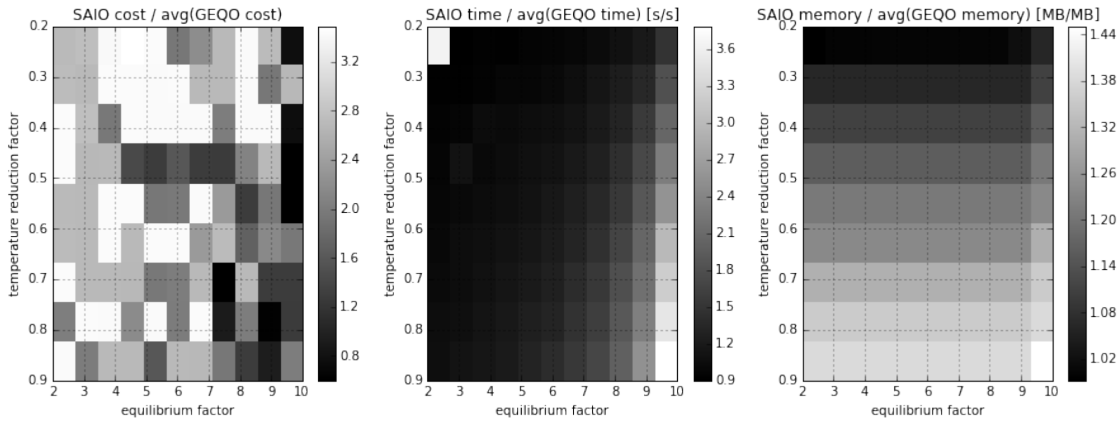
```
SELECT * FROM tab_1, tab_2 ... tab_N
    WHERE (tab_1.col_1 = tab_2.col_1) AND (..) AND (tab_1.col_1 = tab_N.col_1);
```

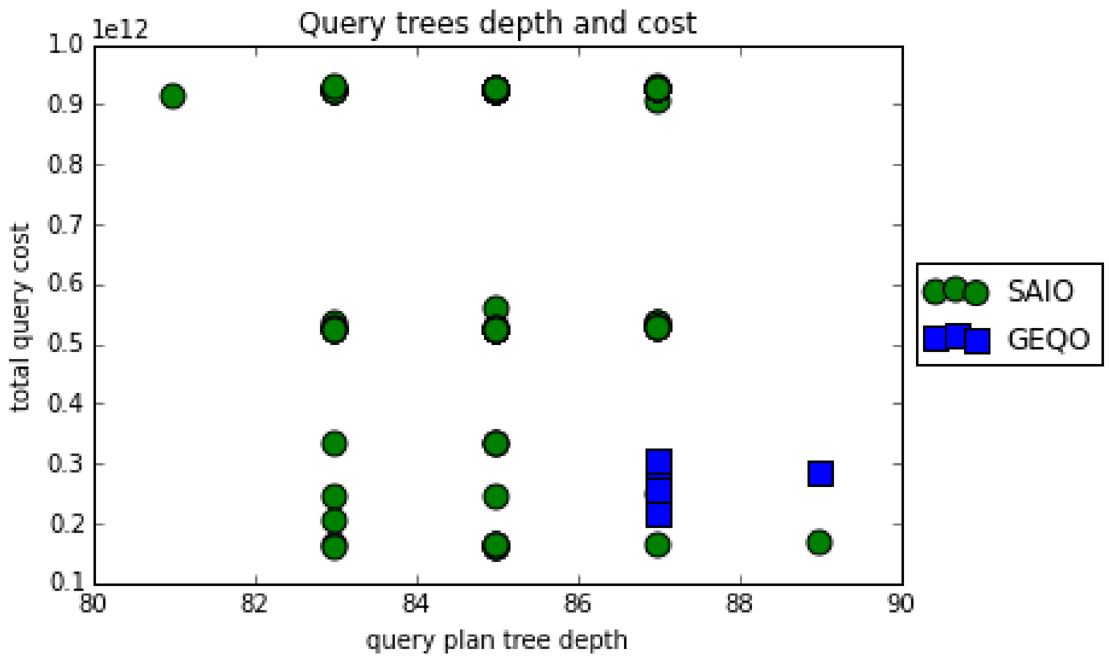
30 arms The best solutions here were found by SAIO, however average solution found by GEQO was much better than average SAIO solution. SAIO also needed more time and more memory to optimize the table order for this query.

```
In [16]: display_info_for_test_case('star_query_30_arms')
```


star_query_30_arms
1e12
Cost []
Optimizing time [s]
Memory usage [MB]
GEQO
SAIO

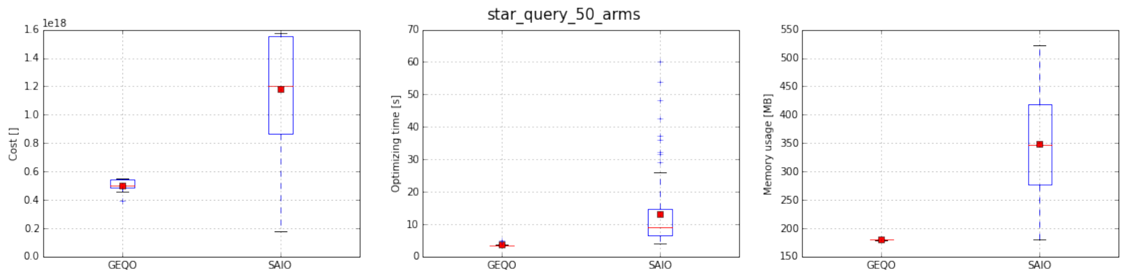

GEQO vs SAIO star_query_30_arms
1e11
cost
geqo cost
average saio cost
time [s]
geqo time
average saio time
memory [MB]
geqo memory
average saio memory
geqo effort

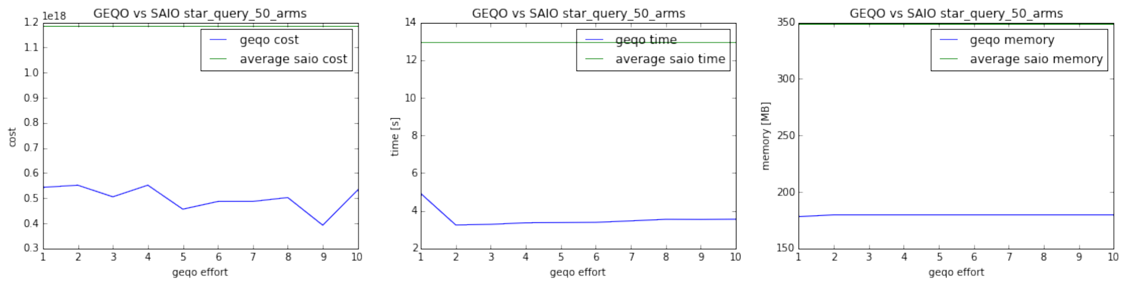

SAIO cost / avg(GEQO cost)
SAIO time / avg(GEQO time) [s/s]
SAIO memory / avg(GEQO memory) [MB/MB]
temperature reduction factor
equilibrium factor

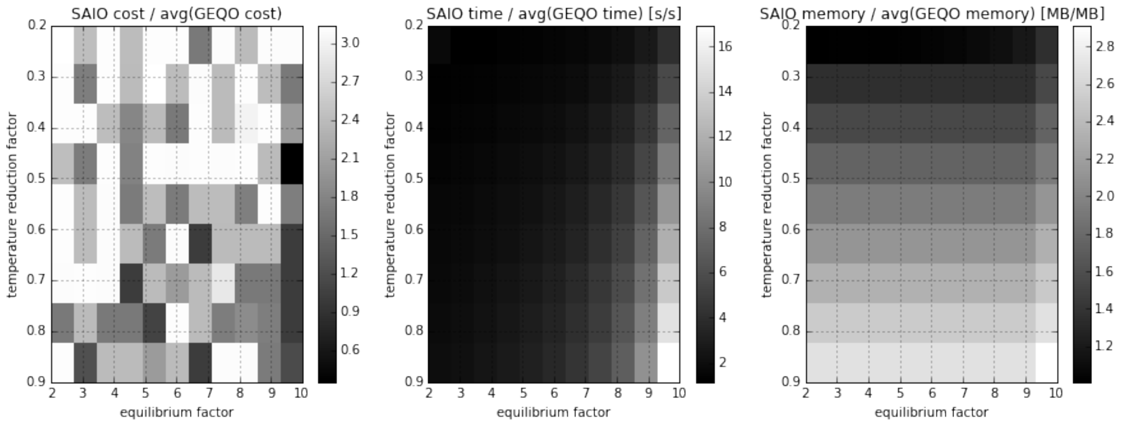


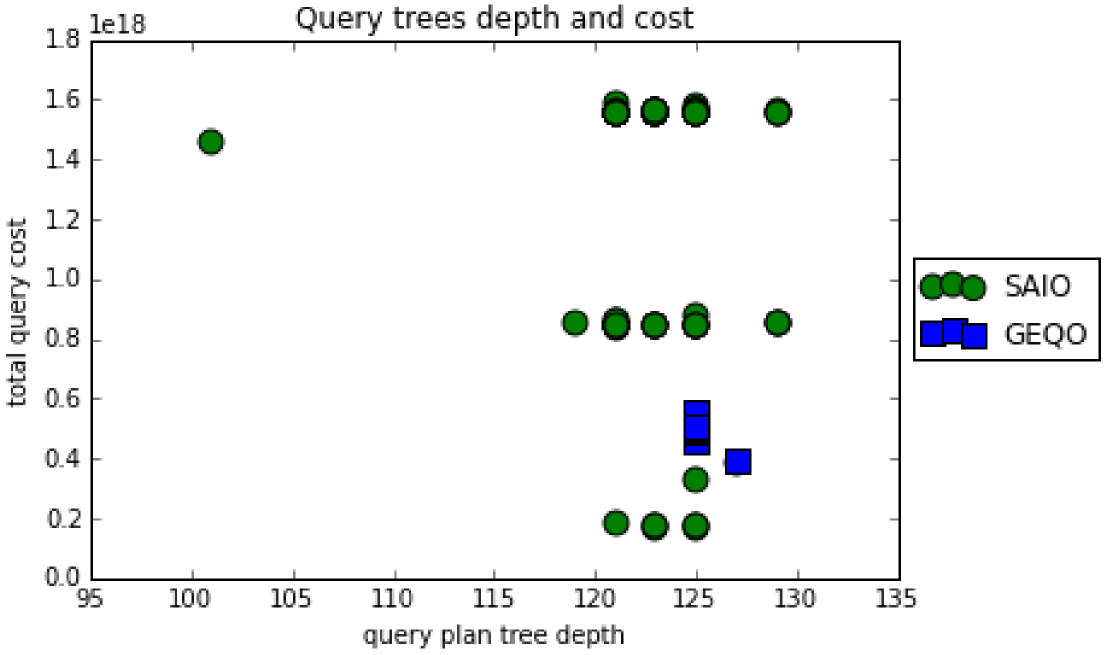
50 arms The same holds for this query.

```
In [17]: display_info_for_test_case('star_query_50_arms')
```



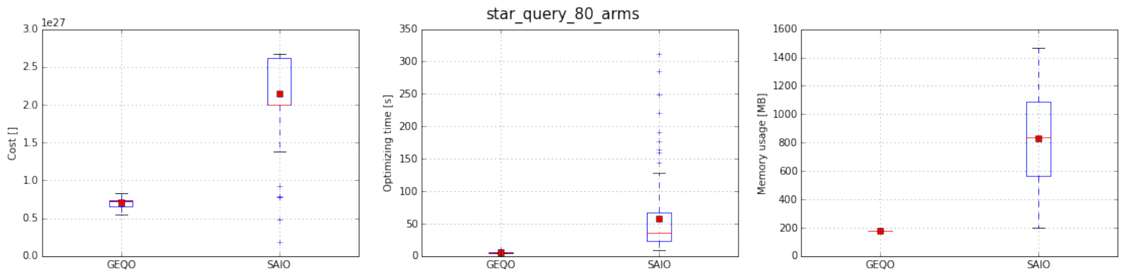


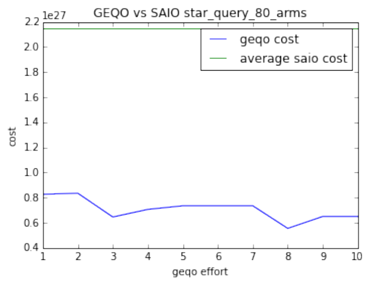
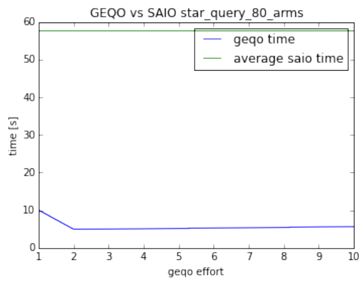
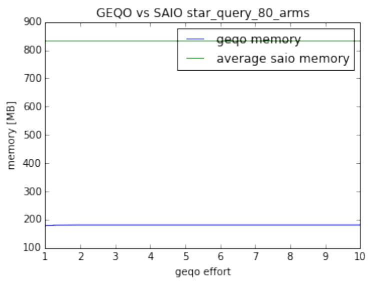
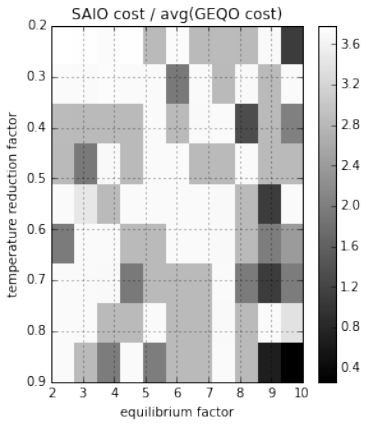
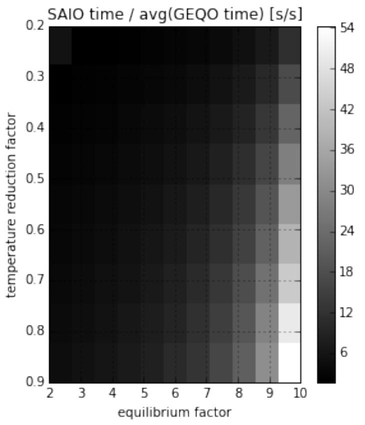
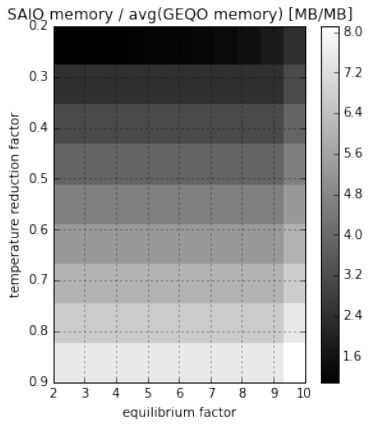
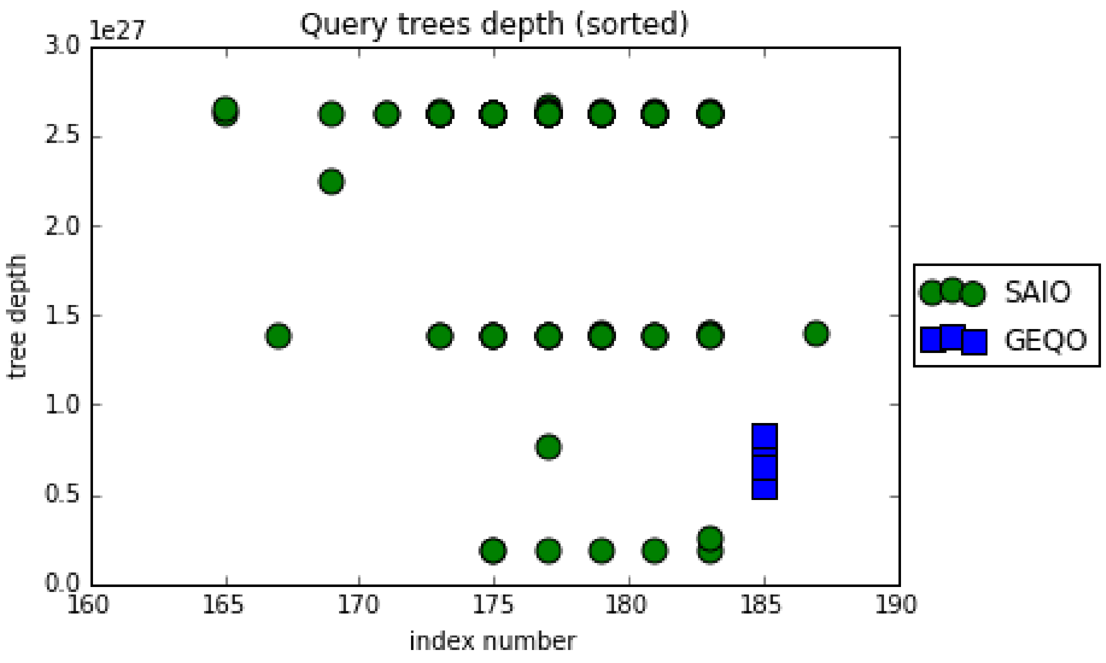




80 arms The result is the same as for the previous queries. GEQO finds better average solution than SAIO in shorter time using much less memory.

```
In [13]: display_info_for_test_case('star_query_80_arms')
```




GEQO vs SAIO star_query_80_arms
1e27
cost
geqo effort
geqo cost
average saio cost

GEQO vs SAIO star_query_80_arms
time [s]
geqo effort
geqo time
average saio time

GEQO vs SAIO star_query_80_arms
memory [MB]
geqo effort
geqo memory
average saio memory

SAIO cost / avg(GEQO cost)
temperature reduction factor
equilibrium factor

SAIO time / avg(GEQO time) [s/s]
temperature reduction factor
equilibrium factor

SAIO memory / avg(GEQO memory) [MB/MB]
temperature reduction factor
equilibrium factor

Query trees depth (sorted)
1e27
tree depth
index number
SAIO
GEQO

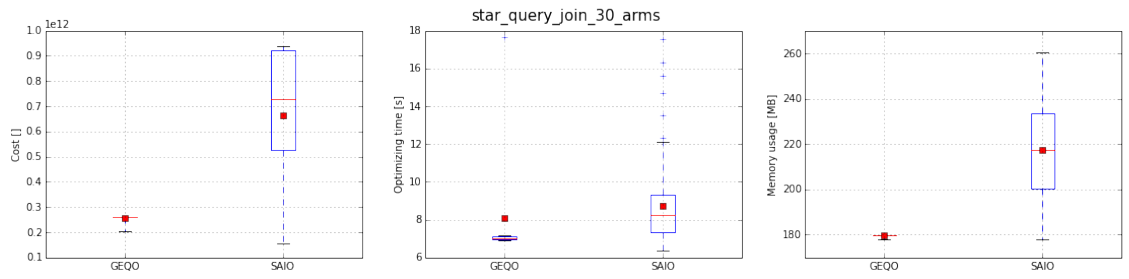
4.2.2 star query - joins

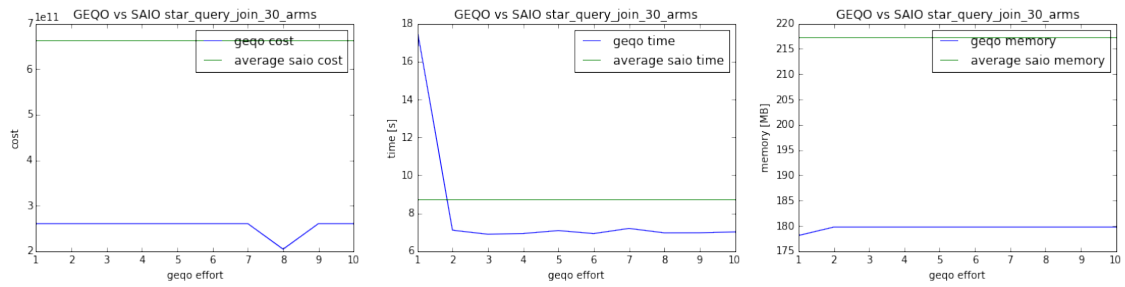
These queries look like:

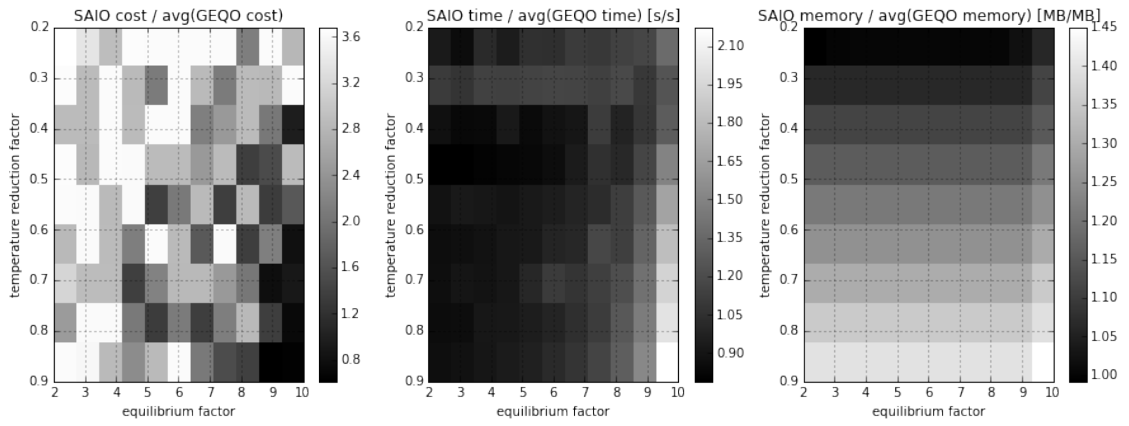
```
SELECT * FROM center JOIN arm0 ON (arm0.col = center.col0 or arm0.col = center.col0)
 JOIN arm1 ON (arm1.col = center.col1 or arm1.col = center.col2)
 ...
 JOIN armN ON (armN.col = center.colN or armM.col = center.colN);
```

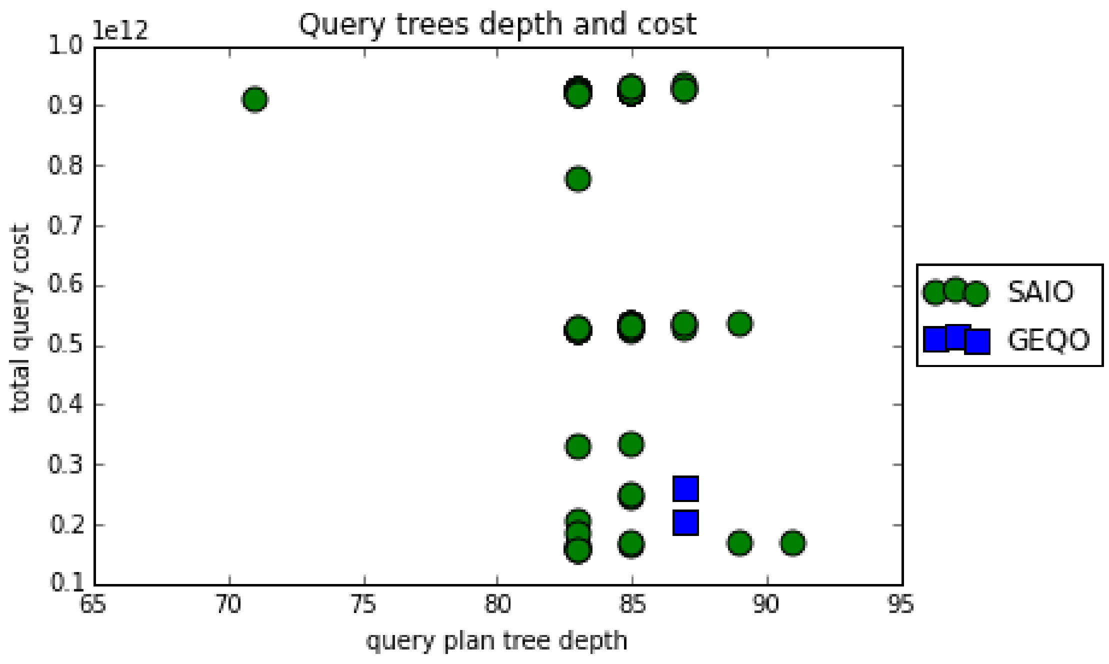
30 arms Still the same conclusions as for previous cartesian queries. GEQO wins.

```
In [18]: display_info_for_test_case('star_query_join_30_arms')
```



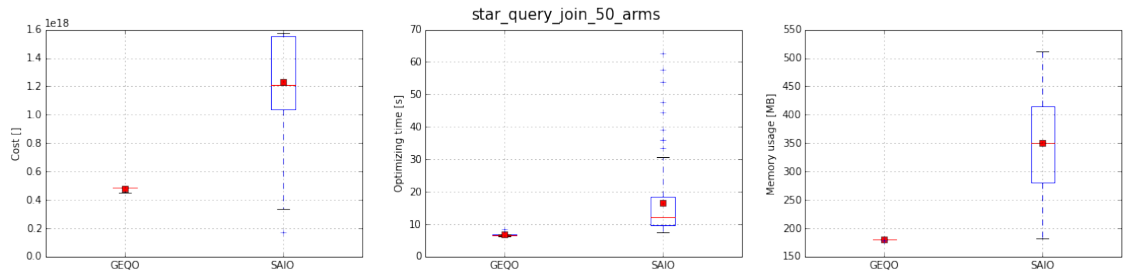


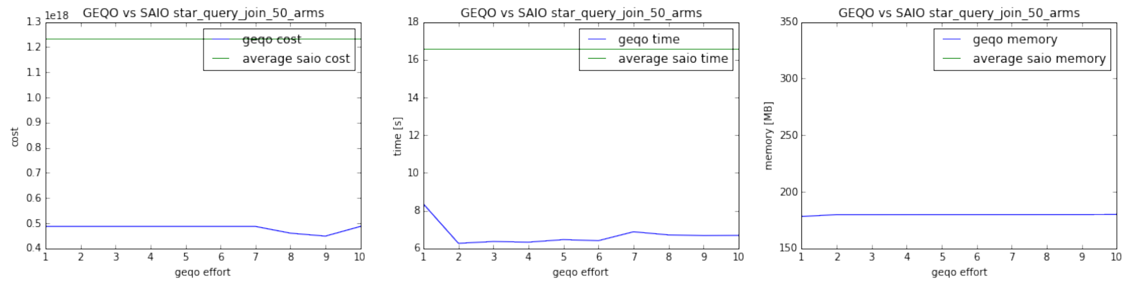


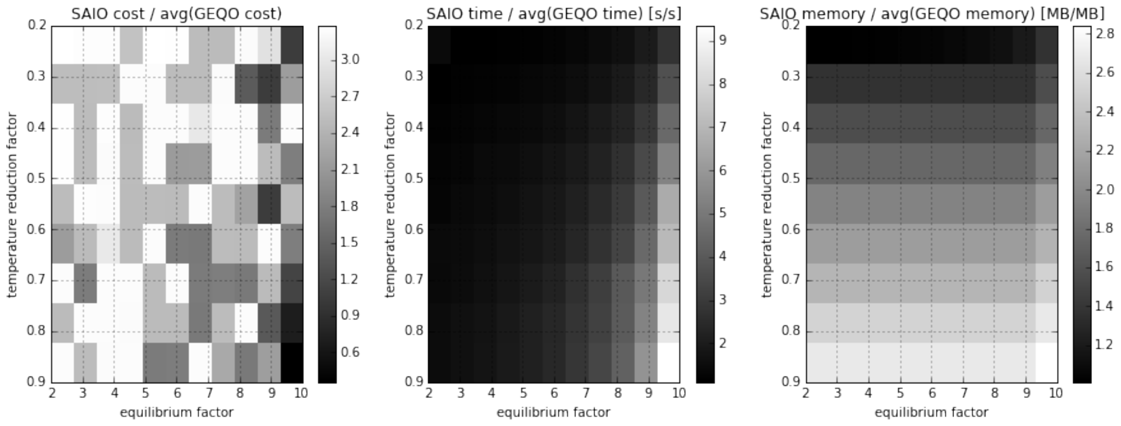


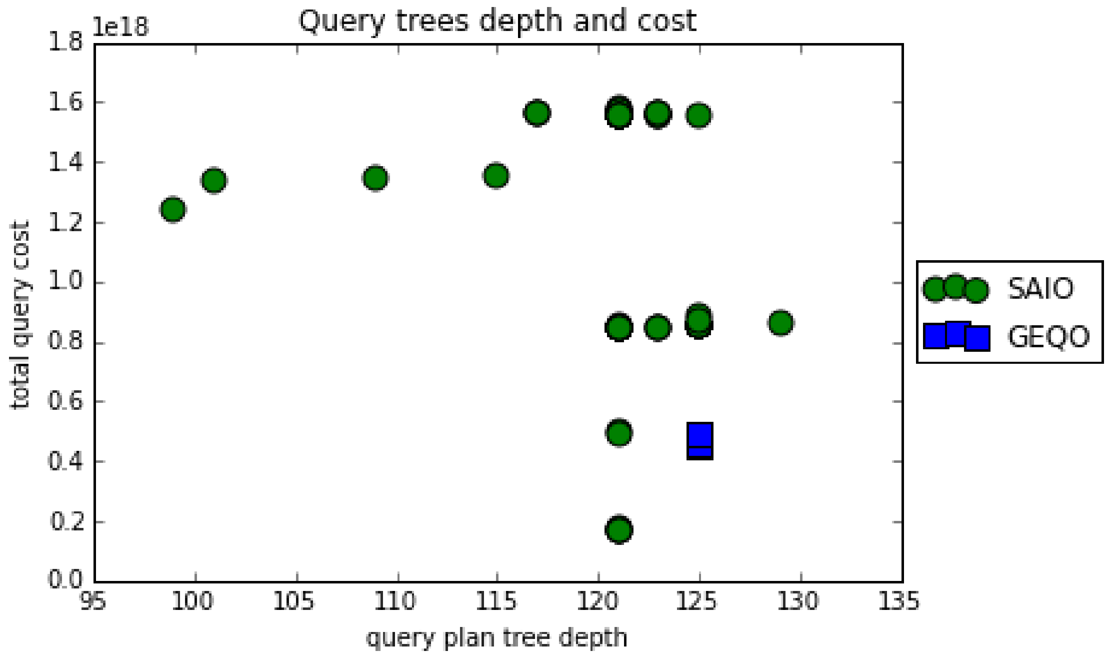
50 arms

```
In [19]: display_info_for_test_case('star_query_join_50_arms')
```









4.2.3 flat queries

```
SELECT * FROM  table_0
    JOIN/LEFT JOIN/RIGHT JOIN table_random ON table_random.col_random = table_random2.col_random2
    ...
    JOIN/LEFT JOIN/RIGHT JOIN table_random ON table_random.col_random = table_random2.col_random2;
```

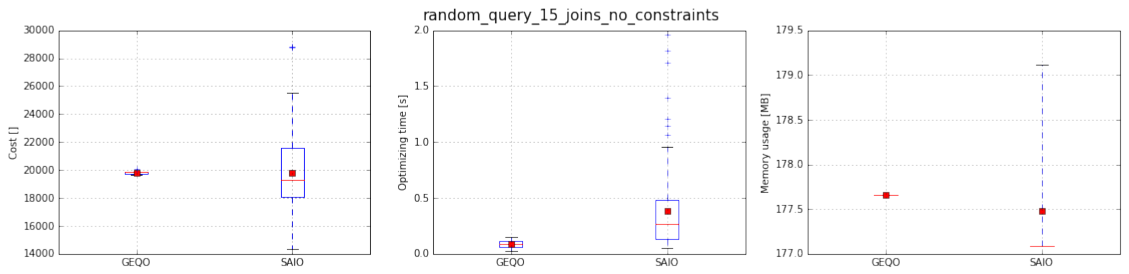
15 JOINS (JOINS, LEFT JOINS, RIGHT JOINS) For queries with 15 JOINS we can see that the results generated by the libraries are pretty much comparable. GEQO has a bit worse average and median solution but it is more predictable than SAIO.

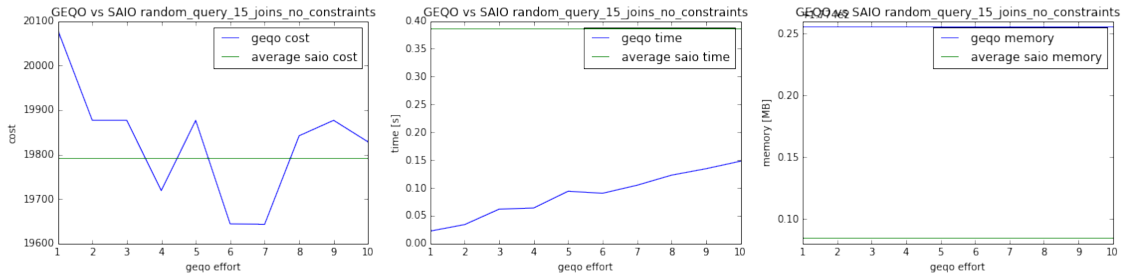
For queries with 15 LEFT JOINS SAIO finds worse solutions but needs less time and memory than GEQO.

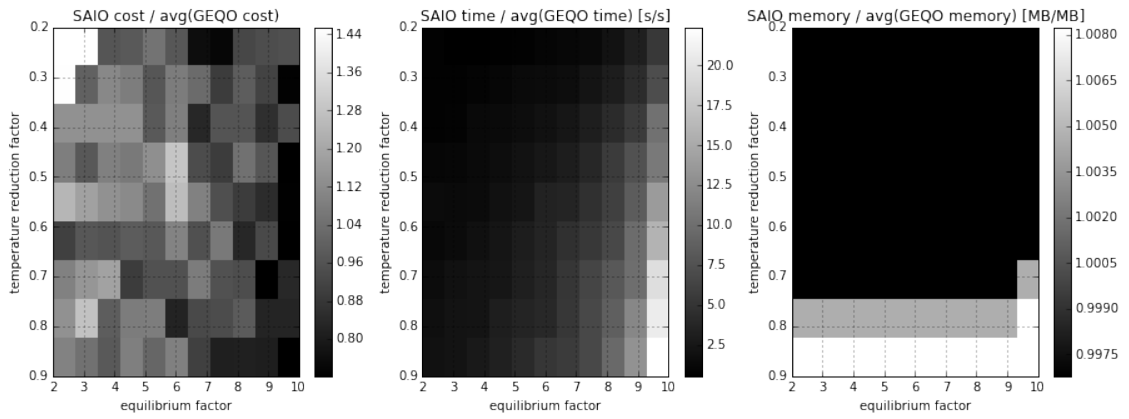
For queries with 15 RIGHT JOINS we can see that GEQO returns some strangely high estimations. It looks like bushy trees produced by SAIO are more suitable to model queries with many RIGHT JOINs.

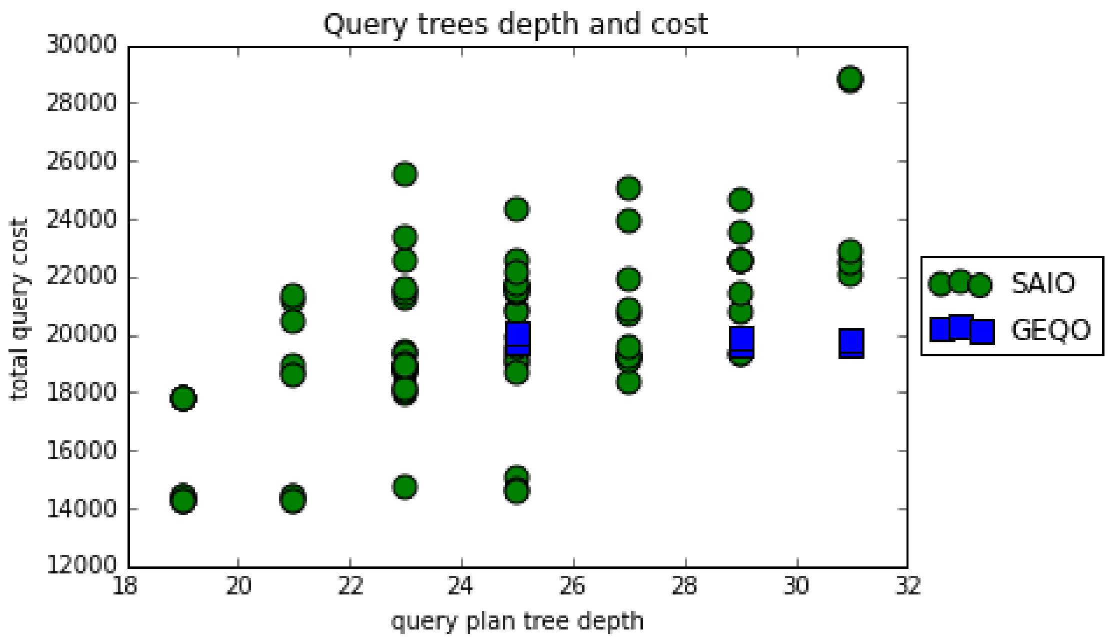
For a mixed query with 5 JOINS, 5 LEFT JOINS and 5 RIGHT JOINS SAIO wins again. This is probably due to presence of RIGHT JOINs in the query. GEQO doesn't seem to handle them well.

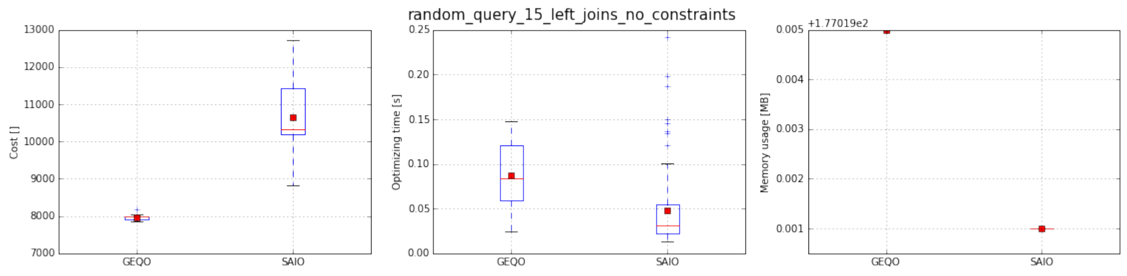
```
In [28]: display_info_for_test_case('random_query_15_joins_no_constraints')
         display_info_for_test_case('random_query_15_left_joins_no_constraints')
         display_info_for_test_case('random_query_15_right_joins_no_constraints')
         display_info_for_test_case('random_query_5_joins_5_left_5_right')
```

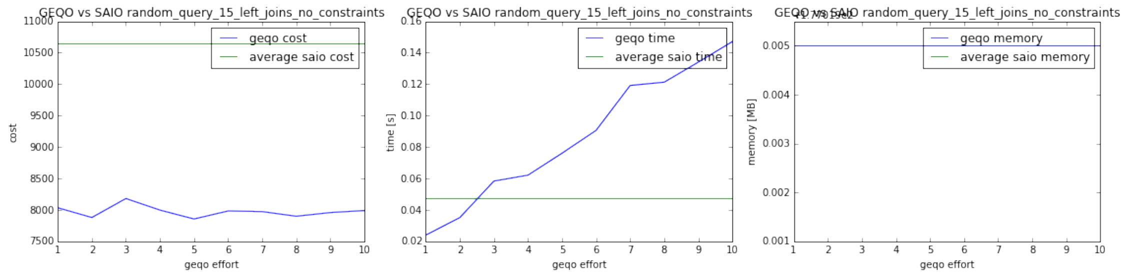


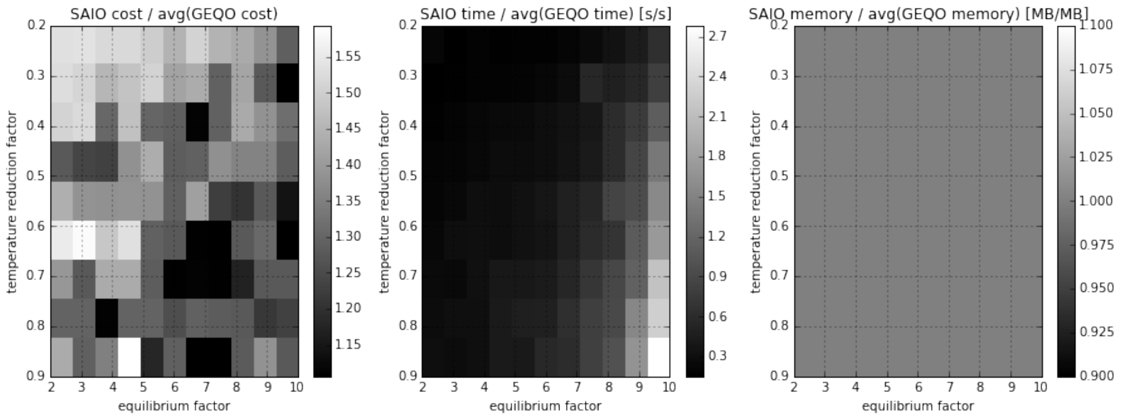


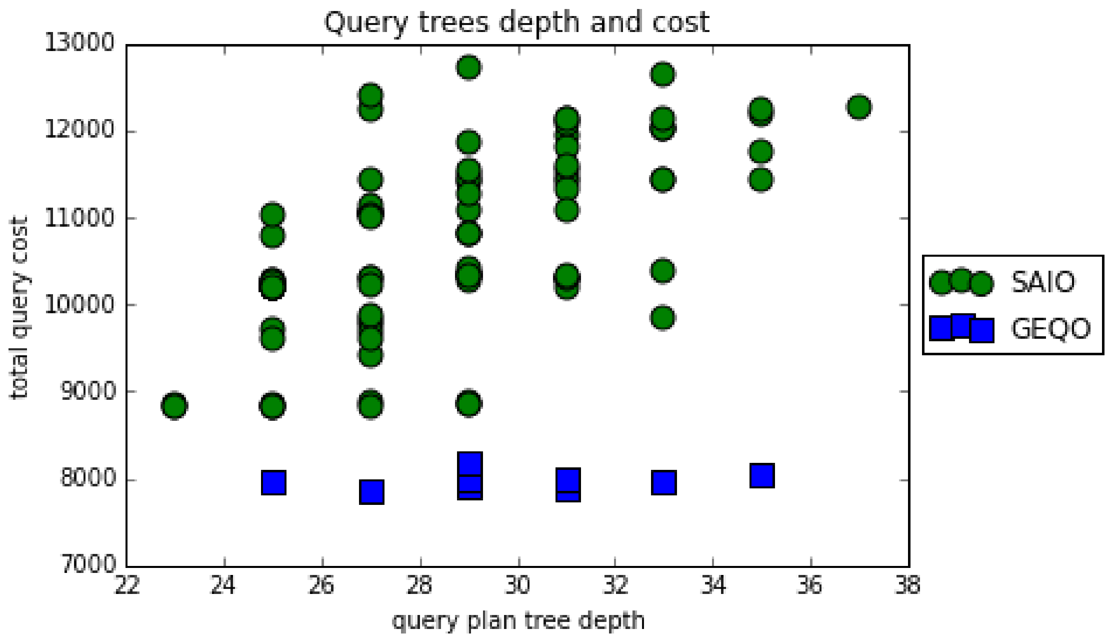


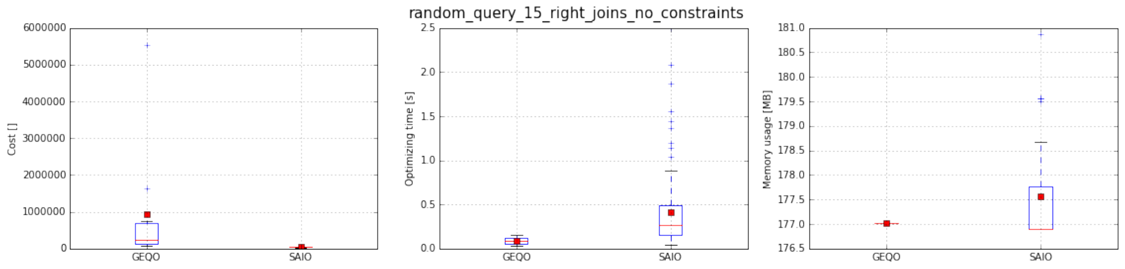

Query trees depth and cost
total query cost
query plan tree depth
SAIO
GEQO

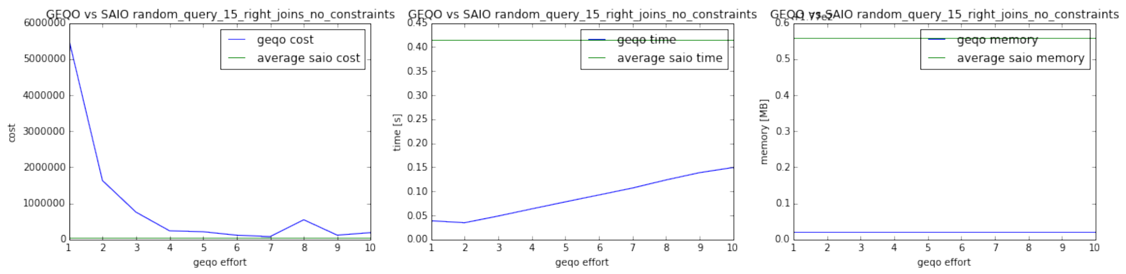

random_query_15_left_joins_no_constraints
Cost []
Optimizing time [s]
Memory usage [MB]
+1.77019e2
GEQO
SAIO

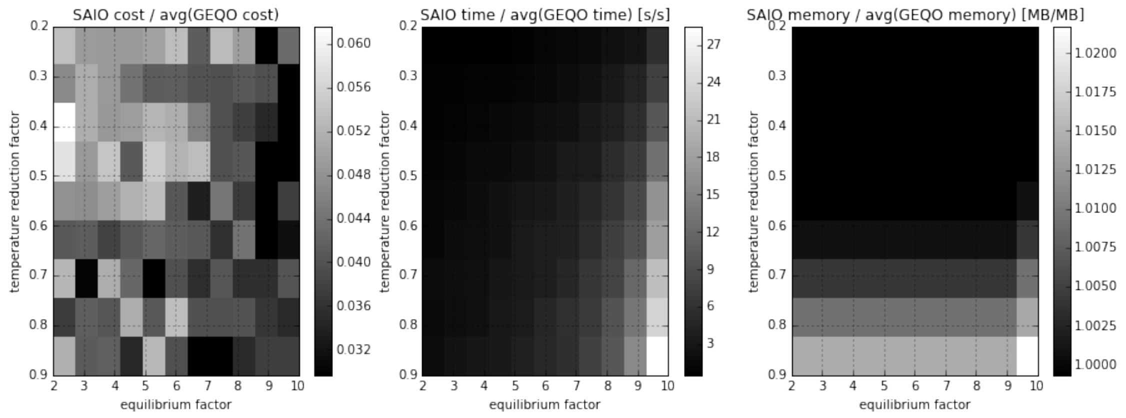

GEQO vs SAIO random_query_15_left_joins_no_constraints
geqo cost
average saio cost
geqo time
average saio time
geqo memory
average saio memory
cost
time [s]
memory [MB]
geqo effort

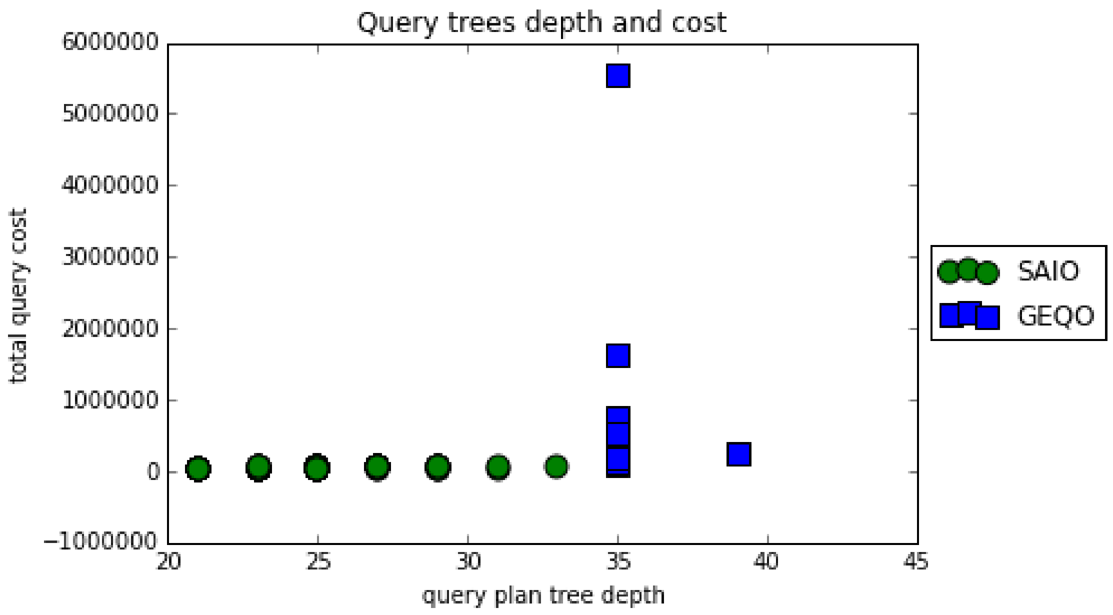

SAIO cost / avg(GEQO cost)
SAIO time / avg(GEQO time) [s/s]
SAIO memory / avg(GEQO memory) [MB/MB]
temperature reduction factor
equilibrium factor

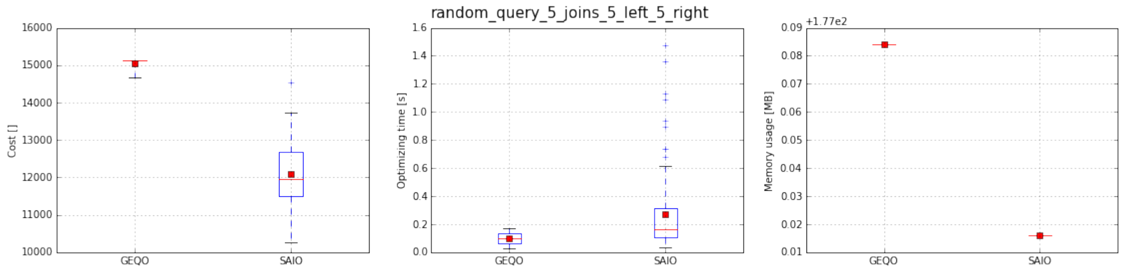

Query trees depth and cost
total query cost
query plan tree depth
SAIO
GEQO

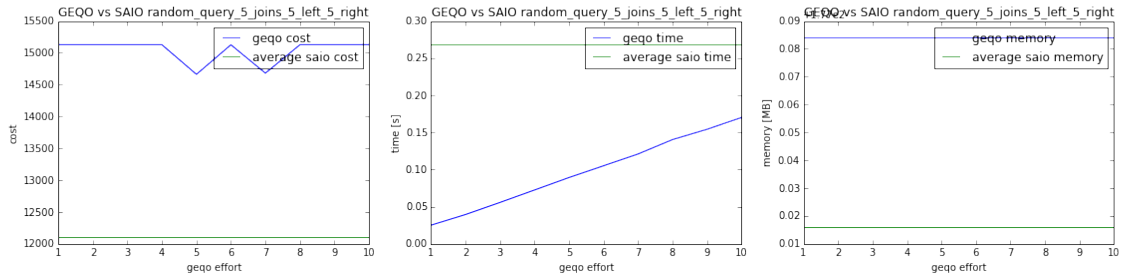

random_query_15_right_joins_no_constraints
Cost []
Optimizing time [s]
Memory usage [MB]
GEQO
SAIO

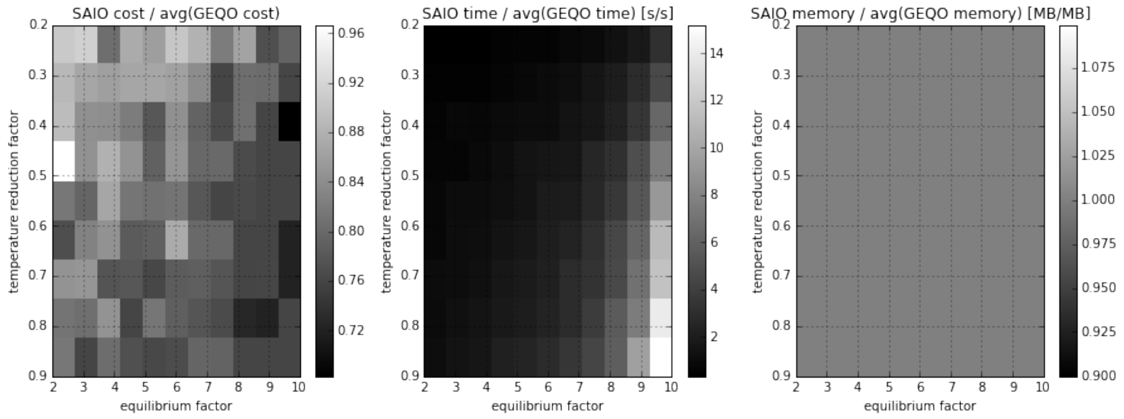

GEQO vs SAIO random_query_15_right_joins_no_constraints
geqo cost
average saio cost
cost
geqo effort
geqo time
average saio time
time [s]
geqo memory
average saio memory
memory [MB]

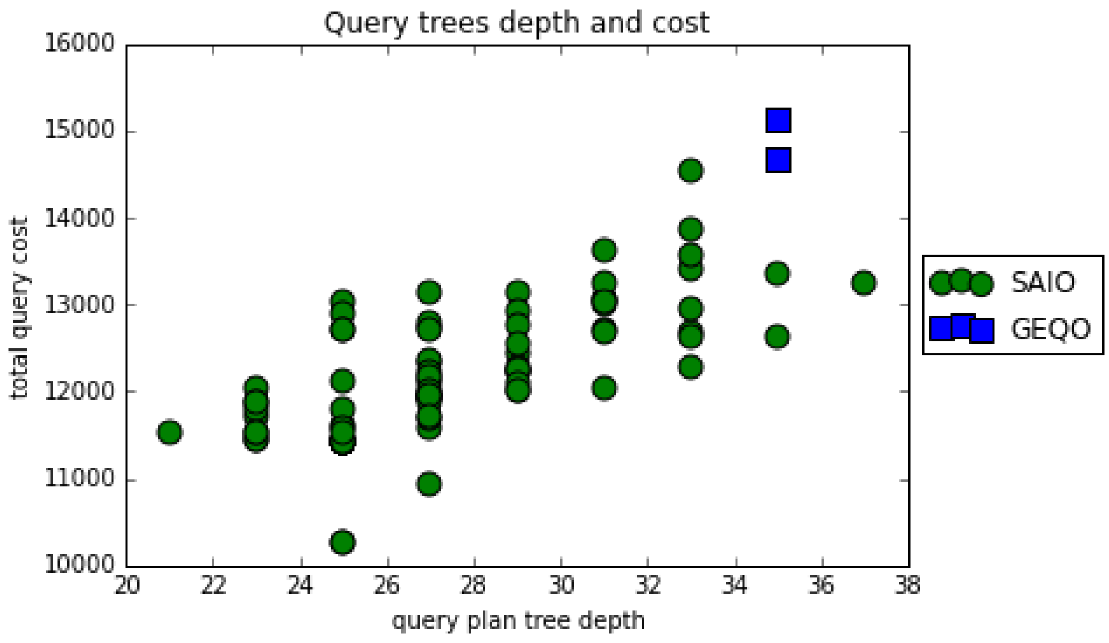

SAIO cost / avg(GEQO cost)
SAIO time / avg(GEQO time) [s/s]
SAIO memory / avg(GEQO memory) [MB/MB]
temperature reduction factor
equilibrium factor


Query trees depth and cost
total query cost
query plan tree depth
SAIO
GEQO


random_query_5_joins_5_left_5_right
Cost []
Optimizing time [s]
Memory usage [MB]
+1.77e2
GEQO
SAIO

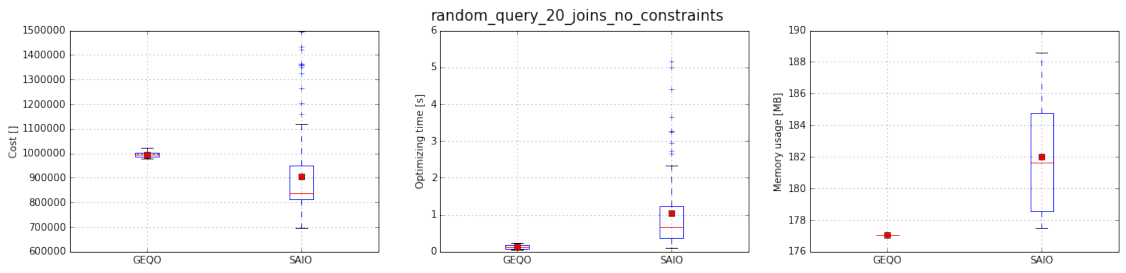

GEQO vs SAIO random_query_5_joins_5_left_5_right
geqo cost
average saio cost
geqo time
average saio time
geqo memory
average saio memory
cost
time [s]
memory [MB]
geqo effort

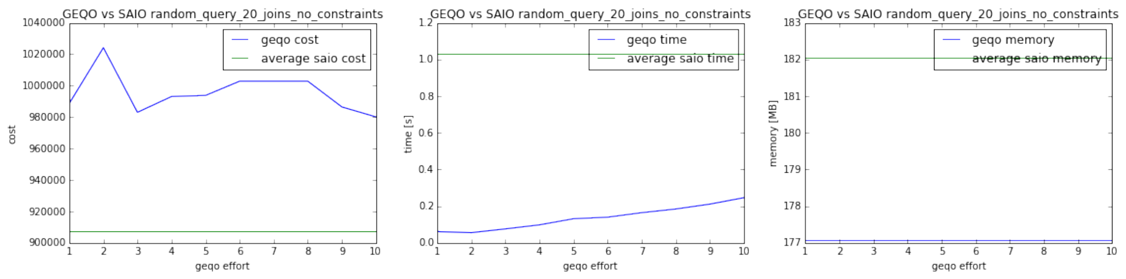

SAIO cost / avg(GEQO cost)
SAIO time / avg(GEQO time) [s/s]
SAIO memory / avg(GEQO memory) [MB/MB]
temperature reduction factor
equilibrium factor

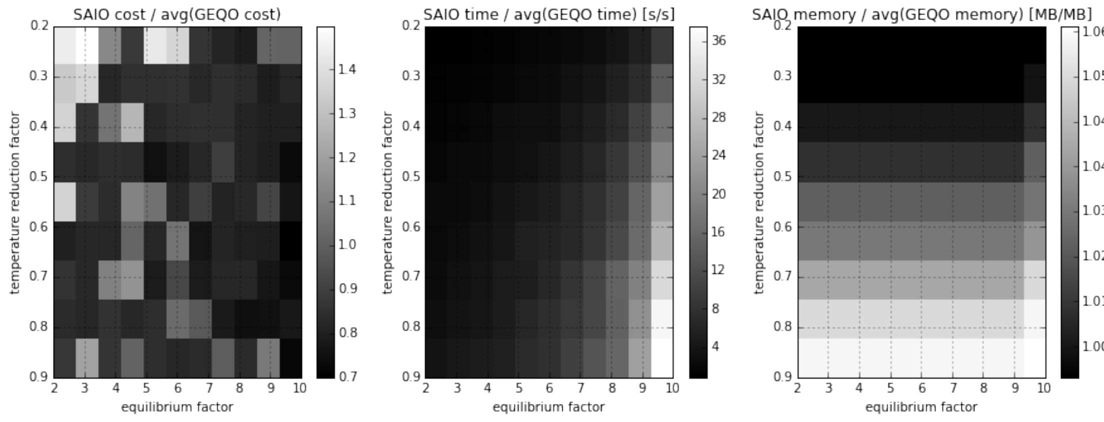


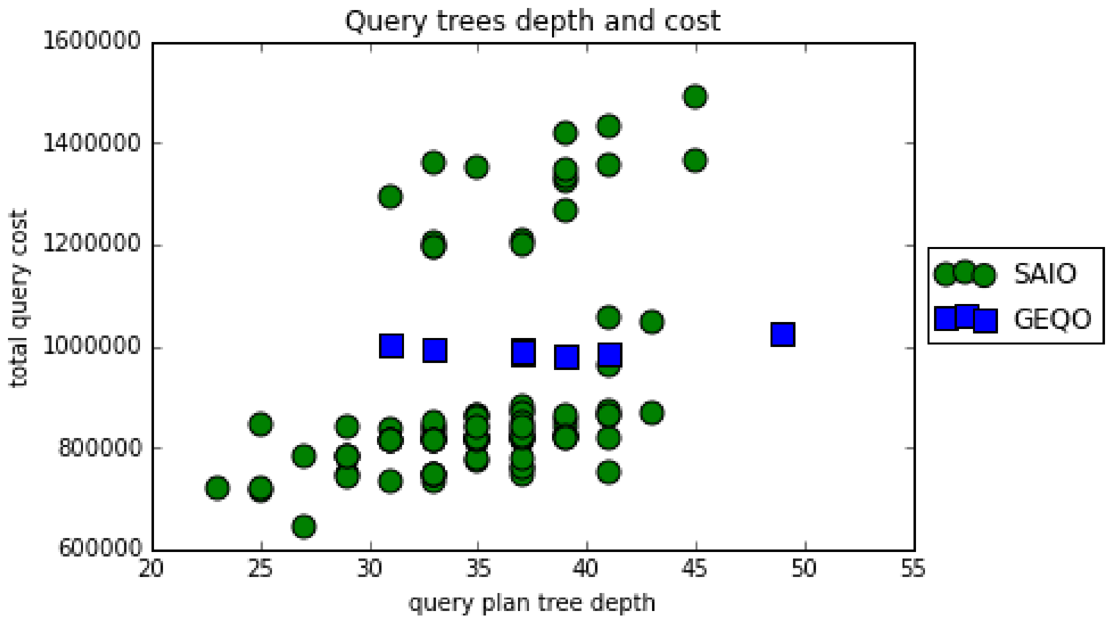
20 JOINS For these queries the observations from the previous smaller queries hold. SAIO seems comparable to GEQO for queries with regular JOINS, worse for queries with LEFT JOINS and much better (correct!) for queries with RIGHT JOINS and mixed queries.

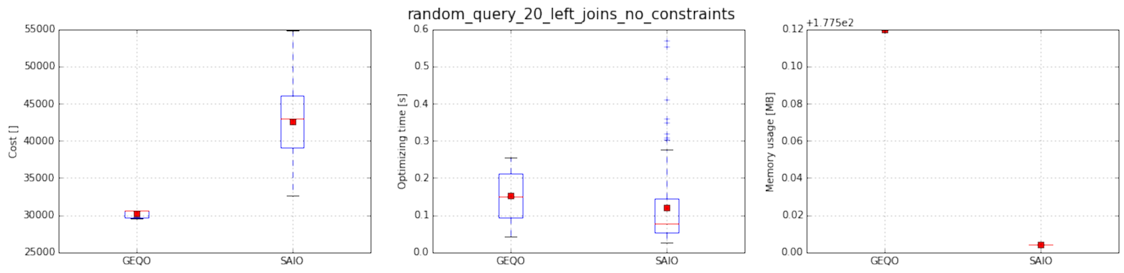
```
In [21]: display_info_for_test_case('random_query_20_joins_no_constraints')
         display_info_for_test_case('random_query_20_left_joins_no_constraints')
         display_info_for_test_case('random_query_20_right_joins_no_constraints')
         display_info_for_test_case('random_query_10_joins_5_left_5_right')
```

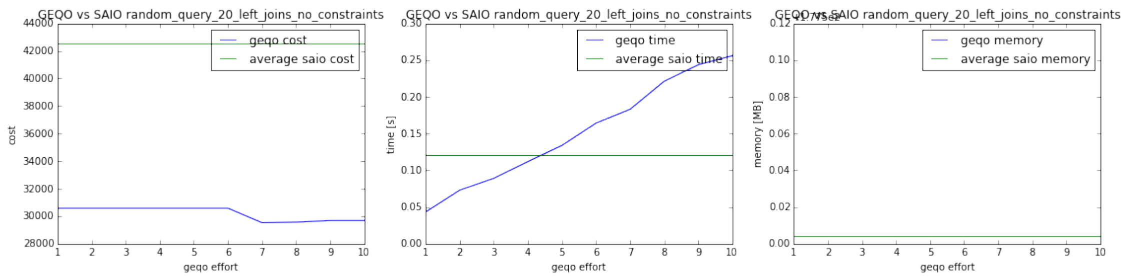


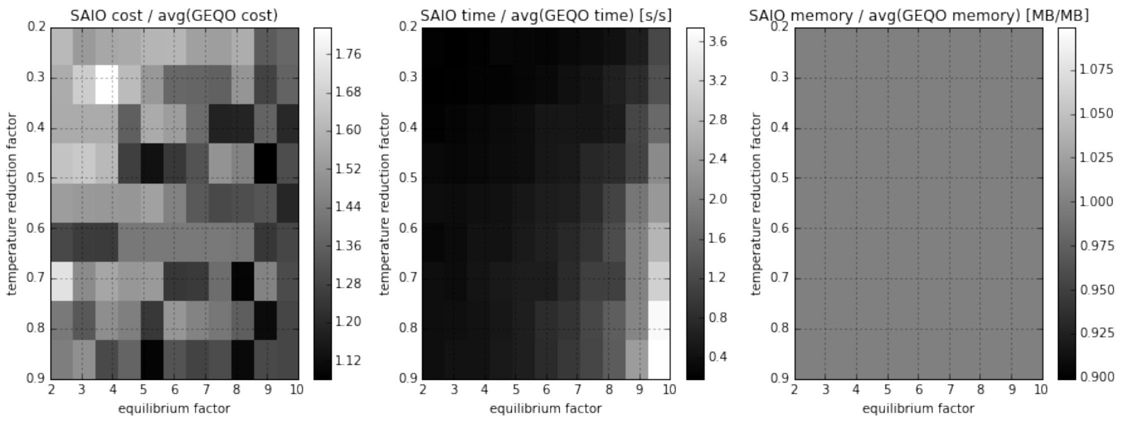


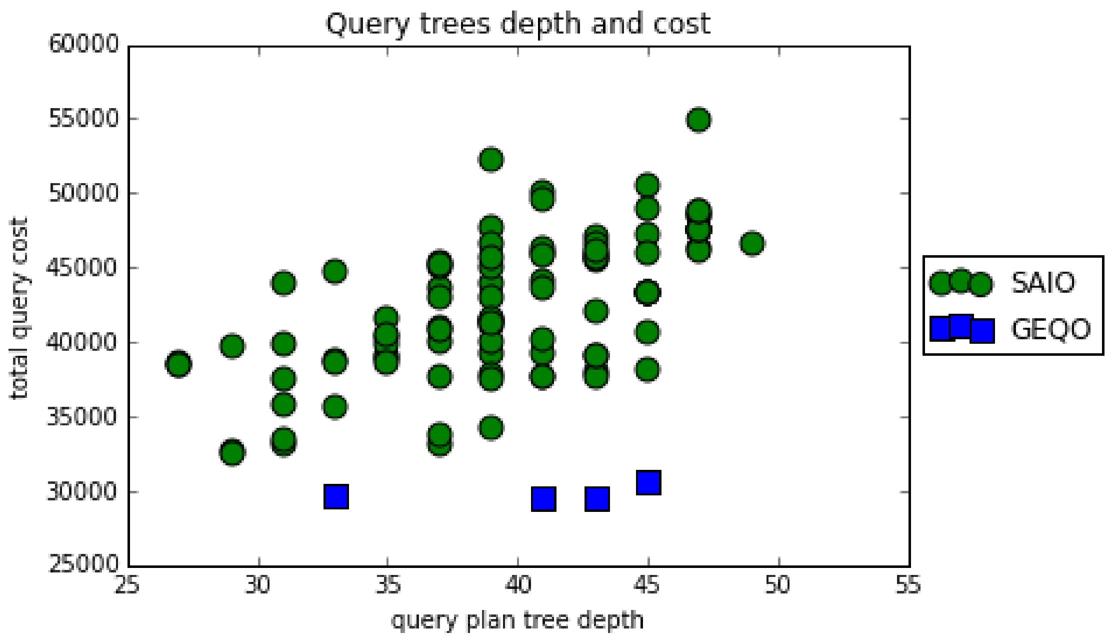

SAIO cost / avg(GEQO cost)
SAIO time / avg(GEQO time) [s/s]
SAIO memory / avg(GEQO memory) [MB/MB]
temperature reduction factor
equilibrium factor

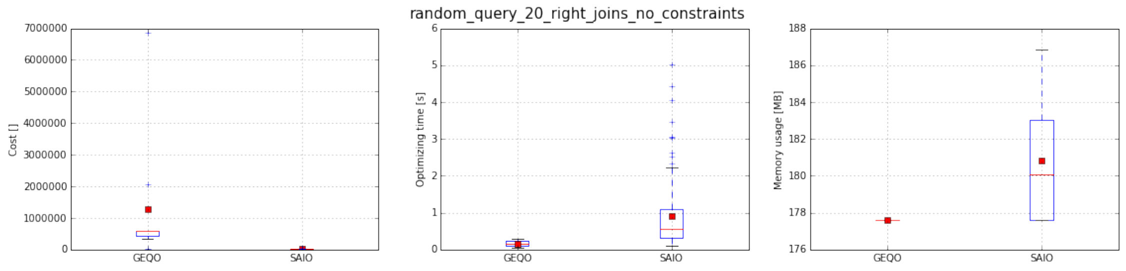

Query trees depth and cost
total query cost
query plan tree depth
SAIO
GEQO

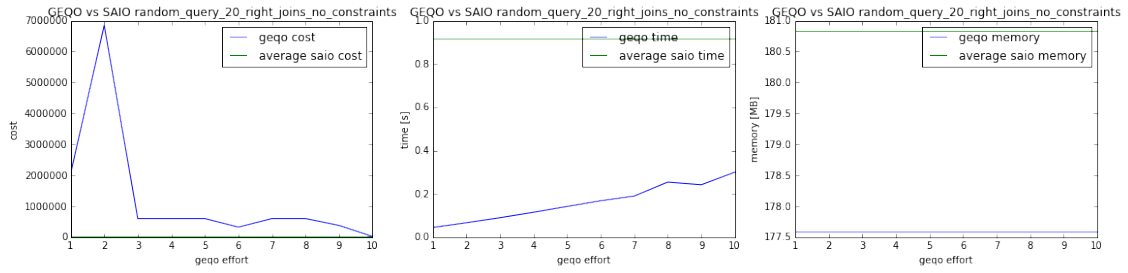

random_query_20_left_joins_no_constraints
Cost []
Optimizing time [s]
Memory usage [MB]
+1.775e2
GEQO
SAIO

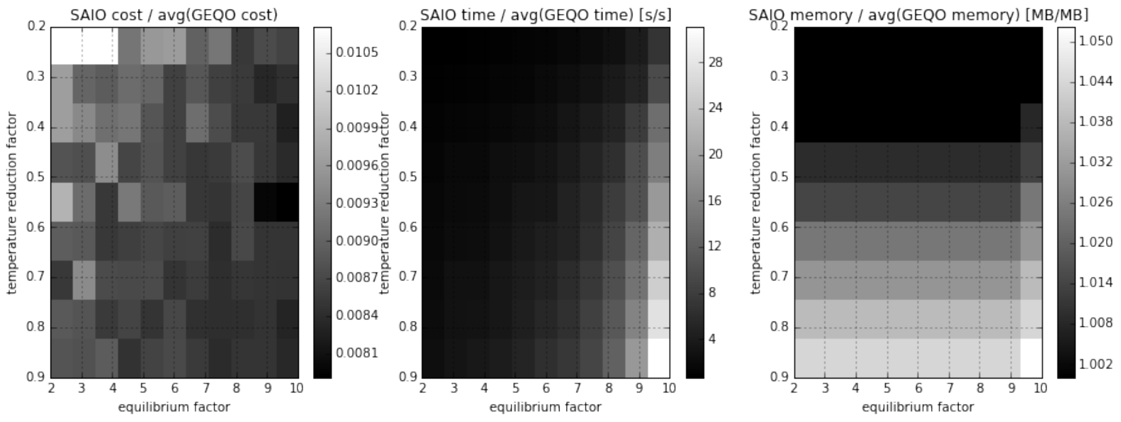

GEQO vs SAIO random_query_20_left_joins_no_constraints
geqo cost
average saio cost
cost
geqo effort
geqo time
average saio time
time [s]
geqo memory
average saio memory
memory [MB]

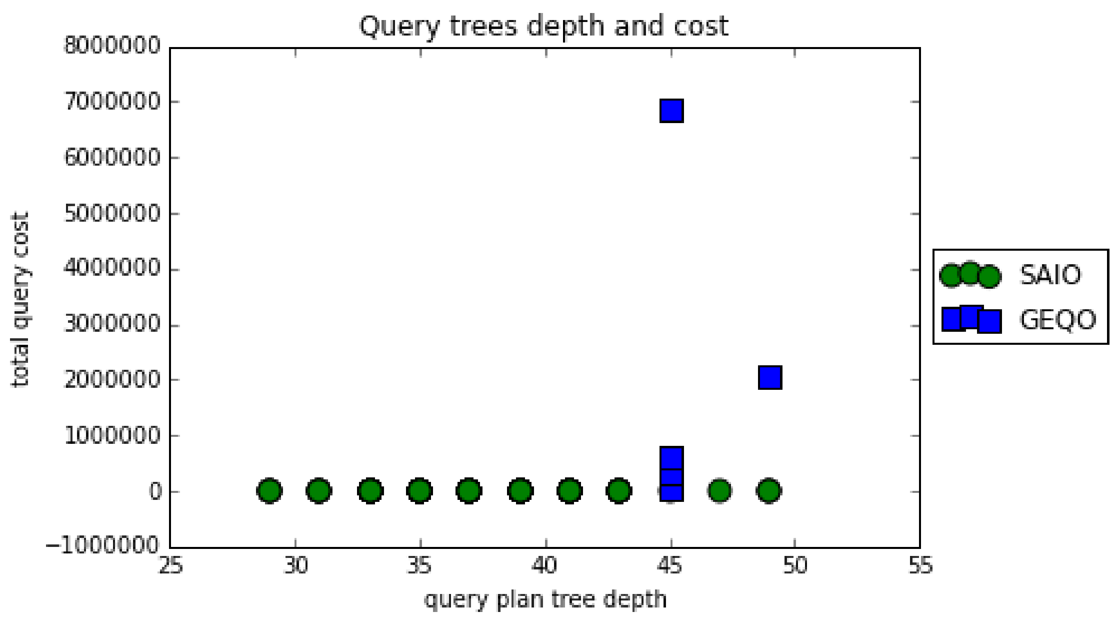

SAIO cost / avg(GEQO cost)
SAIO time / avg(GEQO time) [s/s]
SAIO memory / avg(GEQO memory) [MB/MB]
temperature reduction factor
equilibrium factor

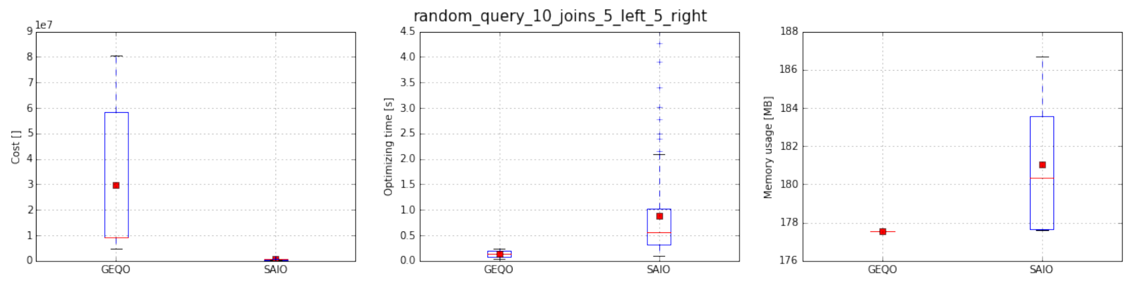

Query trees depth and cost
total query cost
query plan tree depth
SAIO
GEQO

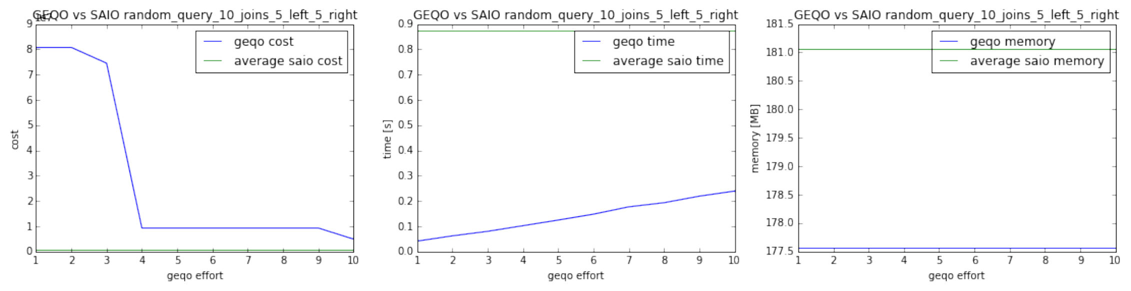

random_query_20_right_joins_no_constraints
Cost []
Optimizing time [s]
Memory usage [MB]
GEQO
SAIO

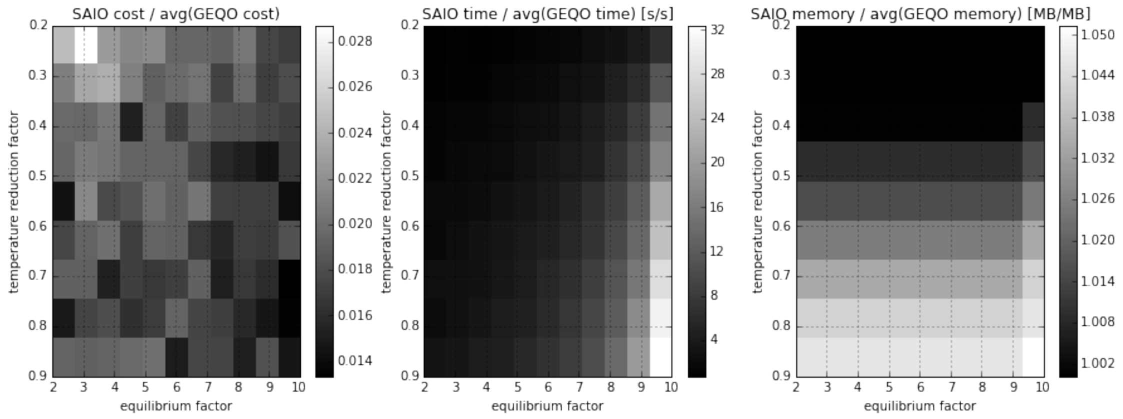

GEQO vs SAIO random_query_20_right_joins_no_constraints
geqo cost
average saio cost
geqo time
average saio time
geqo memory
average saio memory
cost
time [s]
memory [MB]
geqo effort

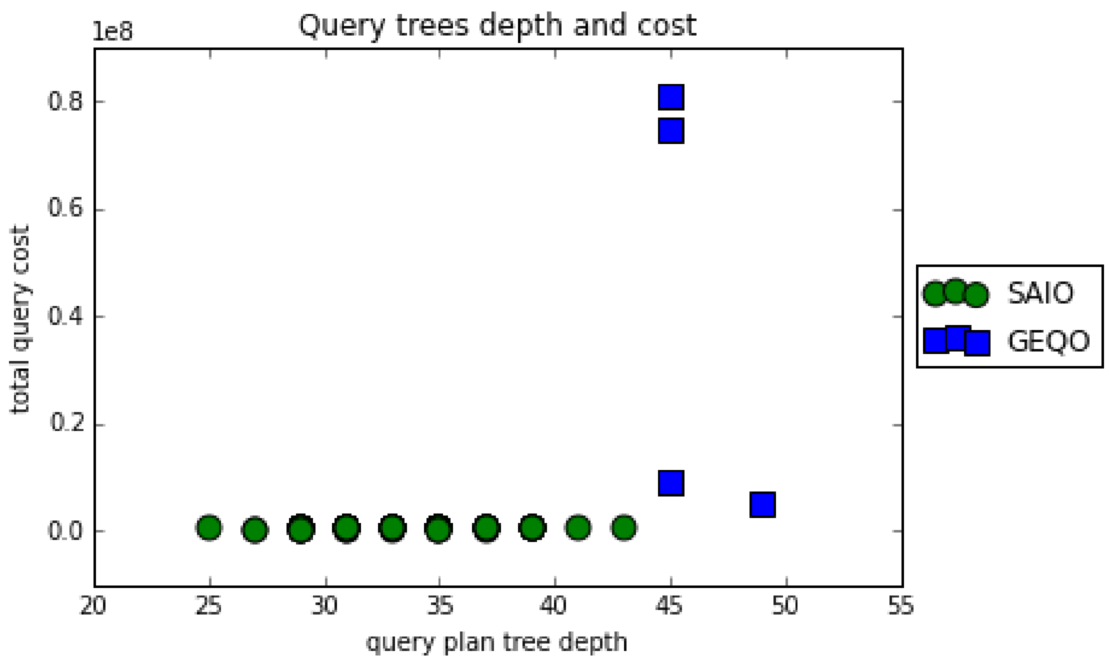

SAIO cost / avg(GEQO cost)
SAIO time / avg(GEQO time) [s/s]
SAIO memory / avg(GEQO memory) [MB/MB]
temperature reduction factor
equilibrium factor


Query trees depth and cost
total query cost
query plan tree depth
SAIO
GEQO


random_query_10_joins_5_left_5_right
Cost []
Optimizing time [s]
Memory usage [MB]
GEQO
SAIO

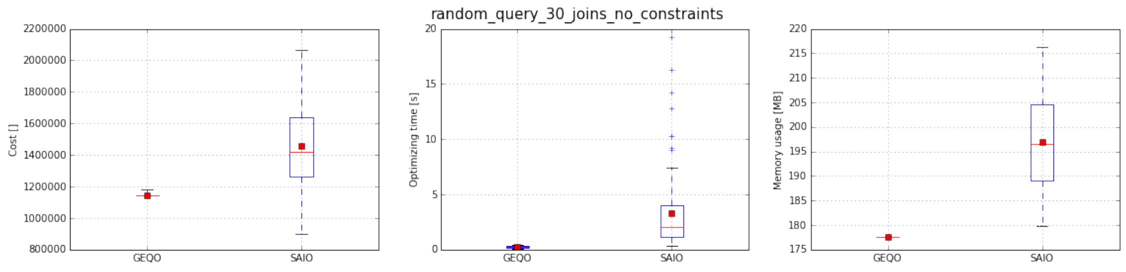

GEQO vs SAIO random_query_10_joins_5_left_5_right
geqo cost
average saio cost
geqo time
average saio time
geqo memory
average saio memory
cost
time [s]
memory [MB]
geqo effort

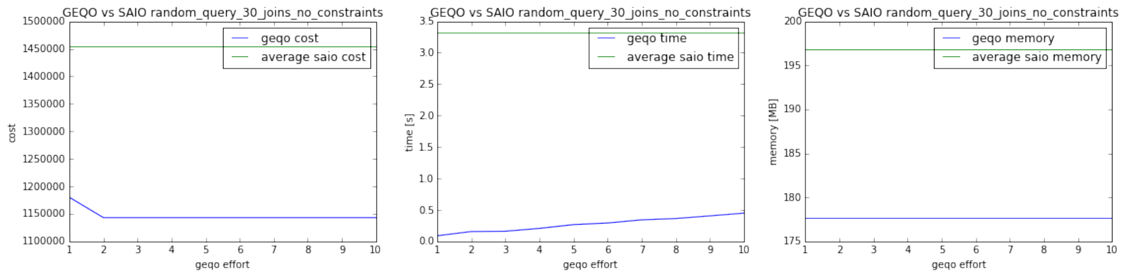


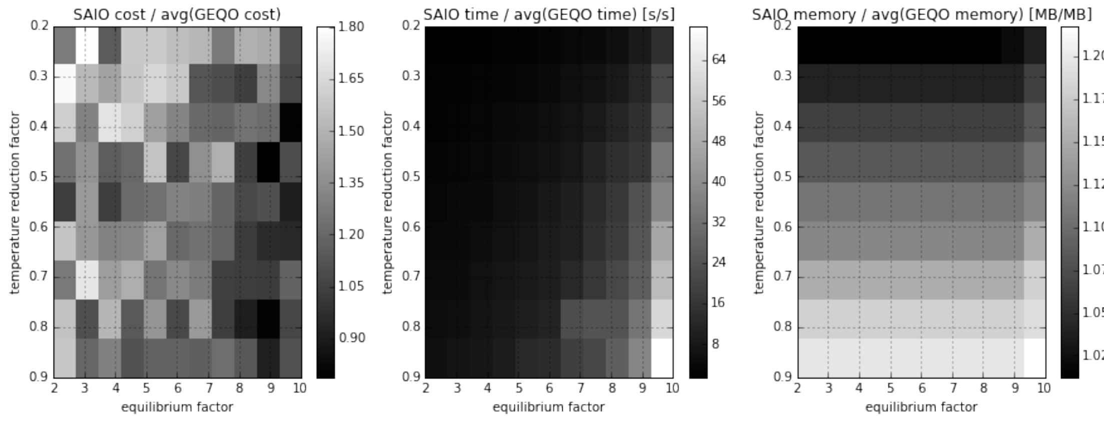


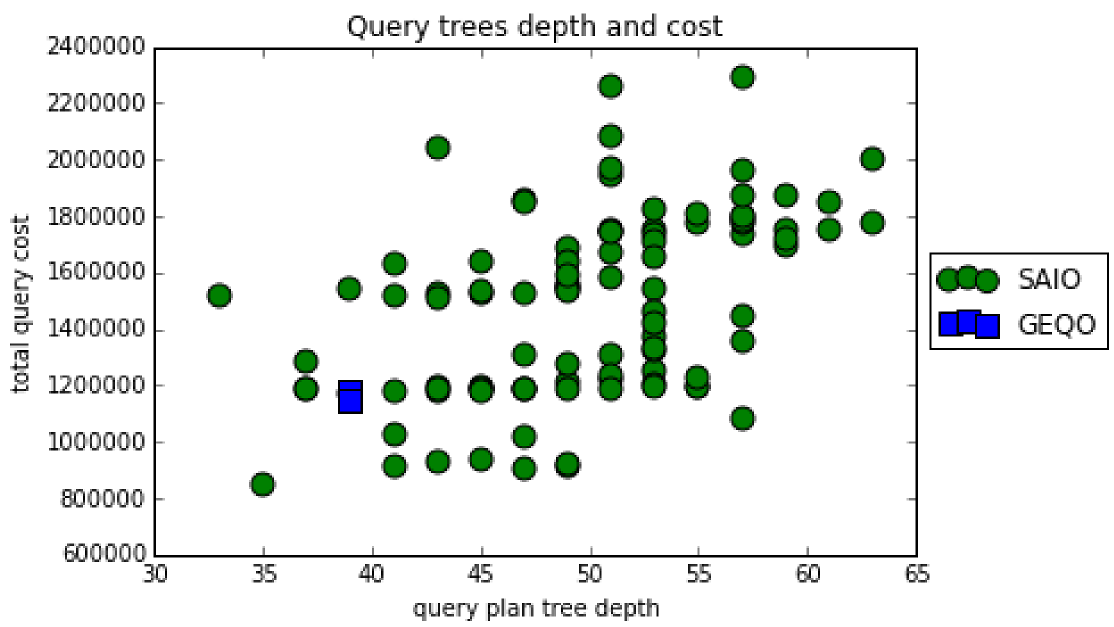
30 JOINS The observations from previous cases of flat queries hold.

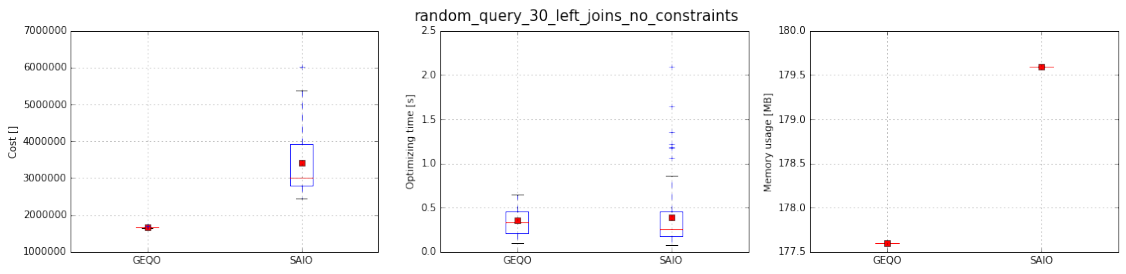
```
In [22]: display_info_for_test_case('random_query_30_joins_no_constraints')
         display_info_for_test_case('random_query_30_left_joins_no_constraints')
         display_info_for_test_case('random_query_30_right_joins_no_constraints')
         display_info_for_test_case('random_query_10_joins_10_left_10_right')
```

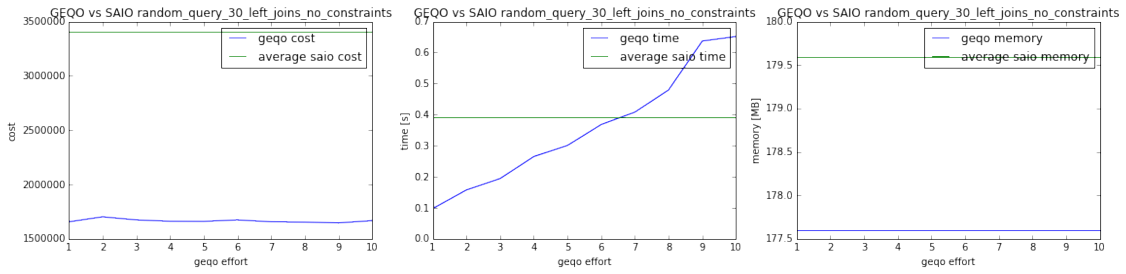

random_query_30_joins_no_constraints
Cost []
Optimizing time [s]
Memory usage [MB]
GEQO
SAIO

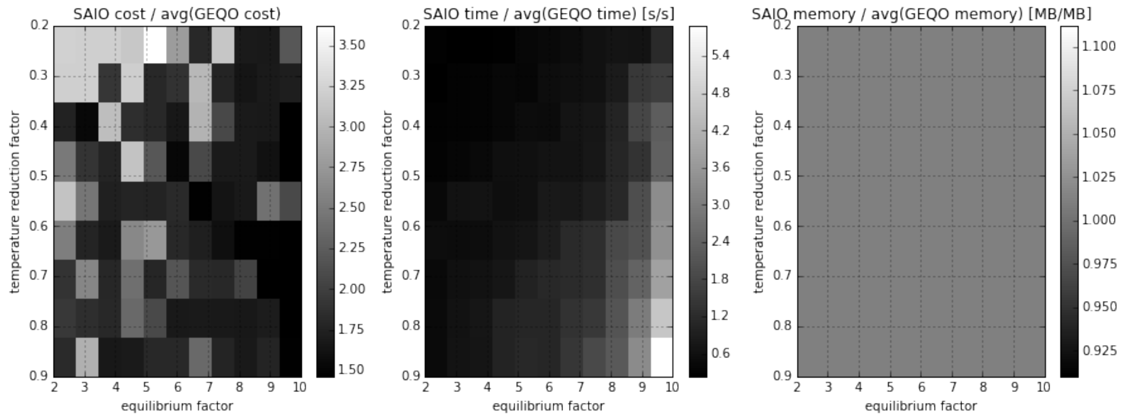

GEQO vs SAIO random_query_30_joins_no_constraints
geqo cost
average saio cost
cost
geqo effort
geqo time
average saio time
time [s]
geqo memory
average saio memory
memory [MB]

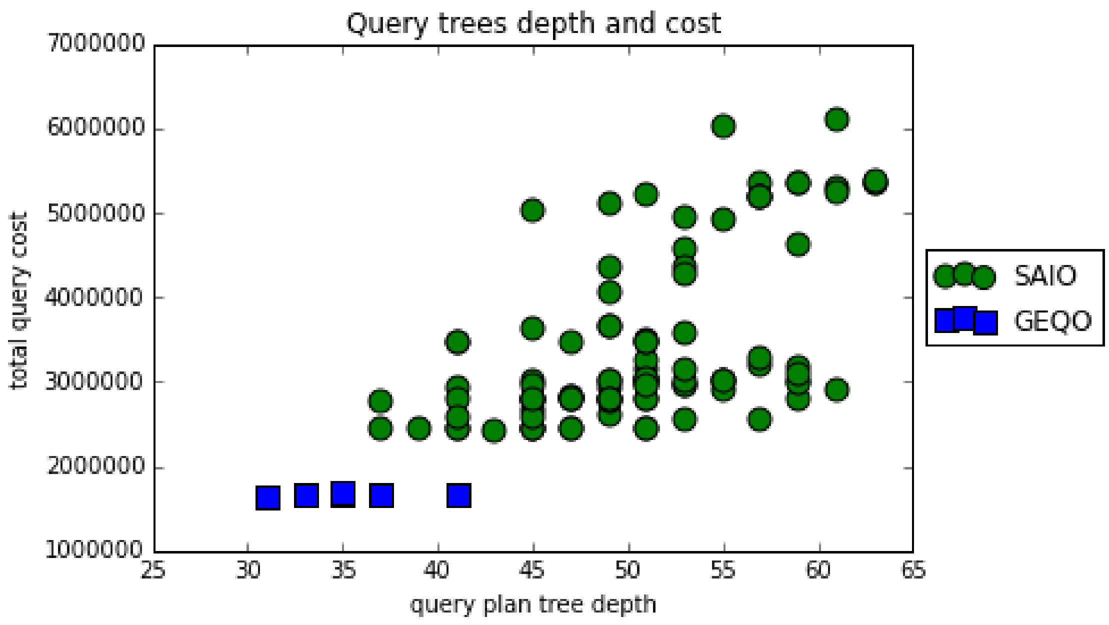

SAIO cost / avg(GEQO cost)
SAIO time / avg(GEQO time) [s/s]
SAIO memory / avg(GEQO memory) [MB/MB]
temperature reduction factor
equilibrium factor

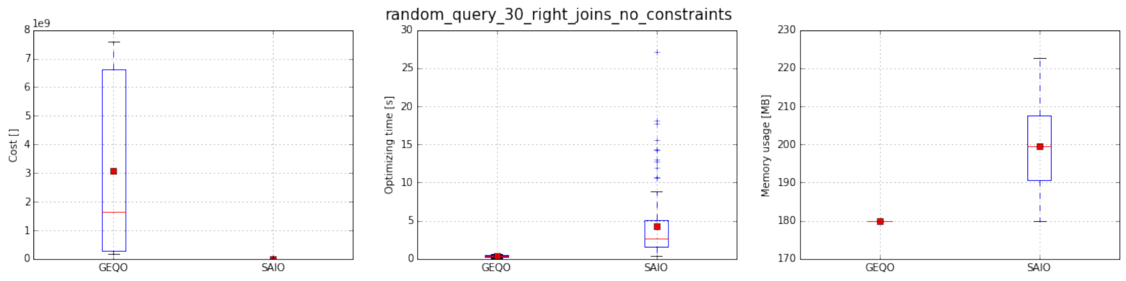

Query trees depth and cost
total query cost
query plan tree depth
SAIO
GEQO

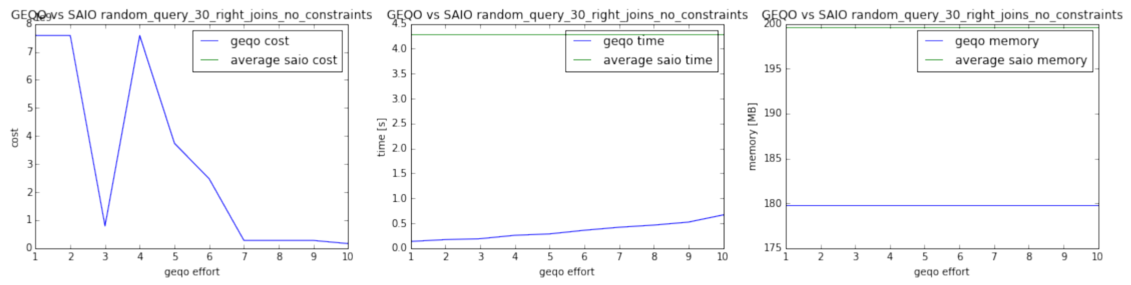

random_query_30_left_joins_no_constraints
Cost []
Optimizing time [s]
Memory usage [MB]
GEQO
SAIO

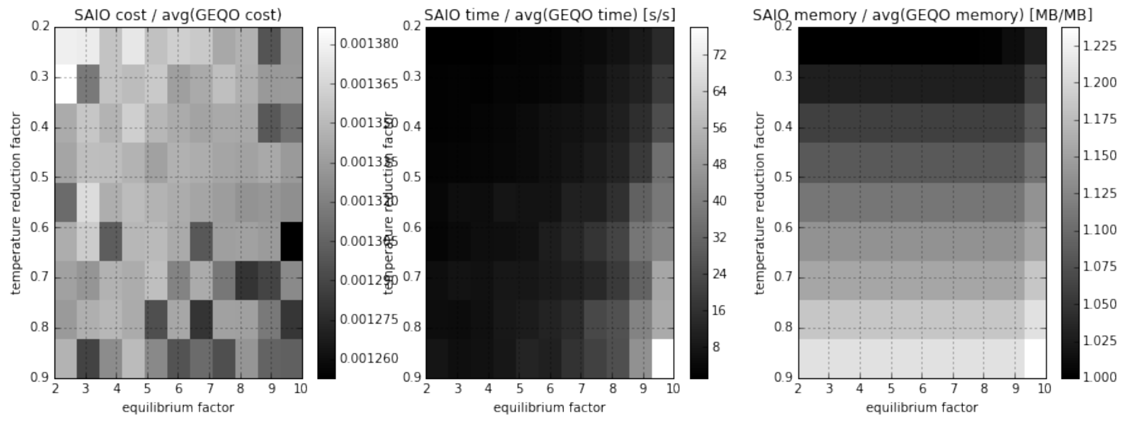

GEQO vs SAIO random_query_30_left_joins_no_constraints
geqo cost
average saio cost
geqo time
average saio time
geqo memory
average saio memory
cost
time [s]
memory [MB]
geqo effort

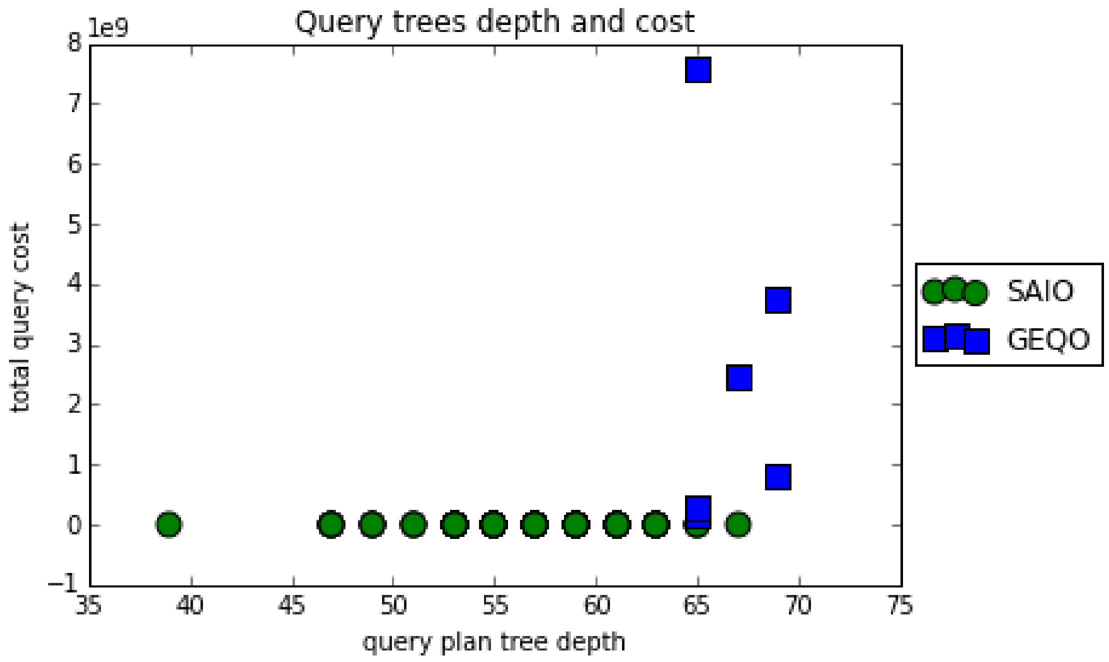

SAIO cost / avg(GEQO cost)
SAIO time / avg(GEQO time) [s/s]
SAIO memory / avg(GEQO memory) [MB/MB]
temperature reduction factor
equilibrium factor

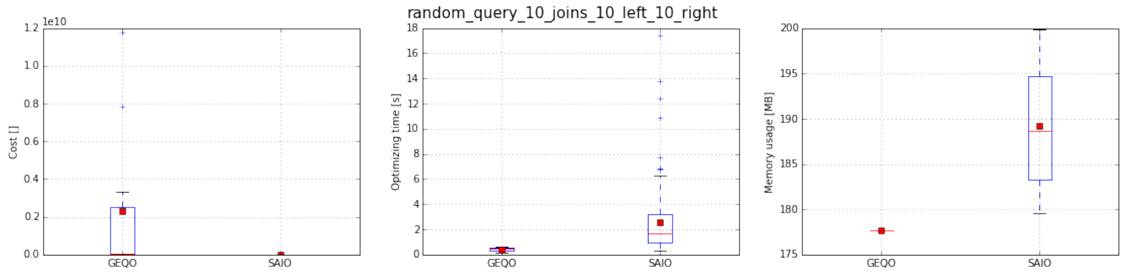

Query trees depth and cost
total query cost
query plan tree depth
SAIO
GEQO

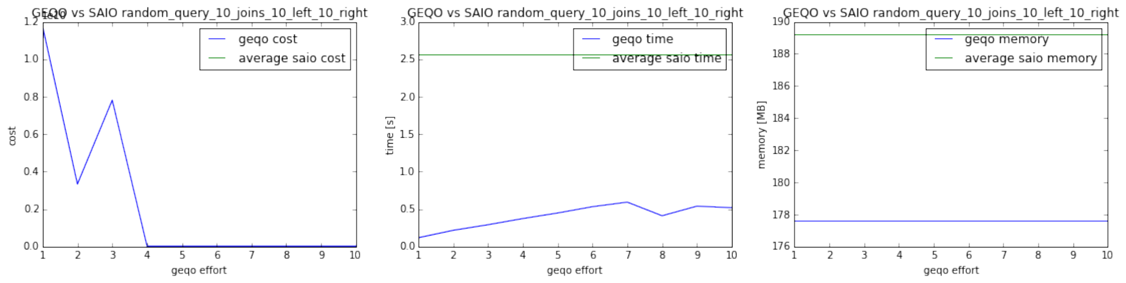

random_query_30_right_joins_no_constraints
Cost []
Optimizing time [s]
Memory usage [MB]
GEQO
SAIO
1e9

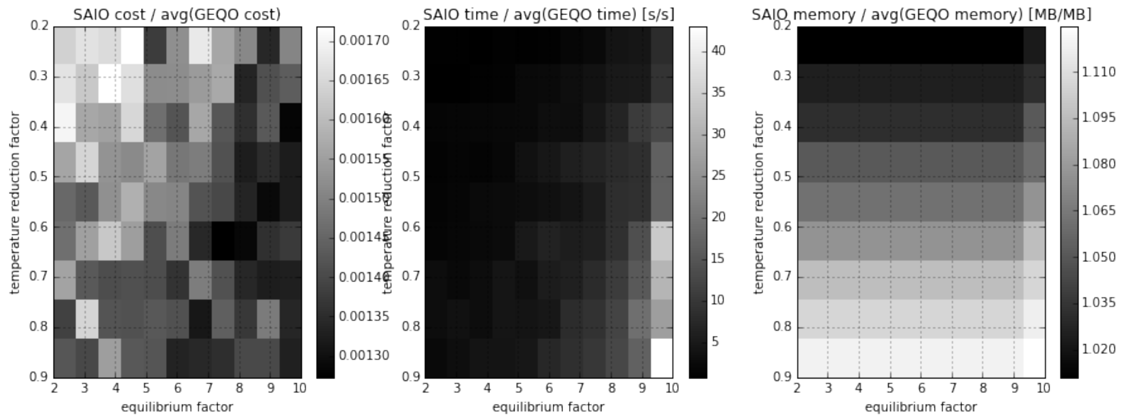

GEQO vs SAIO random_query_30_right_joins_no_constraints
geqo cost
average saio cost
cost
1e9
geqo effort
geqo time
average saio time
time [s]
geqo memory
average saio memory
memory [MB]

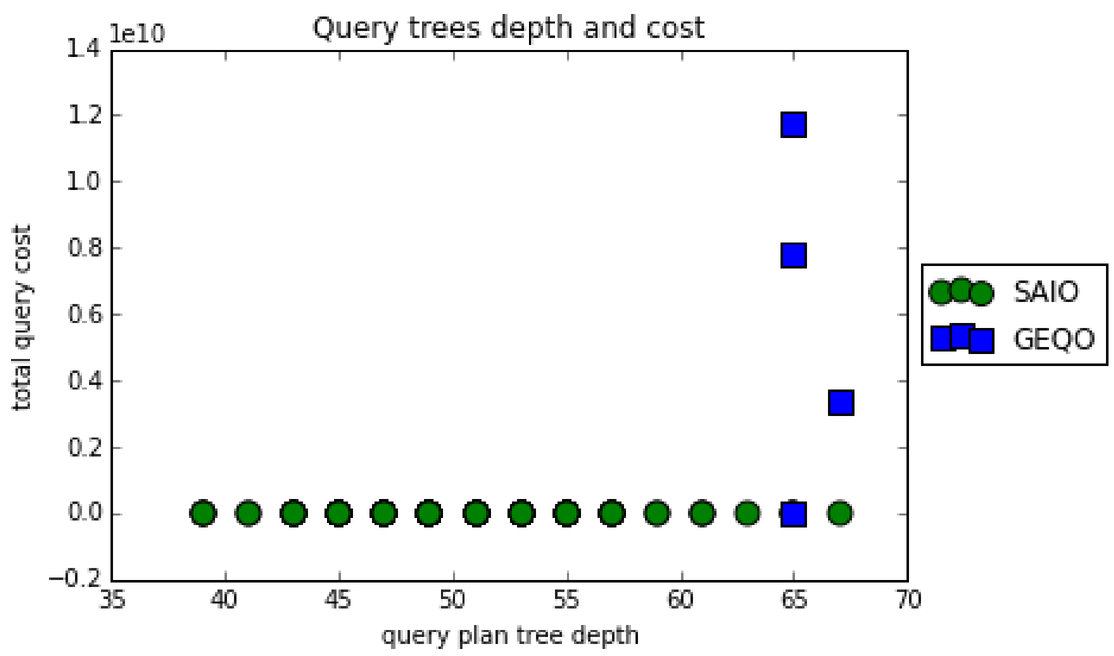

SAIO cost / avg(GEQO cost)
SAIO time / avg(GEQO time) [s/s]
SAIO memory / avg(GEQO memory) [MB/MB]
temperature reduction factor
equilibrium factor


Query trees depth and cost
1e9
total query cost
query plan tree depth
SAIO
GEQO


random_query_10_joins_10_left_10_right
1e10
Cost []
Optimizing time [s]
Memory usage [MB]
GEQO
SAIO


GEQO vs SAIO random_query_10_joins_10_left_10_right
1e10
cost
geqo cost
average saio cost
time [s]
geqo time
average saio time
memory [MB]
geqo memory
average saio memory
geqo effort


SAIO cost / avg(GEQO cost)
SAIO time / avg(GEQO time) [s/s]
SAIO memory / avg(GEQO memory) [MB/MB]
temperature reduction factor
equilibrium factor

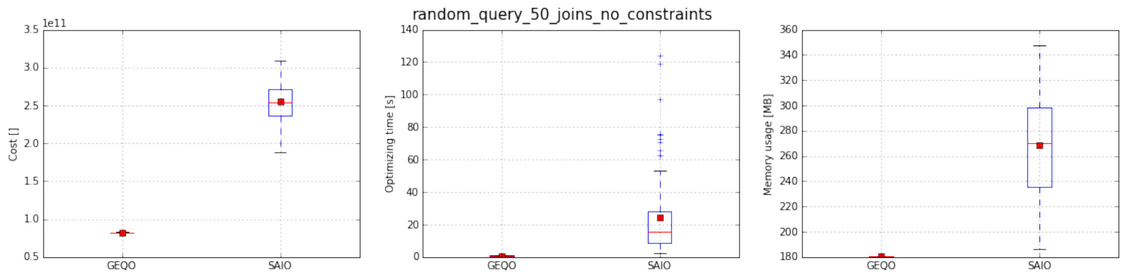


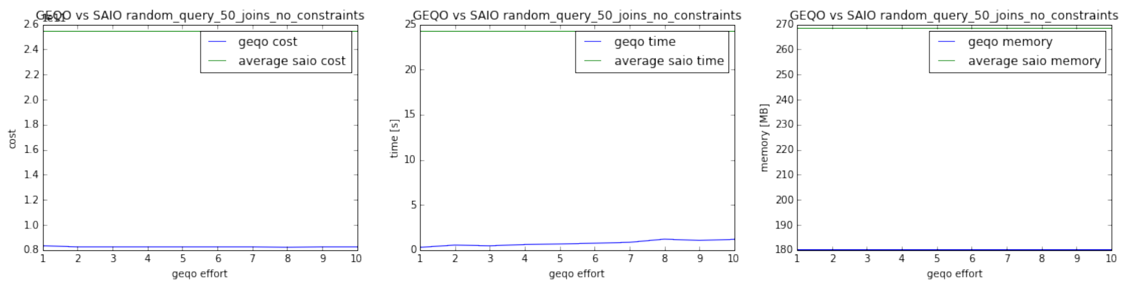
50 JOINS For the next part of flat queries, this time with 50 JOINS, we can see that GEQO produces better results for queries with JOINS and LEFT JOINS.

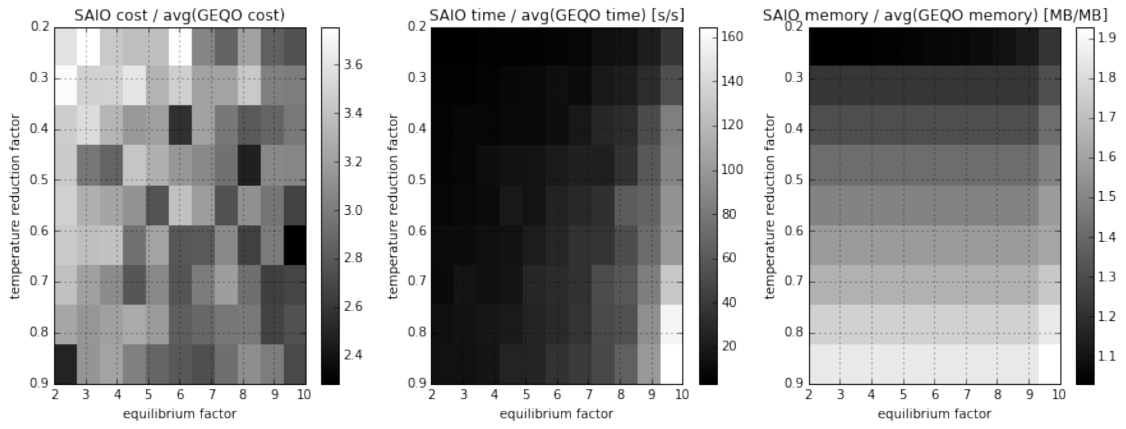
SAIO wins at RIGHT JOIN cases and mixed queries.

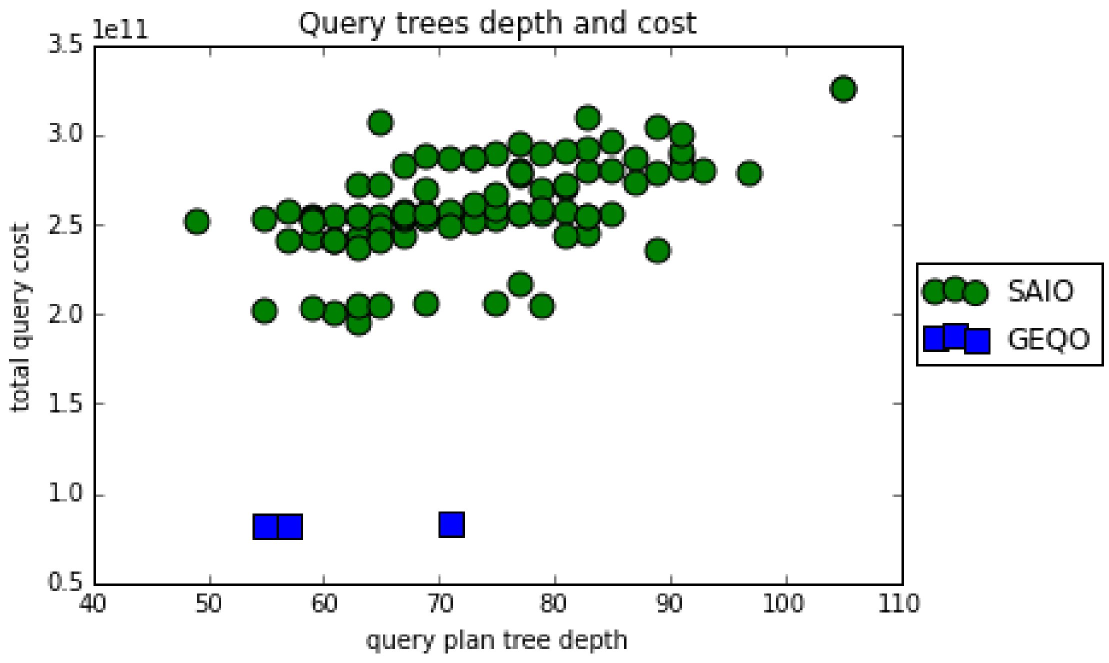
This is interesting that for the query with 50 LEFT JOINS GEQO actually produces a shorter tree than SAIO solutions.

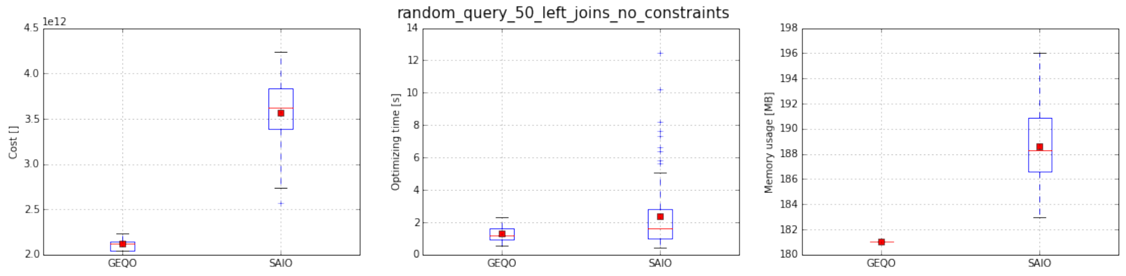
```
In [23]: display_info_for_test_case('random_query_50_joins_no_constraints')
         display_info_for_test_case('random_query_50_left_joins_no_constraints')
         display_info_for_test_case('random_query_50_right_joins_no_constraints')
         display_info_for_test_case('random_query_20_joins_15_left_15_right')
```

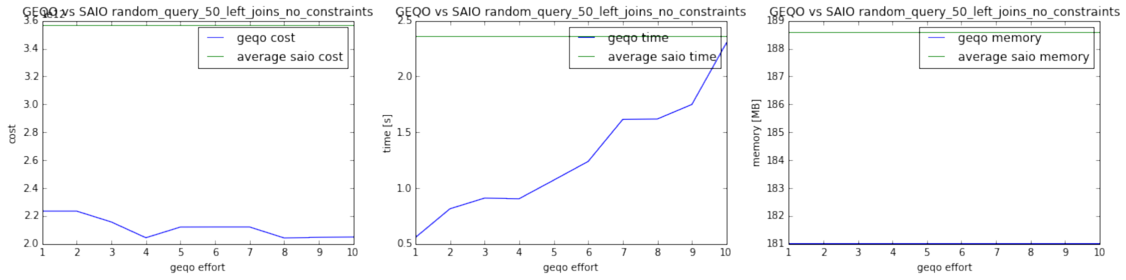


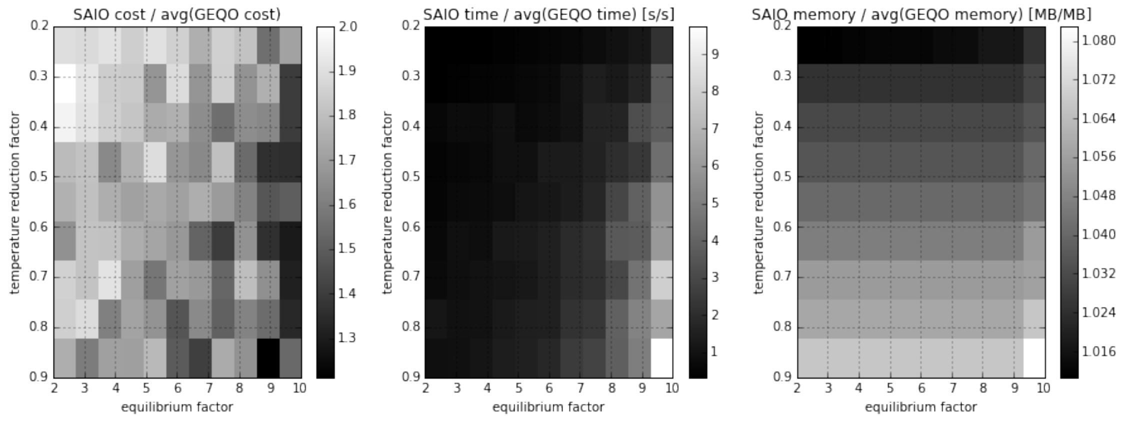

GEQO vs SAIO random_query_50_joins_no_constraints
1e11
cost
geqo effort
geqo cost
average saio cost
time [s]
geqo time
average saio time
memory [MB]
geqo memory
average saio memory

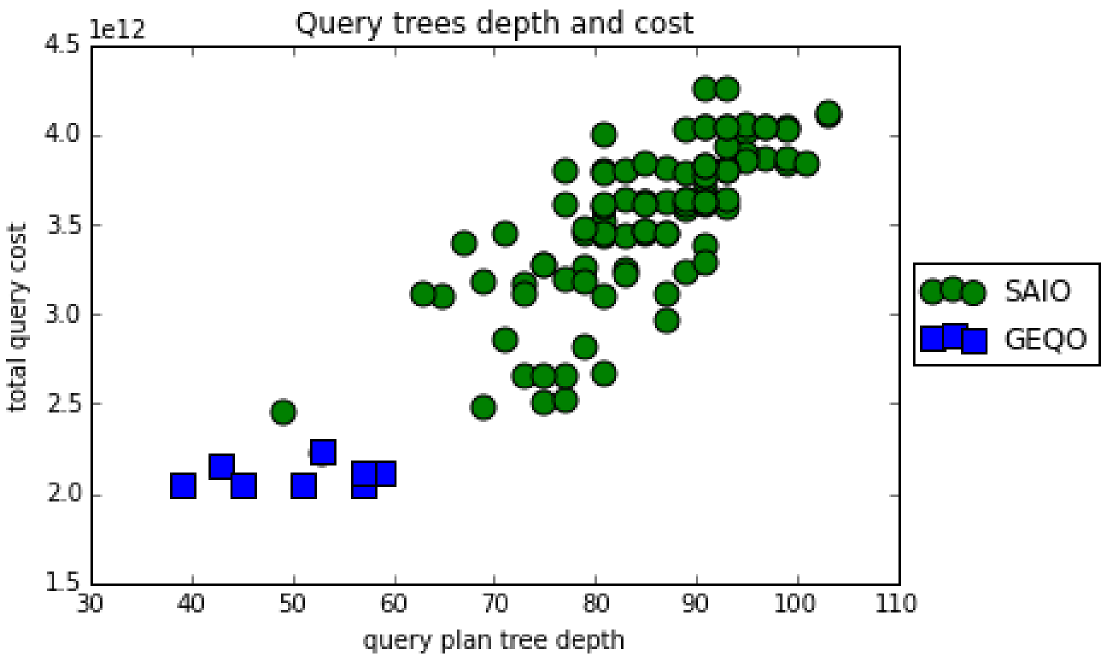

SAIO cost / avg(GEQO cost)
SAIO time / avg(GEQO time) [s/s]
SAIO memory / avg(GEQO memory) [MB/MB]
temperature reduction factor
equilibrium factor

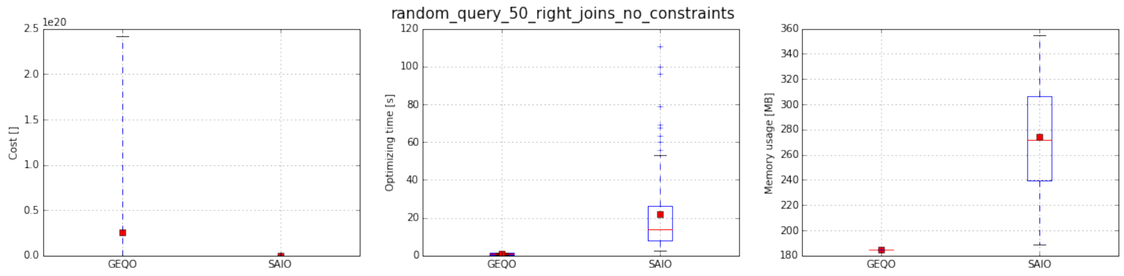

Query trees depth and cost
1e11
total query cost
query plan tree depth
SAIO
GEQO

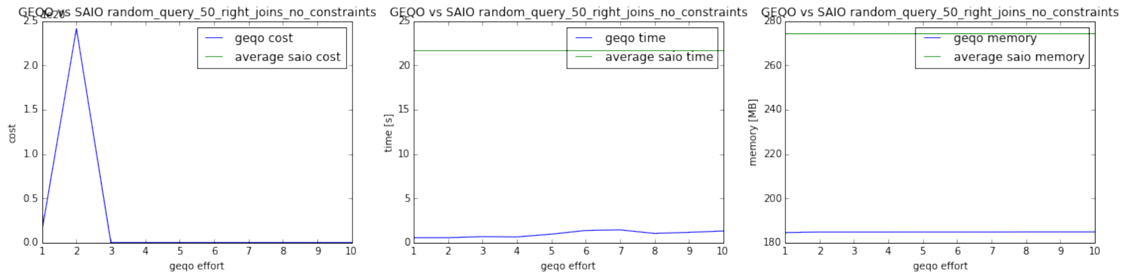

random_query_50_left_joins_no_constraints
Cost []
1e12
Optimizing time [s]
Memory usage [MB]
GEQO
SAIO

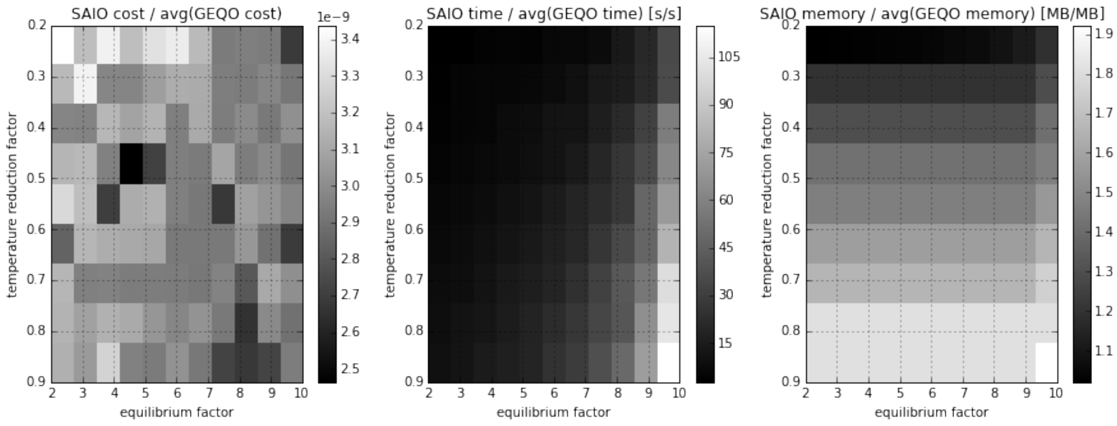

GEQO vs SAIO random_query_50_left_joins_no_constraints
geqo cost
average saio cost
cost
1e12
geqo effort
geqo time
average saio time
time [s]
geqo memory
average saio memory
memory [MB]

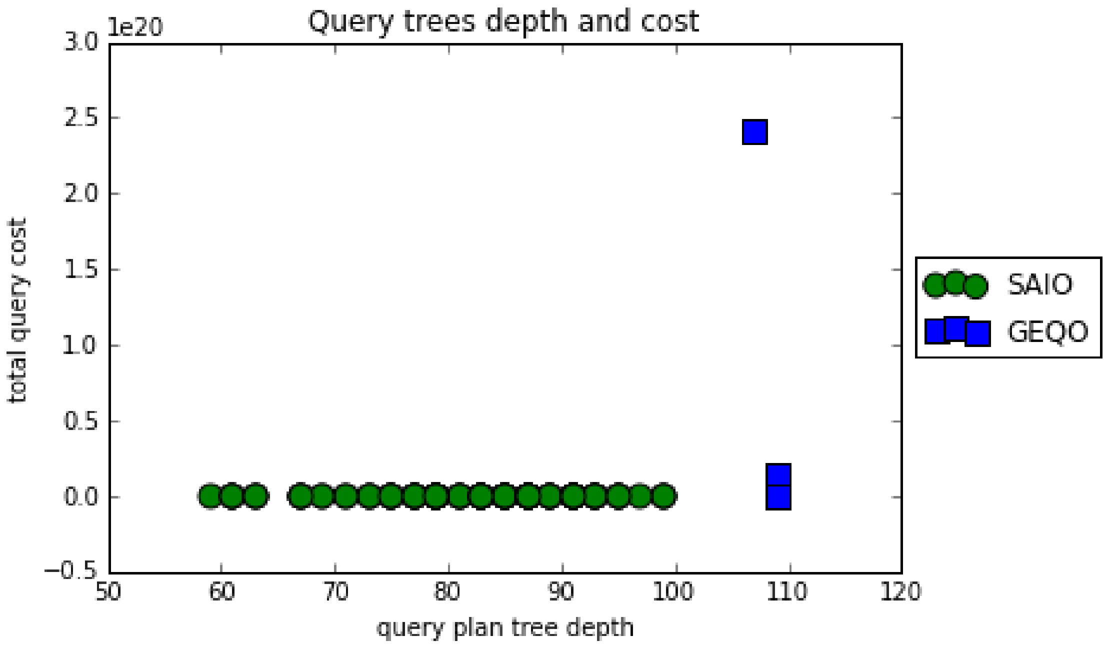

SAIO cost / avg(GEQO cost)
SAIO time / avg(GEQO time) [s/s]
SAIO memory / avg(GEQO memory) [MB/MB]
temperature reduction factor
equilibrium factor

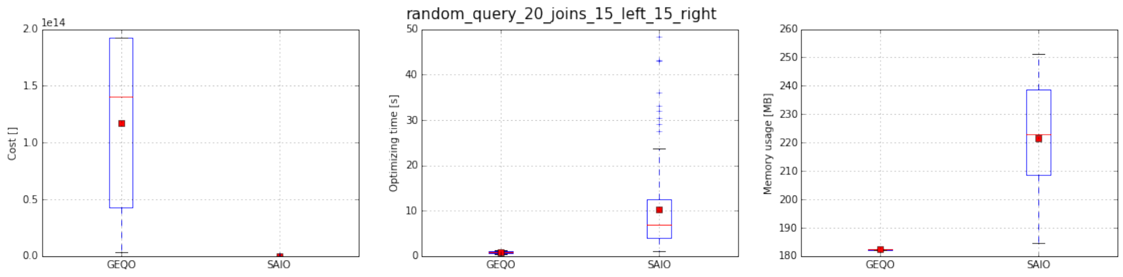

Query trees depth and cost
1e12
4.5
4.0
3.5
3.0
2.5
2.0
1.5
total query cost
30
40
50
60
70
80
90
100
110
query plan tree depth
SAIO
GEQO

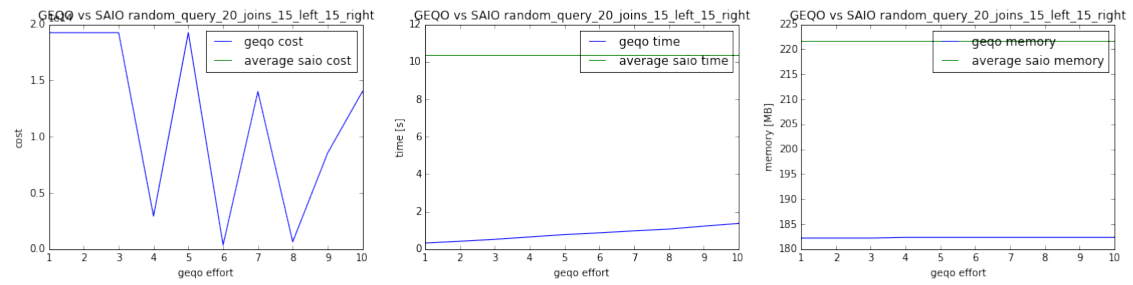

random_query_50_right_joins_no_constraints
1e20
Cost []
GEQO
SAIO
Optimizing time [s]
Memory usage [MB]

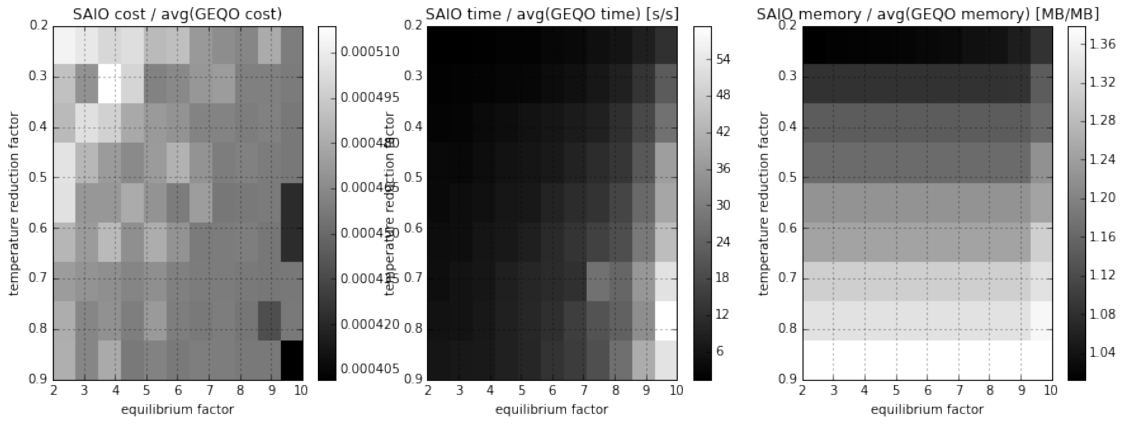

GEQO vs SAIO random_query_50_right_joins_no_constraints
1e20
cost
geqo cost
average saio cost
geqo effort
time [s]
geqo time
average saio time
memory [MB]
geqo memory
average saio memory

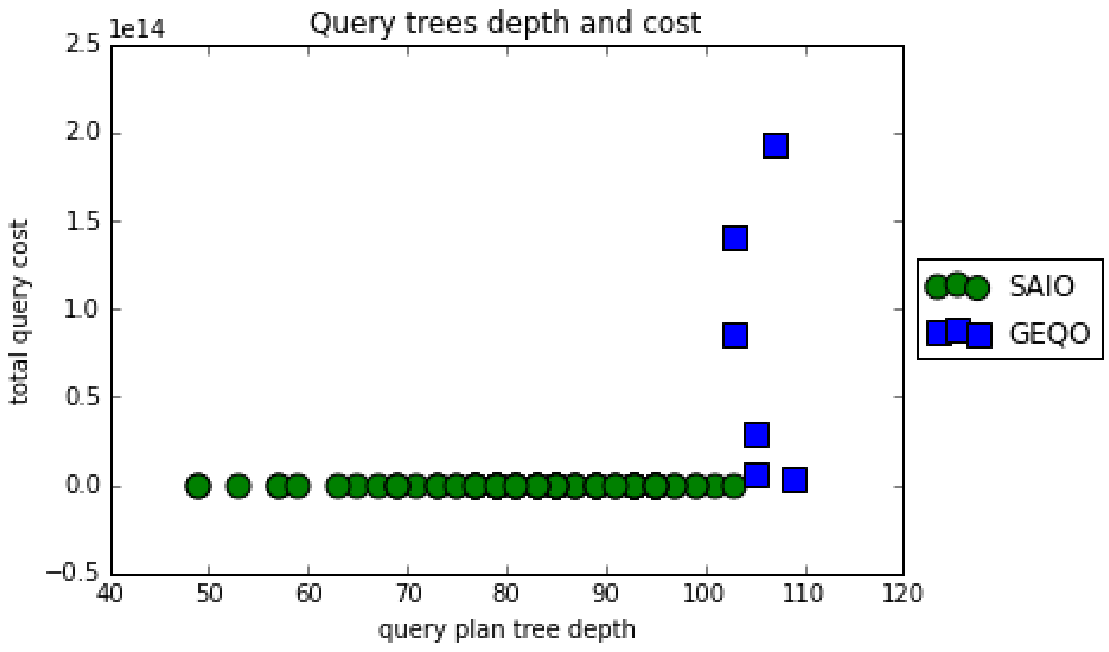

SAIO cost / avg(GEQO cost)
SAIO time / avg(GEQO time) [s/s]
SAIO memory / avg(GEQO memory) [MB/MB]
temperature reduction factor
equilibrium factor
1e−9


Query trees depth and cost
total query cost
query plan tree depth
1e20
SAIO
GEQO


random_query_20_joins_15_left_15_right
Cost []
Optimizing time [s]
Memory usage [MB]
1e14
GEQO
SAIO

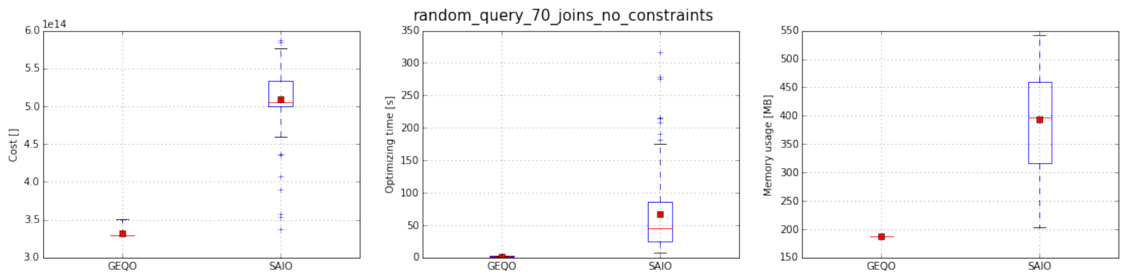

GEQO vs SAIO random_query_20_joins_15_left_15_right
1e14
cost
geqo effort
geqo cost
average saio cost
GEQO vs SAIO random_query_20_joins_15_left_15_right
time [s]
geqo effort
geqo time
average saio time
GEQO vs SAIO random_query_20_joins_15_left_15_right
memory [MB]
geqo effort
geqo memory
average saio memory

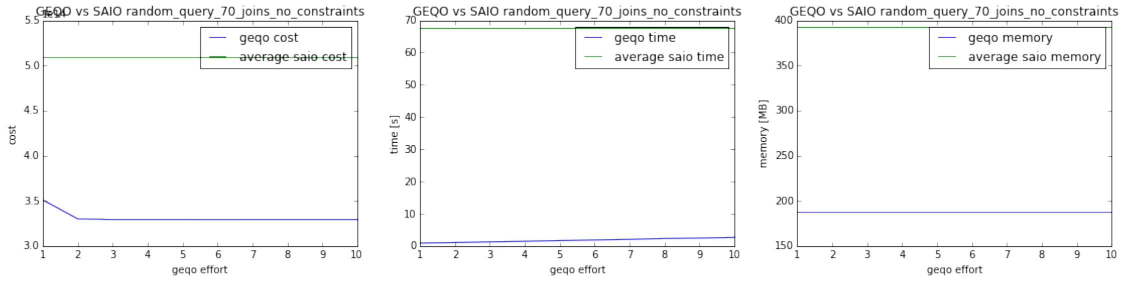

SAIO cost / avg(GEQO cost)
temperature reduction factor
equilibrium factor
SAIO time / avg(GEQO time) [s/s]
temperature reduction factor
equilibrium factor
SAIO memory / avg(GEQO memory) [MB/MB]
temperature reduction factor
equilibrium factor

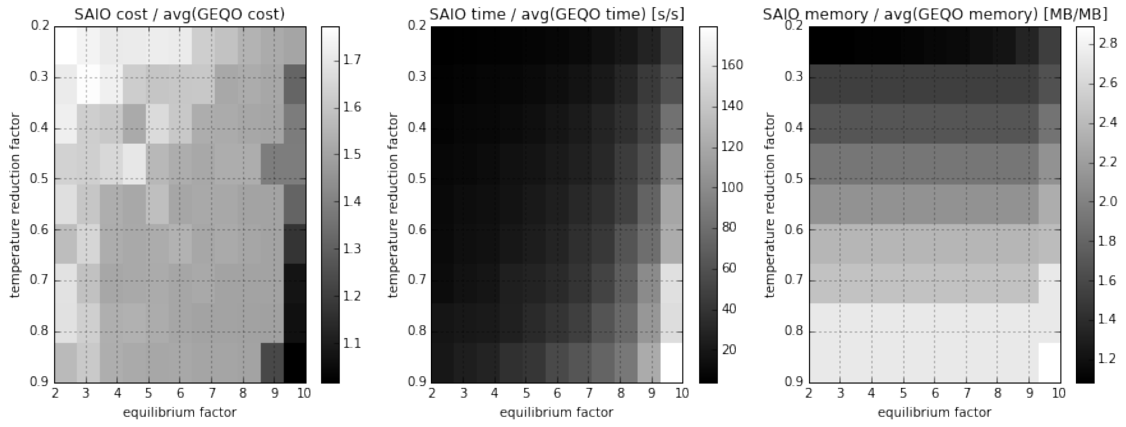

Query trees depth and cost
1e14
total query cost
query plan tree depth
SAIO
GEQO

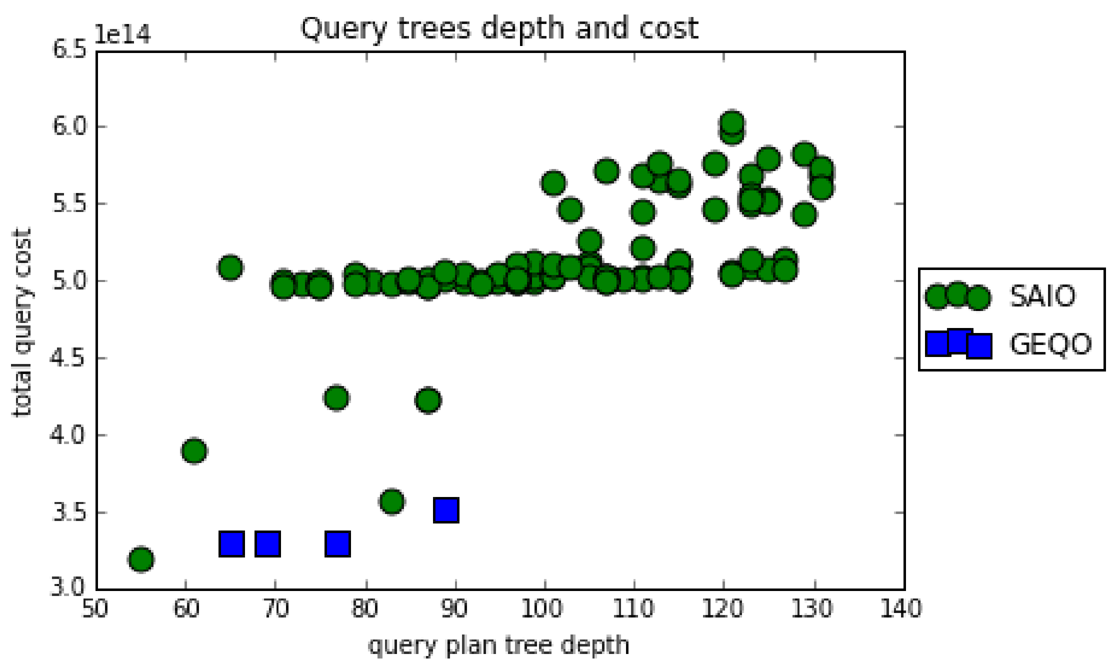
70 JOINS Observations from the previous case hold.

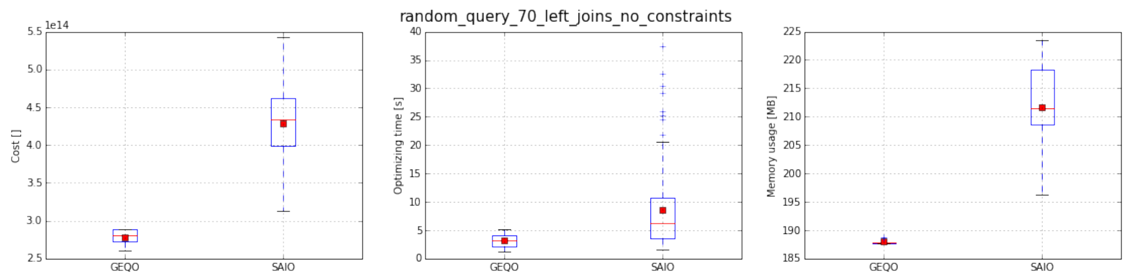
```
In [24]: display_info_for_test_case('random_query_70_joins_no_constraints')
         display_info_for_test_case('random_query_70_left_joins_no_constraints')
         display_info_for_test_case('random_query_70_right_joins_no_constraints')
         display_info_for_test_case('random_query_30_joins_20_left_20_right')
```

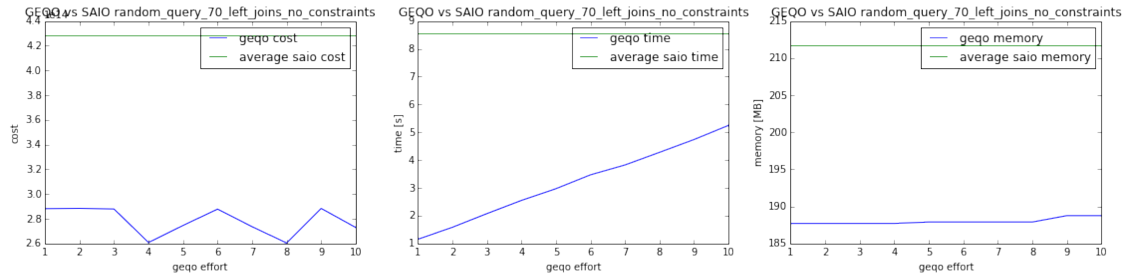


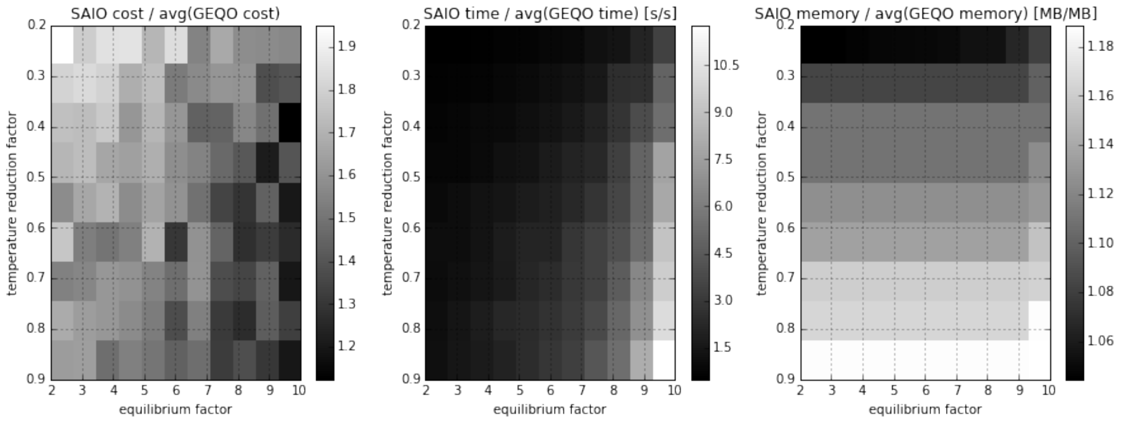


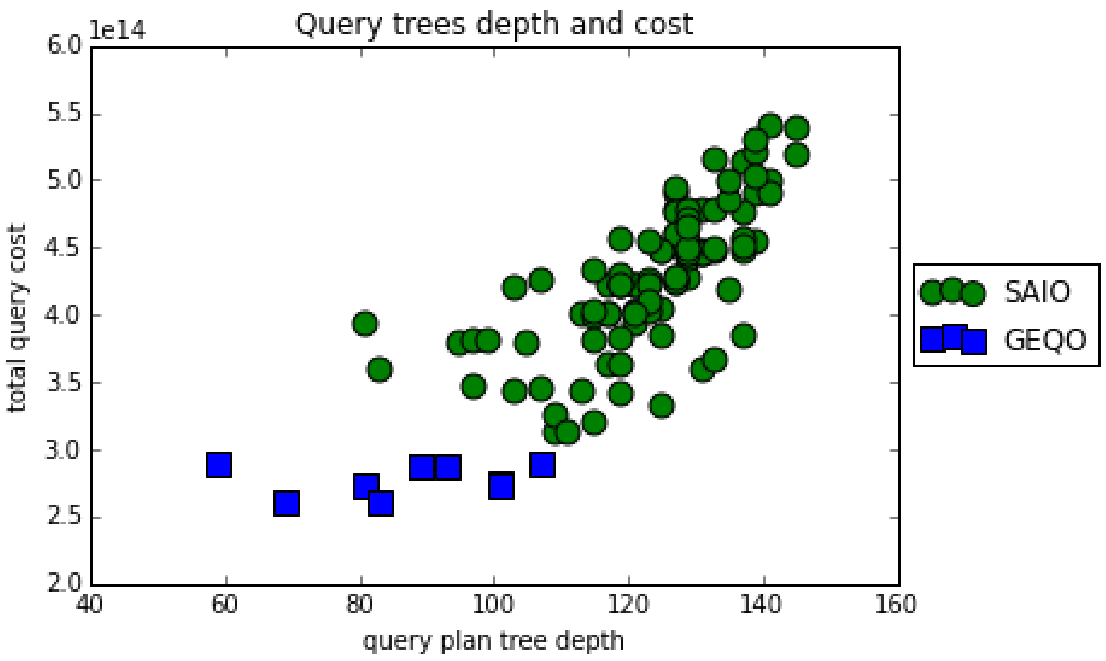


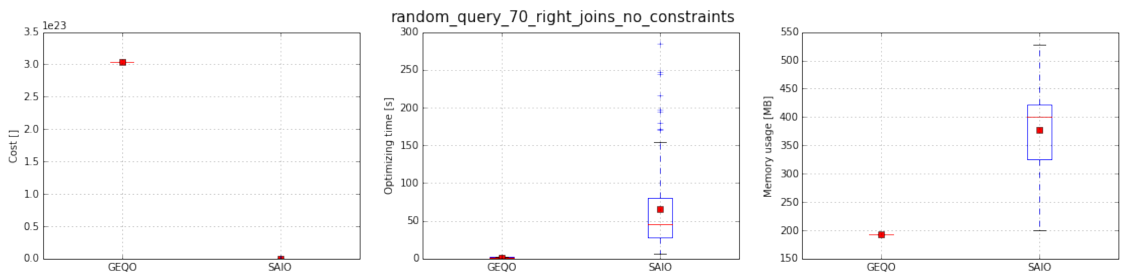

Query trees depth and cost
1e14
total query cost
query plan tree depth
SAIO
GEQO

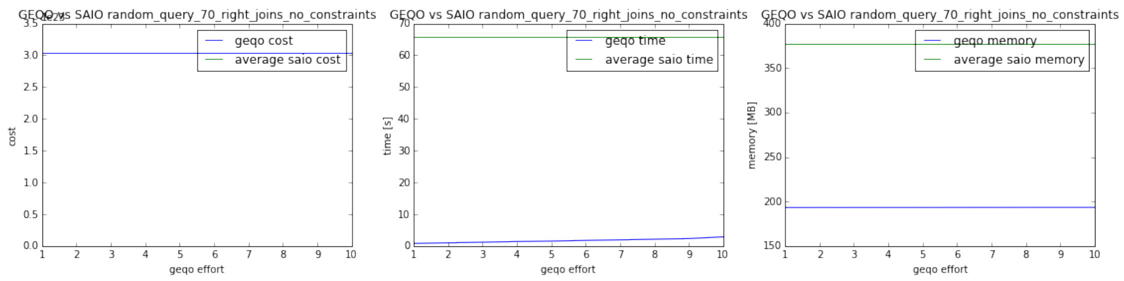

random_query_70_left_joins_no_constraints
1e14
Cost []
Optimizing time [s]
Memory usage [MB]
GEQO
SAIO

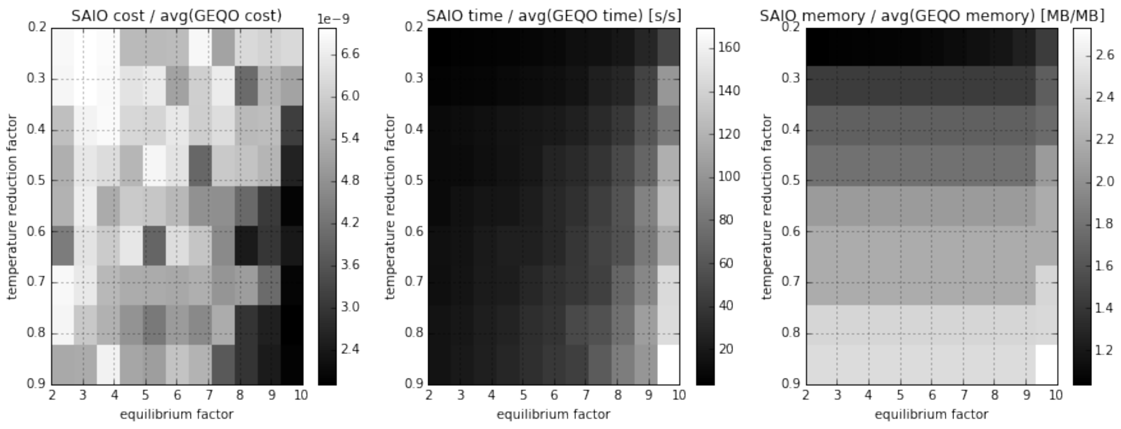

GEQO vs SAIO random_query_70_left_joins_no_constraints
1e14
cost
geqo cost
average saio cost
time [s]
geqo time
average saio time
memory [MB]
geqo memory
average saio memory
geqo effort

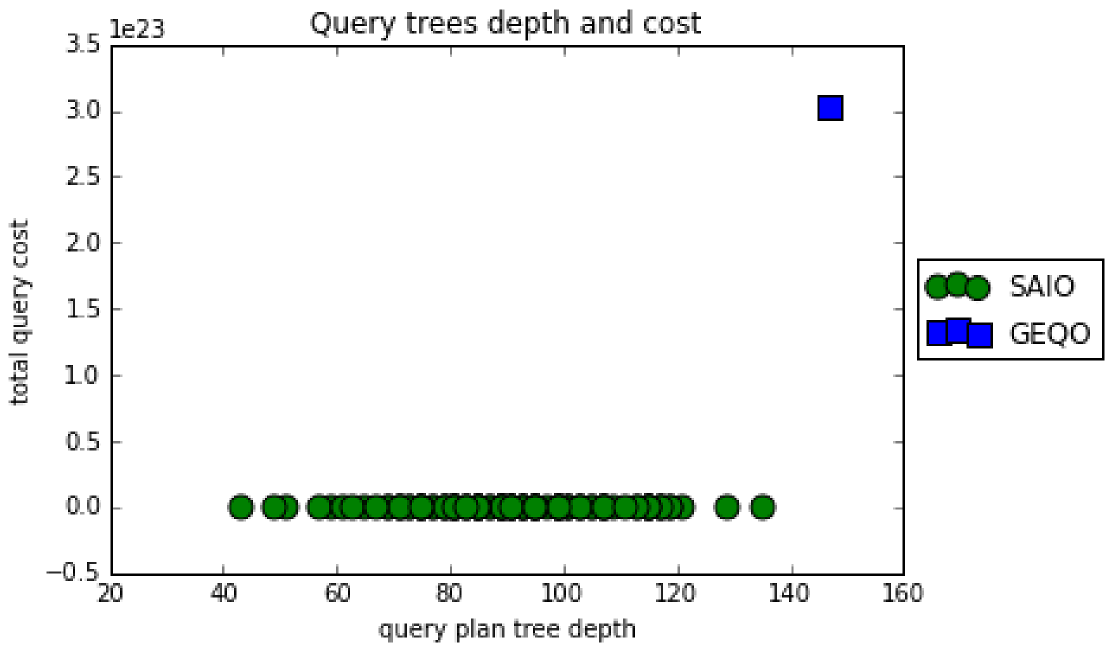

SAIO cost / avg(GEQO cost)
SAIO time / avg(GEQO time) [s/s]
SAIO memory / avg(GEQO memory) [MB/MB]
temperature reduction factor
equilibrium factor

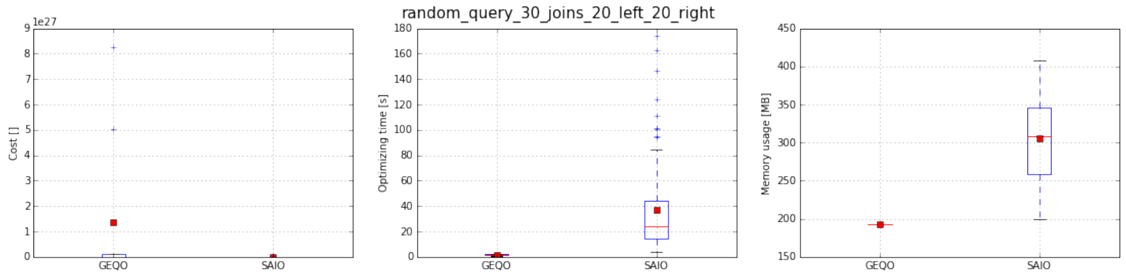

Query trees depth and cost
1e14
total query cost
query plan tree depth
SAIO
GEQO

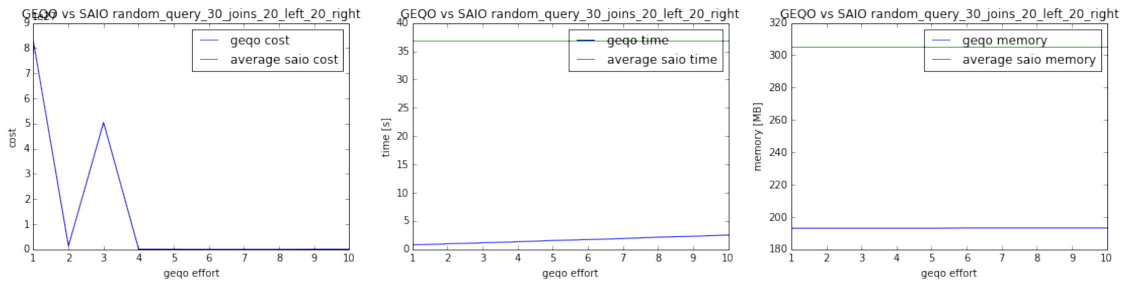

random_query_70_right_joins_no_constraints
1e23
Cost []
Optimizing time [s]
Memory usage [MB]
GEQO
SAIO

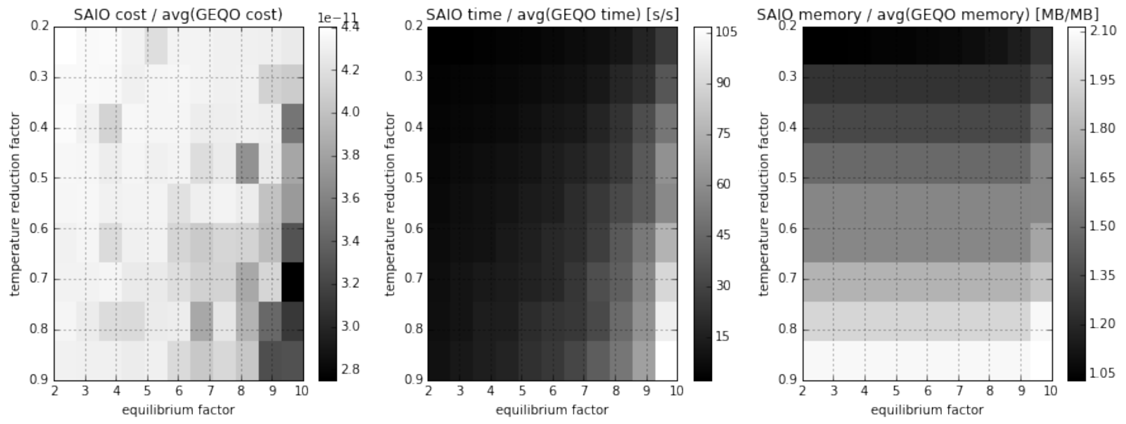

GEQO vs SAIO random_query_70_right_joins_no_constraints
1e23
geqo cost
average saio cost
cost
geqo effort
geqo time
average saio time
time [s]
geqo memory
average saio memory
memory [MB]

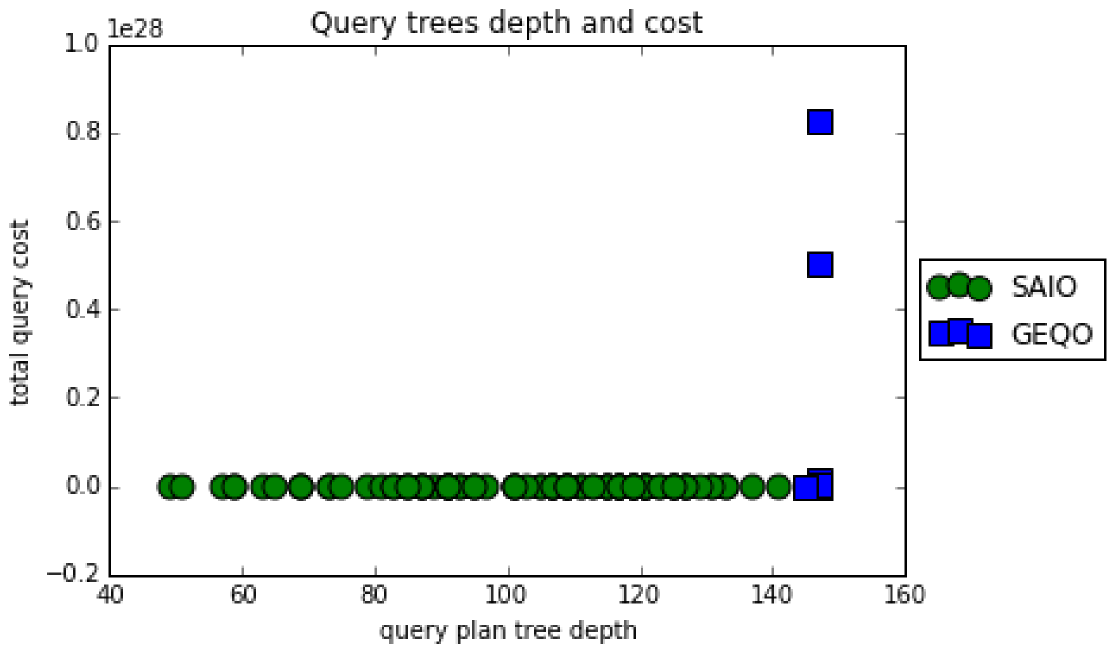

SAIO cost / avg(GEQO cost)
1e−9
SAIO time / avg(GEQO time) [s/s]
SAIO memory / avg(GEQO memory) [MB/MB]
temperature reduction factor
equilibrium factor


Query trees depth and cost
1e23
total query cost
query plan tree depth
SAIO
GEQO


random_query_30_joins_20_left_20_right
1e27
Cost []
Optimizing time [s]
Memory usage [MB]
GEQO
SAIO


GEQO vs SAIO random_query_30_joins_20_left_20_right
1e27
cost
geqo cost
average saio cost
time [s]
geqo time
average saio time
memory [MB]
geqo memory
average saio memory
geqo effort

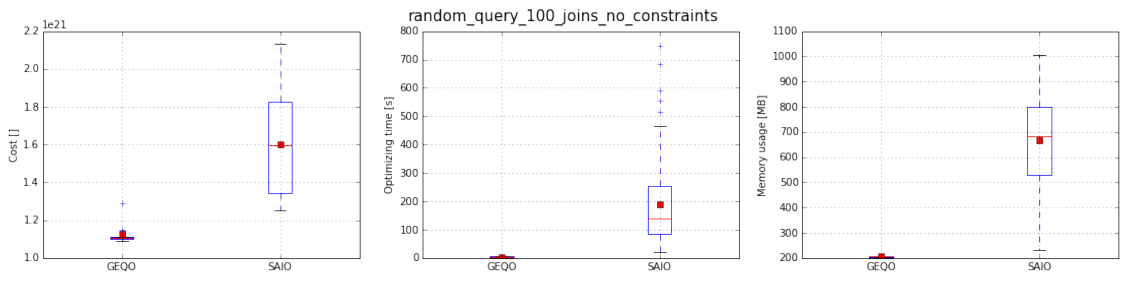

SAIO cost / avg(GEQO cost)
1e−11
SAIO time / avg(GEQO time) [s/s]
SAIO memory / avg(GEQO memory) [MB/MB]
temperature reduction factor
equilibrium factor

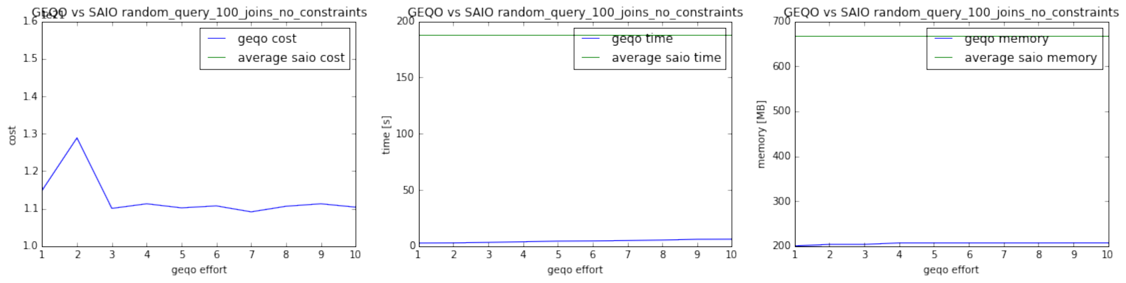


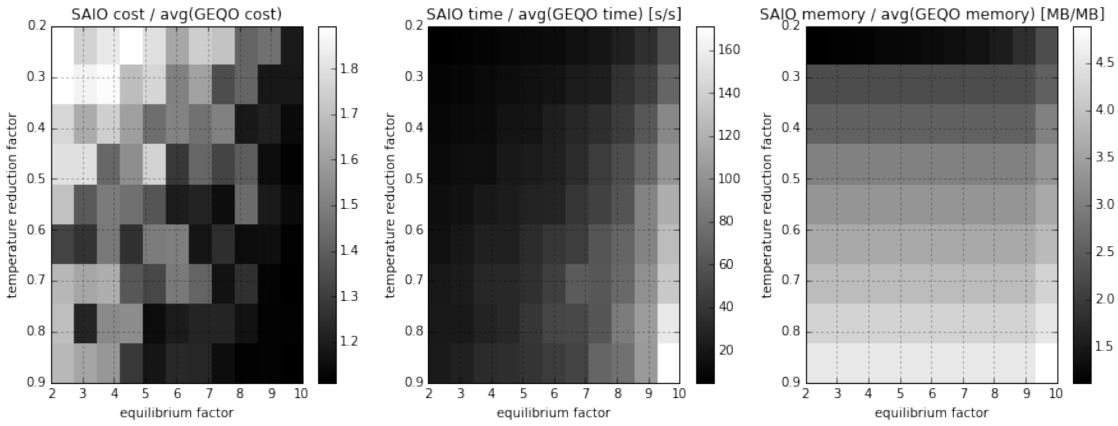
100 JOINS These queries would probably not complete during our lifetime, however the observations from the previous cases hold.

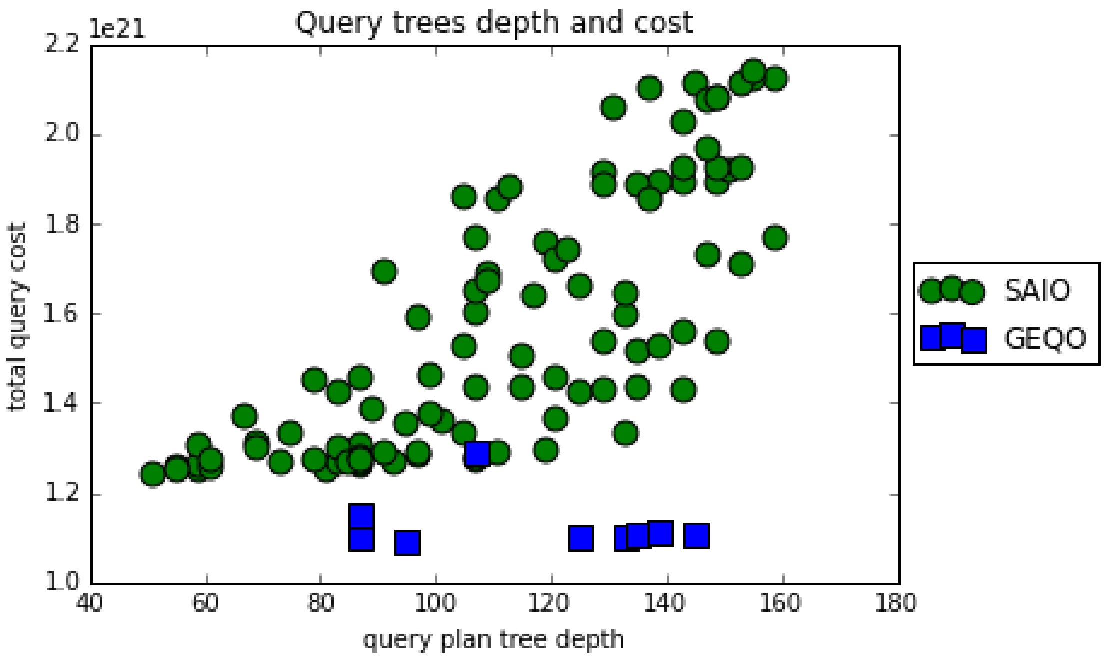
GEQO wins for JOINS and LEFT JOINS, SAIO wins for queries that contain RIGHT JOINS.

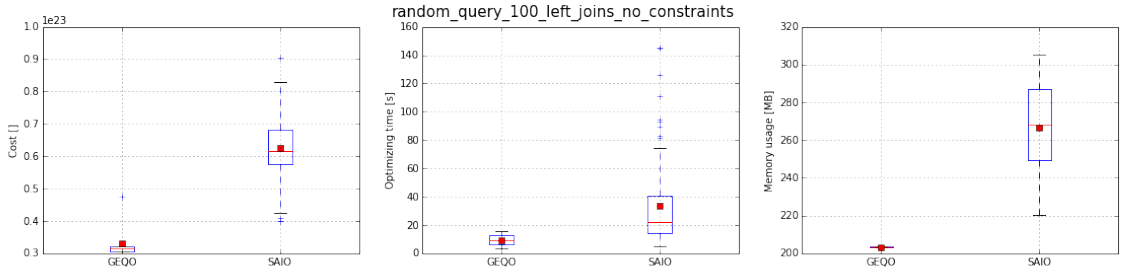
```
In [52]: display_info_for_test_case('random_query_100_joins_no_constraints')
         display_info_for_test_case('random_query_100_left_joins_no_constraints')
         display_info_for_test_case('random_query_100_right_joins_no_constraints')
         display_info_for_test_case('random_query_40_joins_30_left_30_right')
```

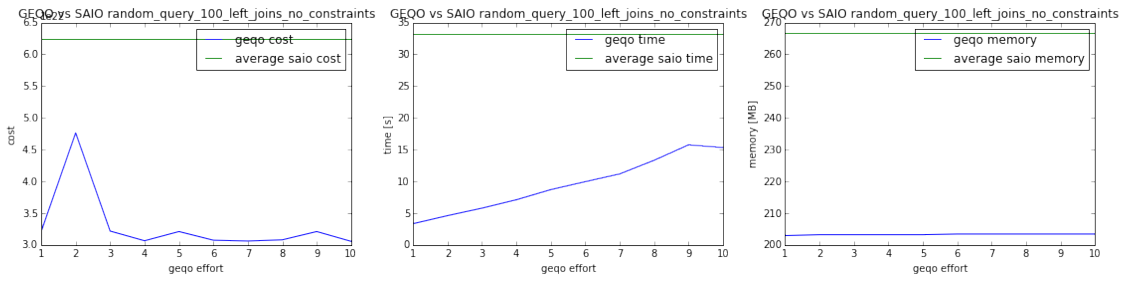


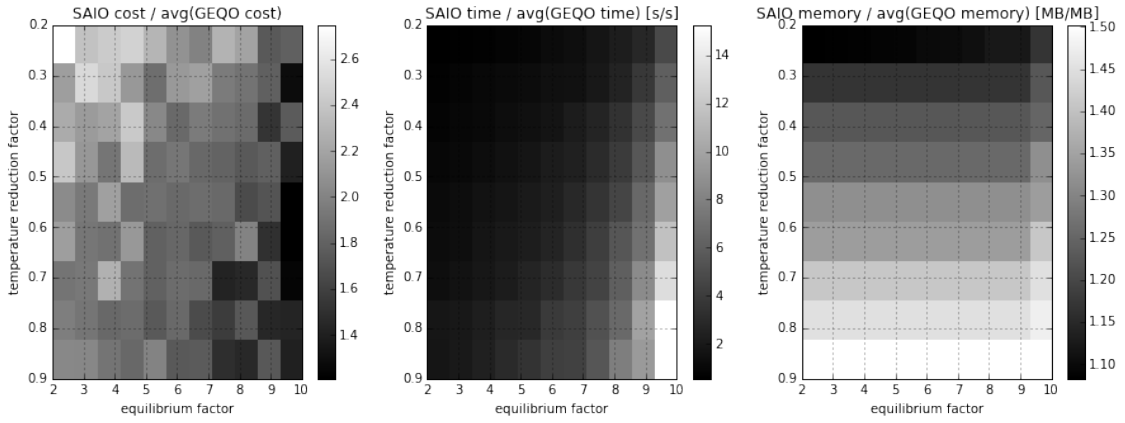


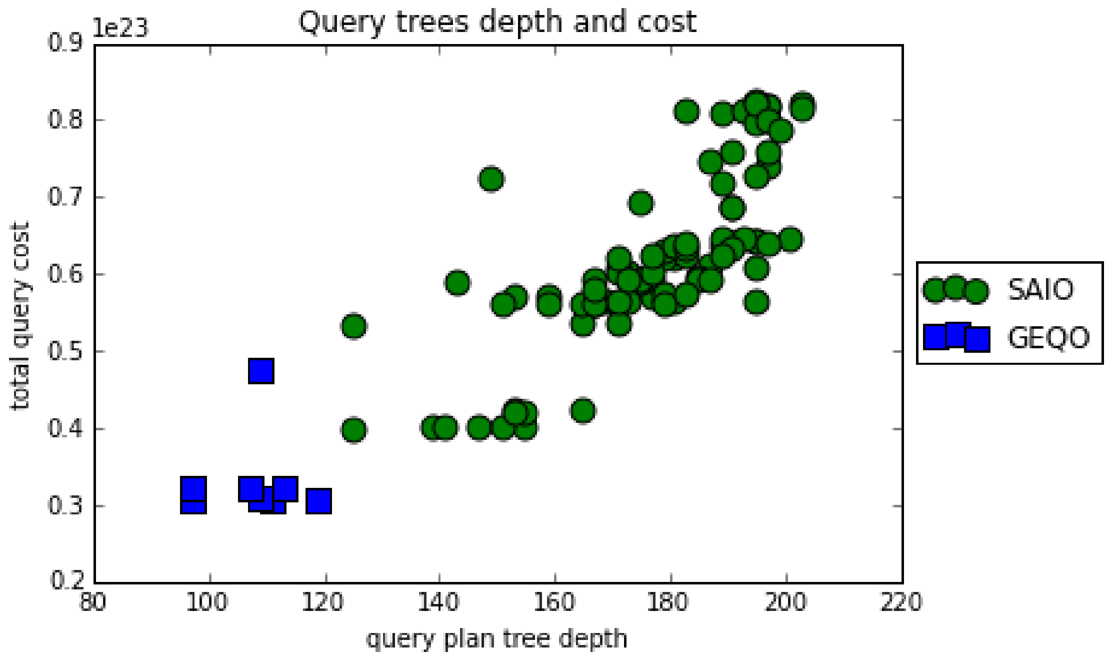

SAIO cost / avg(GEQO cost)
SAIO time / avg(GEQO time) [s/s]
SAIO memory / avg(GEQO memory) [MB/MB]
temperature reduction factor
equilibrium factor

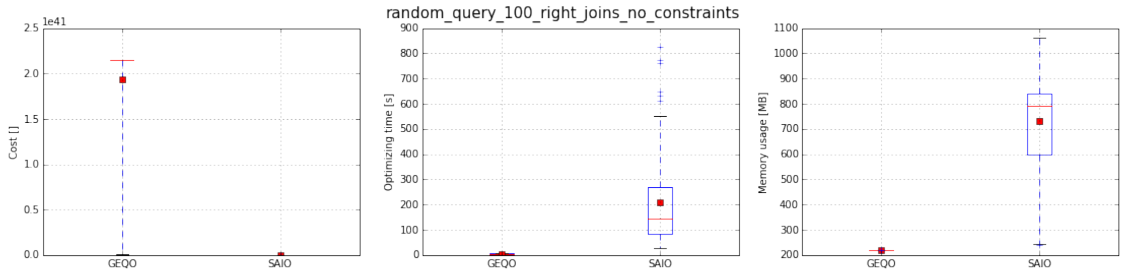

Query trees depth and cost
1e21
total query cost
query plan tree depth
SAIO
GEQO

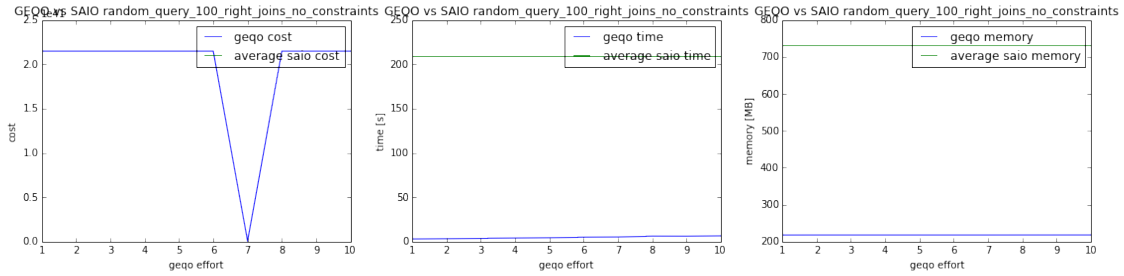

random_query_100_left_joins_no_constraints
1e23
Cost []
Optimizing time [s]
Memory usage [MB]
GEQO
SAIO

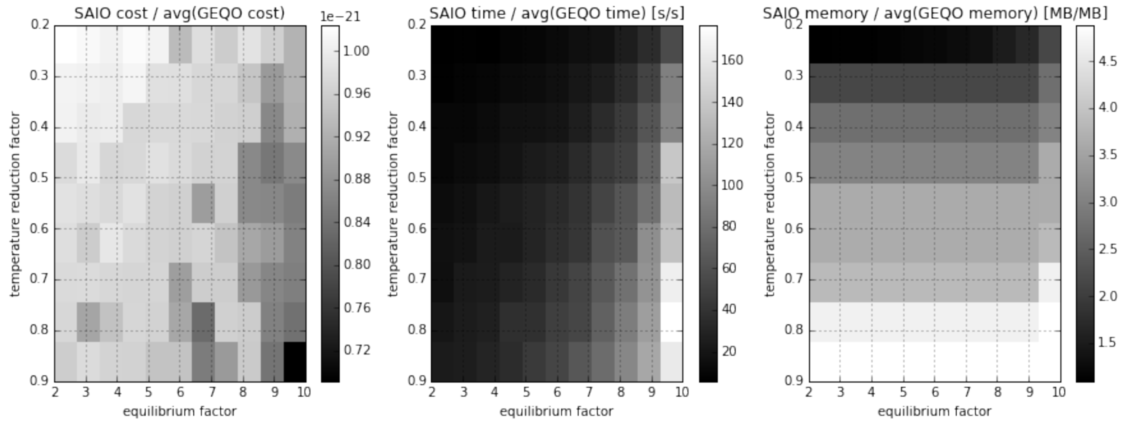

GEQO vs SAIO random_query_100_left_joins_no_constraints
1e22
geqo cost
average saio cost
cost
geqo effort
GEQO vs SAIO random_query_100_left_joins_no_constraints
geqo time
average saio time
time [s]
geqo effort
GEQO vs SAIO random_query_100_left_joins_no_constraints
geqo memory
average saio memory
memory [MB]
geqo effort

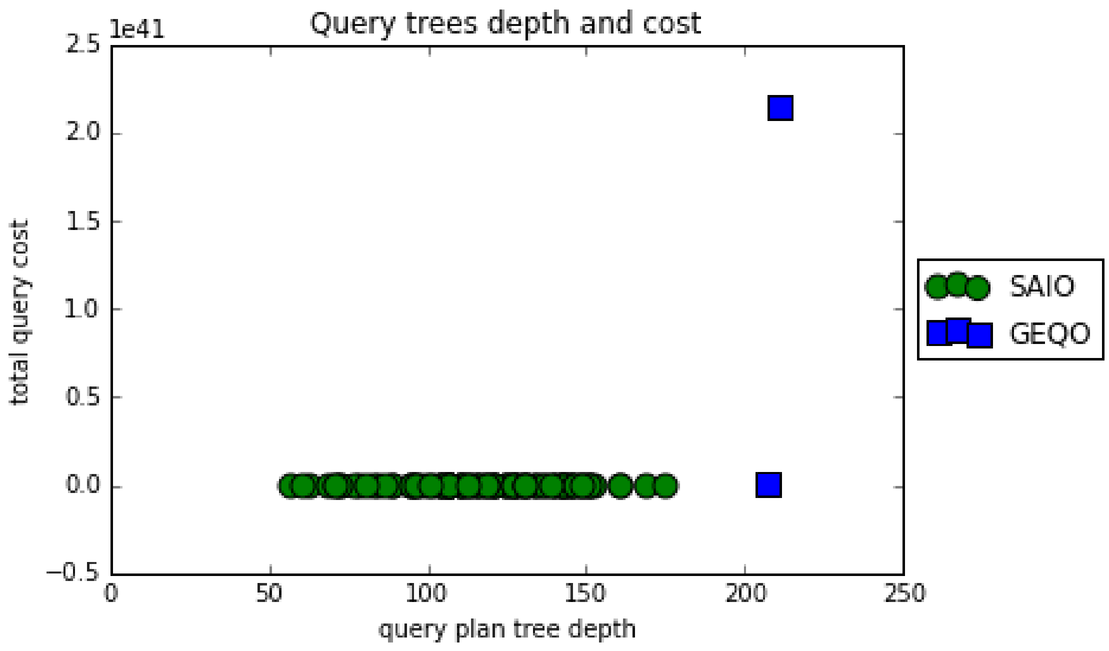

SAIO cost / avg(GEQO cost)
temperature reduction factor
equilibrium factor
SAIO time / avg(GEQO time) [s/s]
temperature reduction factor
equilibrium factor
SAIO memory / avg(GEQO memory) [MB/MB]
temperature reduction factor
equilibrium factor

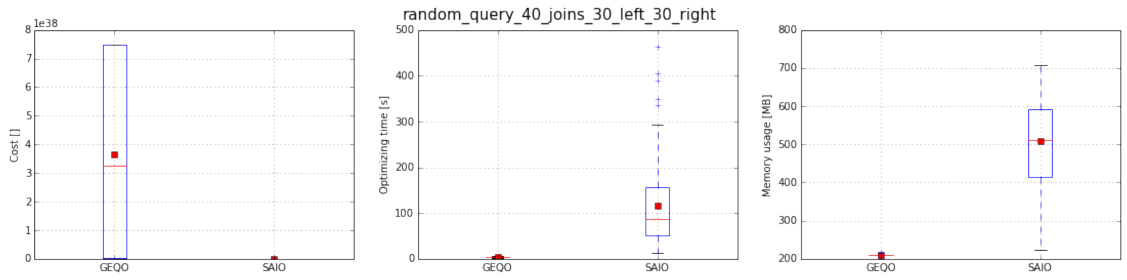

Query trees depth and cost
1e23
total query cost
query plan tree depth
SAIO
GEQO

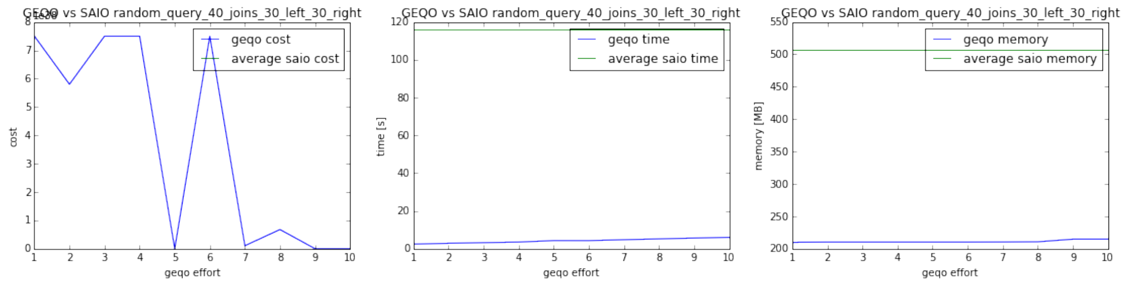

random_query_100_right_joins_no_constraints
1e41
Cost []
Optimizing time [s]
Memory usage [MB]
GEQO
SAIO

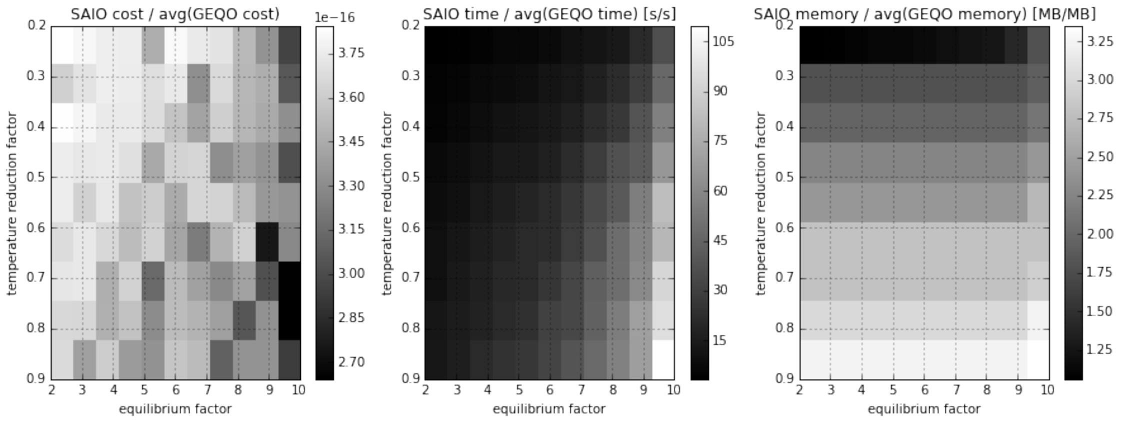

GEQO vs SAIO random_query_100_right_joins_no_constraints
geqo cost
average saio cost
geqo time
average saio time
geqo memory
average saio memory
cost
time [s]
memory [MB]
geqo effort

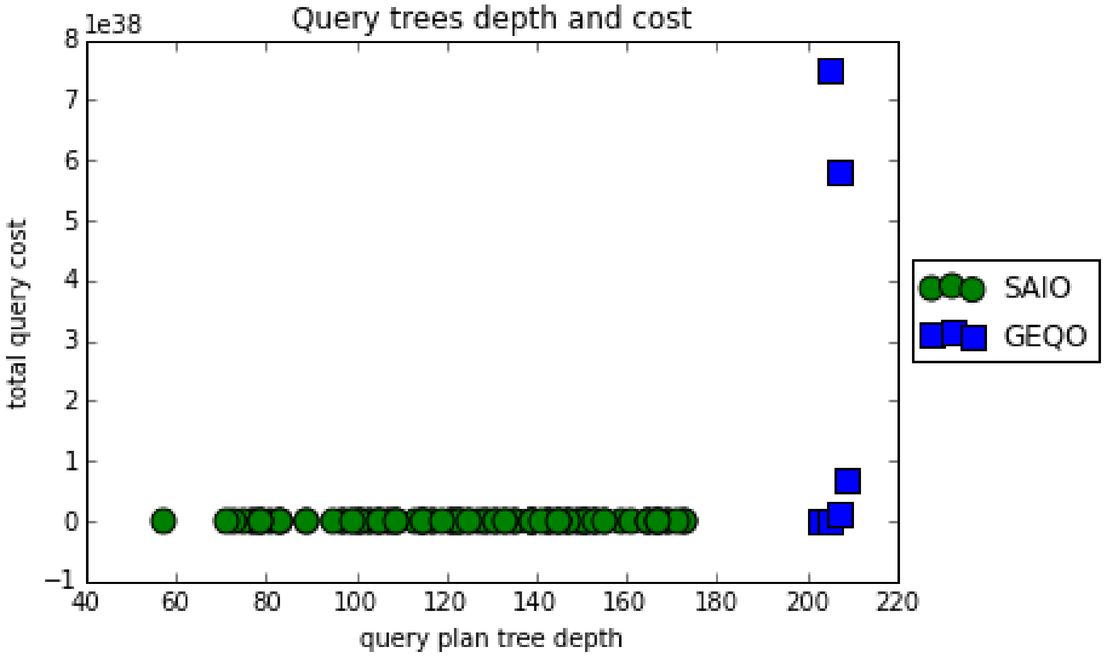

SAIO cost / avg(GEQO cost)
SAIO time / avg(GEQO time) [s/s]
SAIO memory / avg(GEQO memory) [MB/MB]
1e−21
temperature reduction factor
equilibrium factor


Query trees depth and cost
1e41
total query cost
query plan tree depth
SAIO
GEQO


random_query_40_joins_30_left_30_right
1e38
Cost []
Optimizing time [s]
Memory usage [MB]
GEQO
SAIO

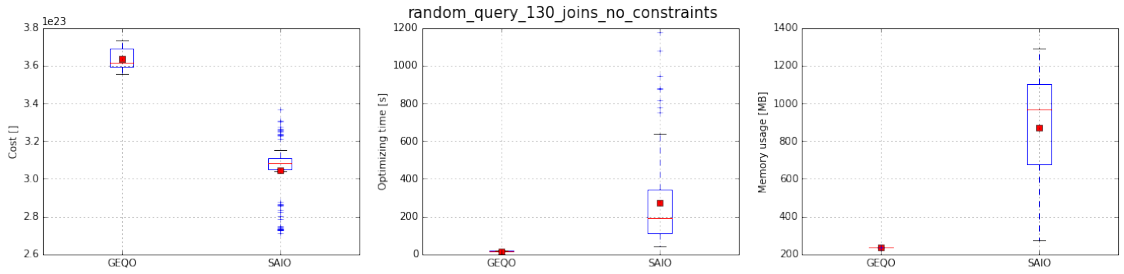

GEQO vs SAIO random_query_40_joins_30_left_30_right
geqo cost
average saio cost
cost
geqo effort
geqo time
average saio time
time [s]
geqo memory
average saio memory
memory [MB]

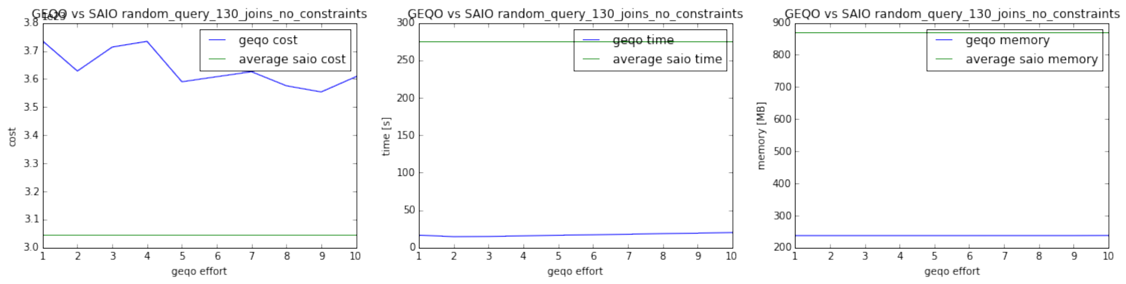




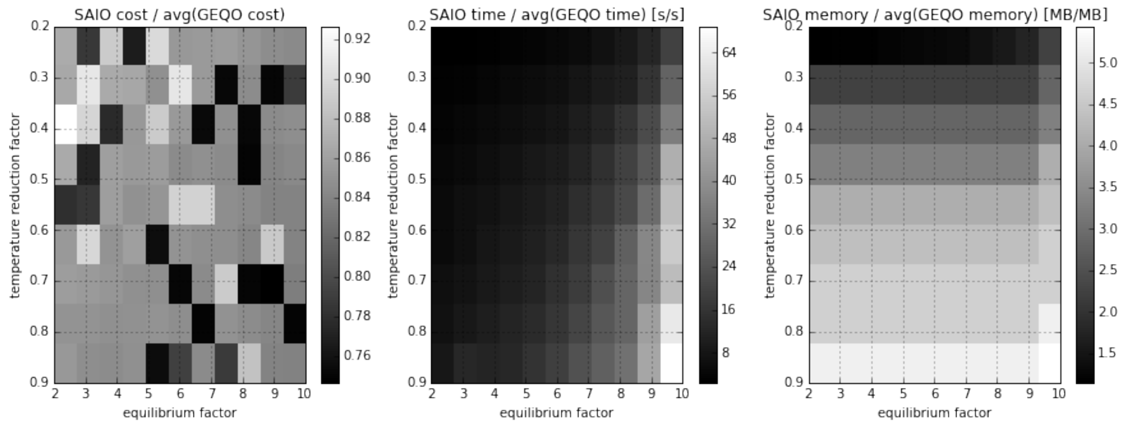
130 joins Previous observations hold.

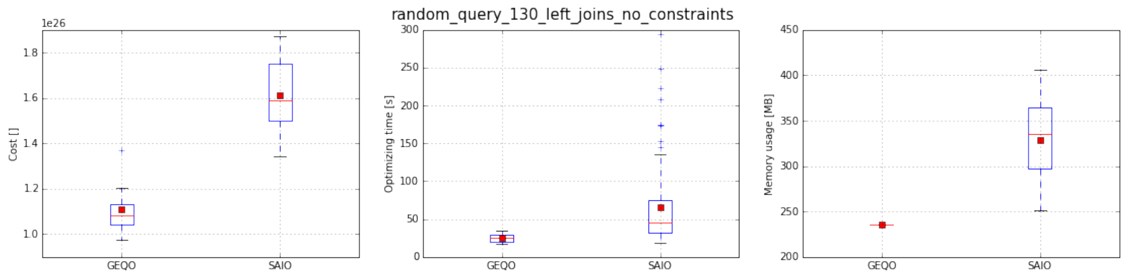
```
In [16]: display_info_for_test_case('random_query_130_joins_no_constraints')
         display_info_for_test_case('random_query_130_left_joins_no_constraints')
         display_info_for_test_case('random_query_130_right_joins_no_constraints')
         display_info_for_test_case('random_query_50_joins_40_left_40_right')
```

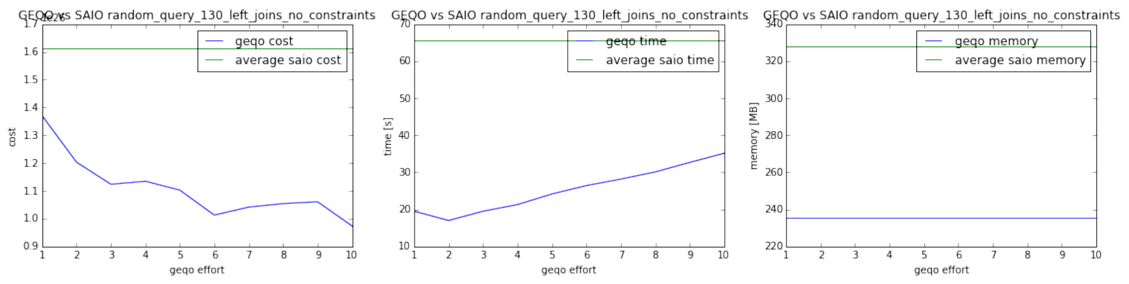

random_query_130_joins_no_constraints
1e23
Cost []
Optimizing time [s]
Memory usage [MB]
GEQO
SAIO


GEQO vs SAIO random_query_130_joins_no_constraints
1e23
cost
geqo cost
average saio cost
time [s]
geqo time
average saio time
memory [MB]
geqo memory
average saio memory
geqo effort

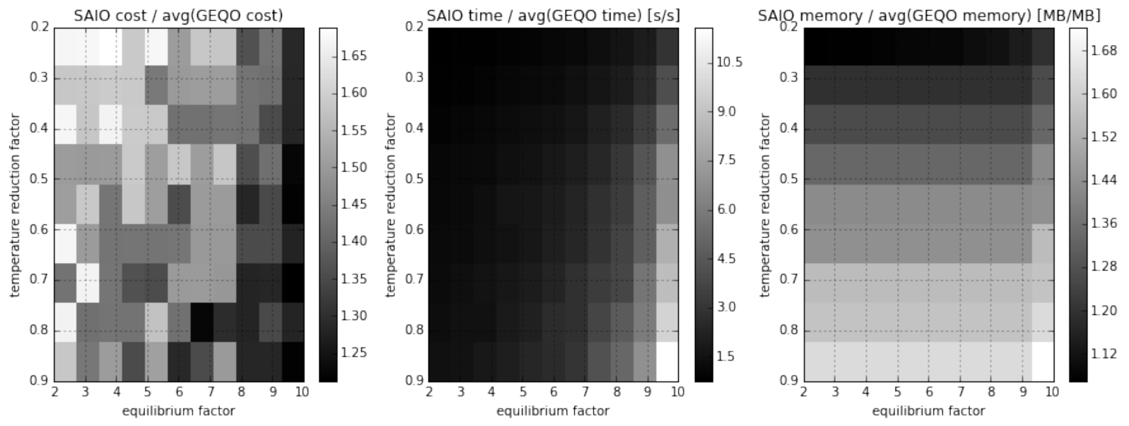
99

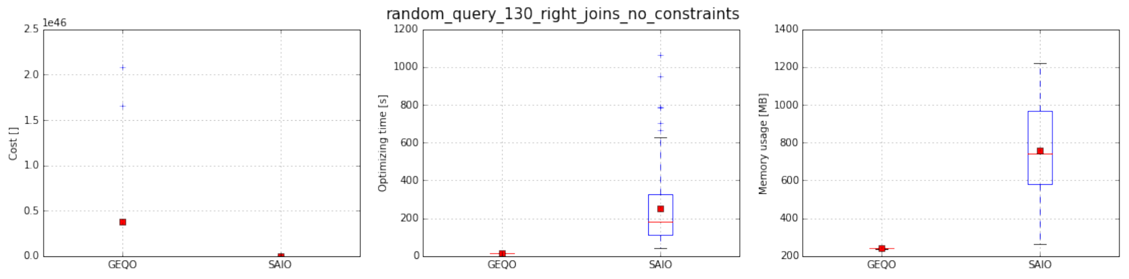

SAIO cost / avg(GEQO cost)
SAIO time / avg(GEQO time) [s/s]
SAIO memory / avg(GEQO memory) [MB/MB]
temperature reduction factor
equilibrium factor

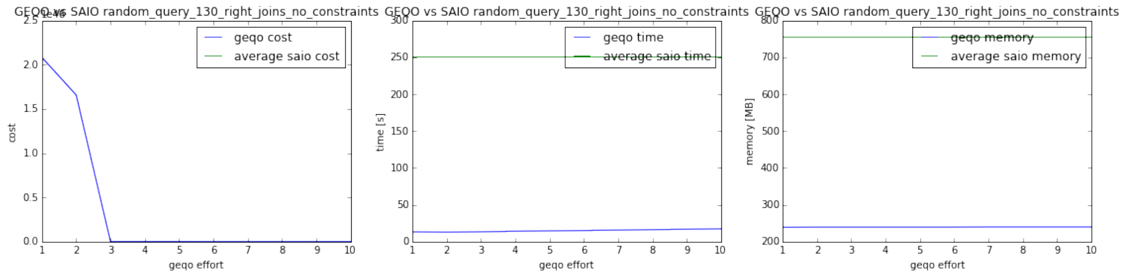

random_query_130_left_joins_no_constraints
1e26
Cost []
Optimizing time [s]
Memory usage [MB]
GEQO
SAIO


GEQO vs SAIO random_query_130_left_joins_no_constraints
1e26
cost
geqo effort
geqo cost
average saio cost
GEQO vs SAIO random_query_130_left_joins_no_constraints
time [s]
geqo effort
geqo time
average saio time
GEQO vs SAIO random_query_130_left_joins_no_constraints
memory [MB]
geqo effort
geqo memory
average saio memory

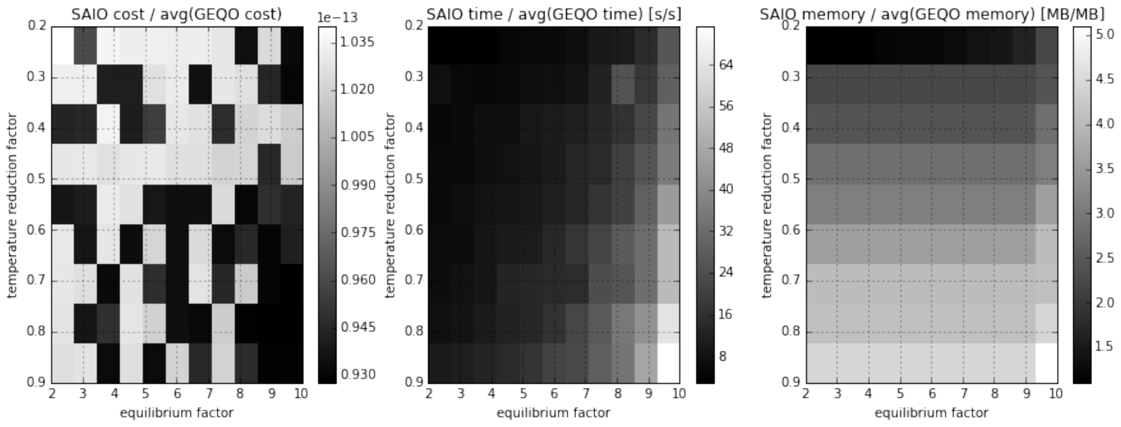
99

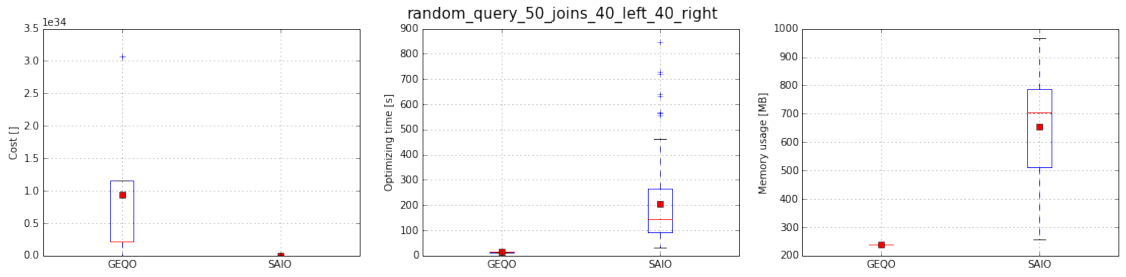

SAIO cost / avg(GEQO cost)
temperature reduction factor
equilibrium factor
SAIO time / avg(GEQO time) [s/s]
temperature reduction factor
equilibrium factor
SAIO memory / avg(GEQO memory) [MB/MB]
temperature reduction factor
equilibrium factor

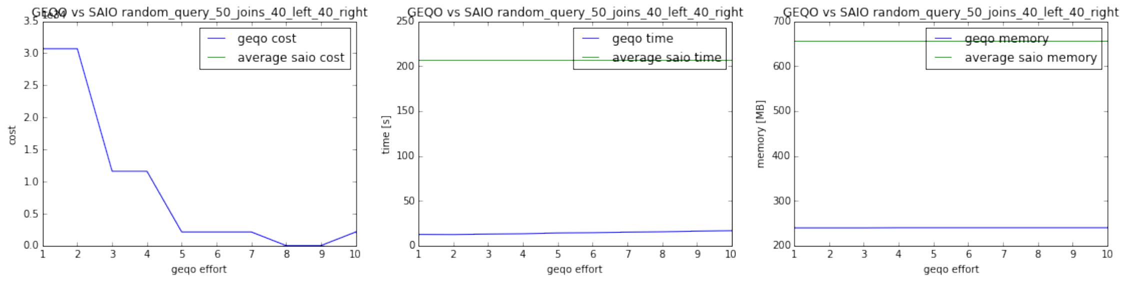

random_query_130_right_joins_no_constraints
1e46
Cost []
GEQO
SAIO
Optimizing time [s]
GEQO
SAIO
Memory usage [MB]
GEQO
SAIO


GEQO vs SAIO random_query_130_right_joins_no_constraints
1e46
cost
geqo effort
geqo cost
average saio cost
GEQO vs SAIO random_query_130_right_joins_no_constraints
time [s]
geqo effort
geqo time
average saio time
GEQO vs SAIO random_query_130_right_joins_no_constraints
memory [MB]
geqo effort
geqo memory
average saio memory

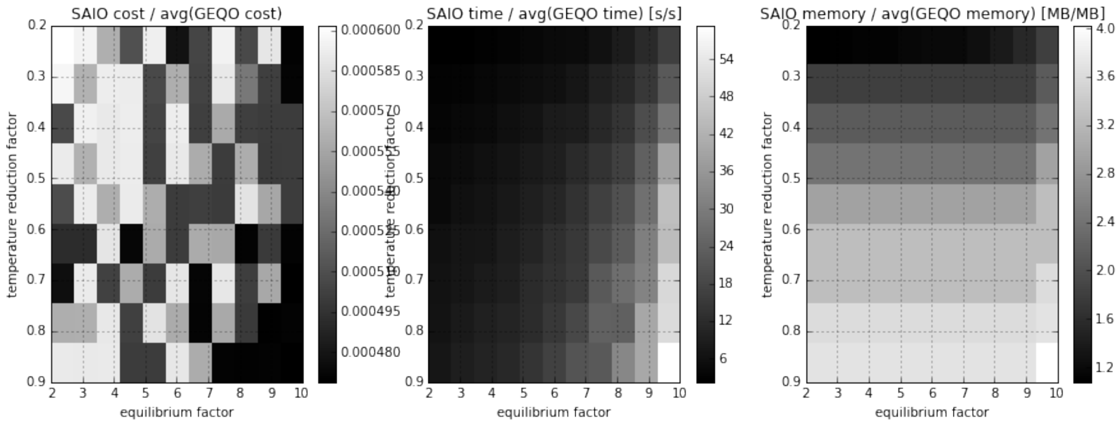
99


SAIO cost / avg(GEQO cost)
SAIO time / avg(GEQO time) [s/s]
SAIO memory / avg(GEQO memory) [MB/MB]
temperature reduction factor
equilibrium factor
1e−13


random_query_50_joins_40_left_40_right
1e34
Cost []
Optimizing time [s]
Memory usage [MB]
GEQO
SAIO


GEQO vs SAIO random_query_50_joins_40_left_40_right
geqo cost
average saio cost
geqo time
average saio time
geqo memory
average saio memory
cost
time [s]
memory [MB]
geqo effort

99



4.2.4 nested queries

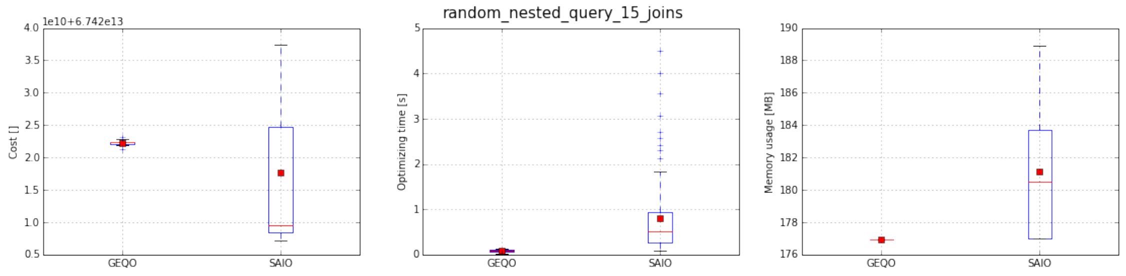
Example:

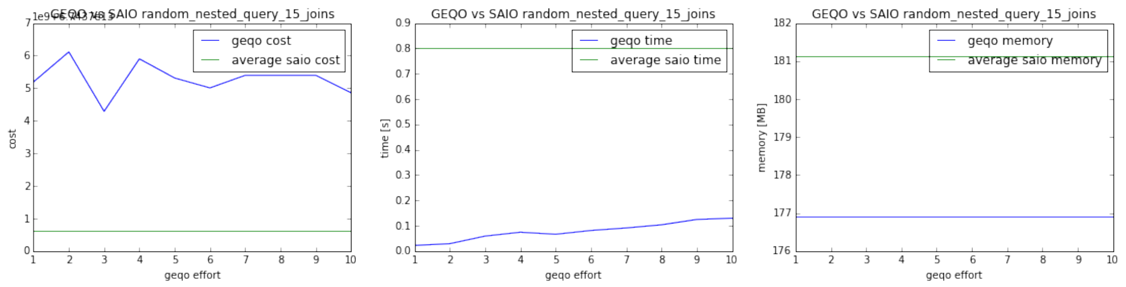
```
SELECT * FROM  table_0  JOIN table_12 ON table_0.col_0 = table_12.col_3
 JOIN table_11 ON table_12.col_1 = table_11.col_0
 JOIN table_5 ON table_0.col_1 = table_5.col_1
 JOIN table_13 ON table_0.col_0 = table_13.col_1
 JOIN table_7 ON table_12.col_3 = table_7.col_1
 JOIN table_8 ON table_11.col_0 = table_8.col_0
 JOIN table_19 ON table_12.col_3 = table_19.col_1
 JOIN table_3 ON table_12.col_1 = table_3.col_1
 JOIN table_1 ON table_0.col_3 = table_1.col_2
 JOIN table_17 ON table_5.col_1 = table_17.col_0
 JOIN (
     SELECT * FROM  table_0  JOIN table_5 ON table_0.col_1 = table_5.col_0
    ) AS subquery_table_0_299
                    ON TRUE
 JOIN table_15 ON table_12.col_0 = table_15.col_2
 JOIN table_6 ON table_12.col_3 = table_6.col_0
 JOIN table_16 ON table_0.col_0 = table_16.col_0;
```

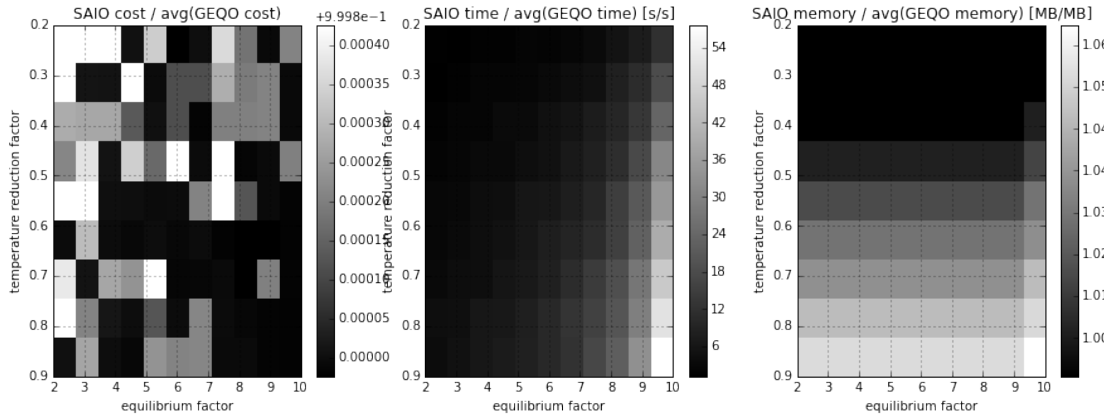
15-20 JOINS For nested queries with 15-20 JOINS we can observe similar situation as for flat queries with such parameters.

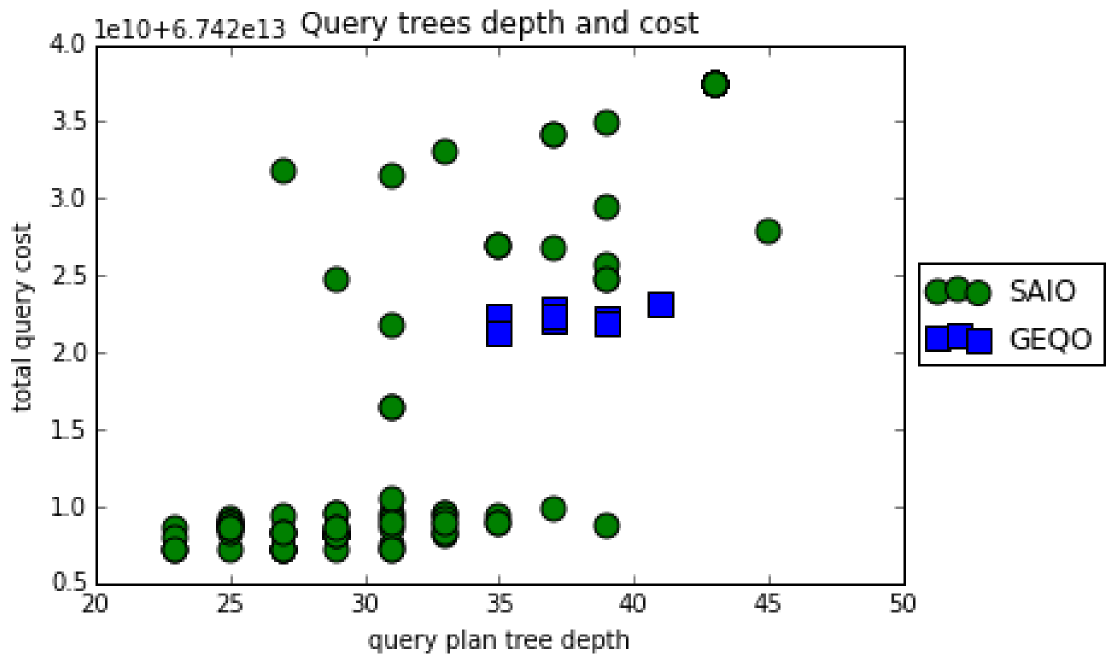
SAIO wins for queries that use JOINS or contain RIGHT JOINS. GEQO wins at optimizing queries with LEFT JOINS.

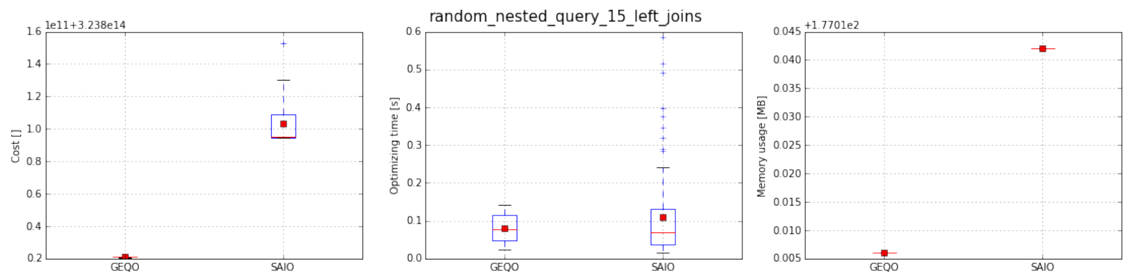
```
In [8]: display_info_for_test_case('random_nested_query_15_joins')
        display_info_for_test_case('random_nested_query_15_left_joins')
        display_info_for_test_case('random_nested_query_15_right_joins')
        display_info_for_test_case('random_nested_query_5_joins_5_left_5_right')
```

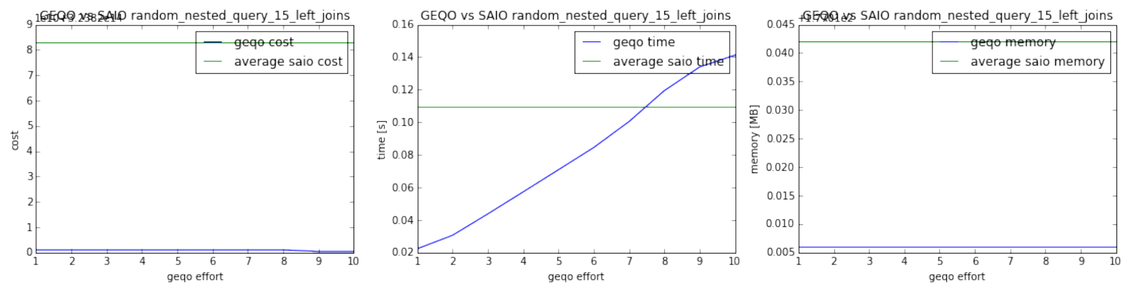

random_nested_query_15_joins
1e10+6.742e13
Cost []
Optimizing time [s]
Memory usage [MB]
GEQO
SAIO

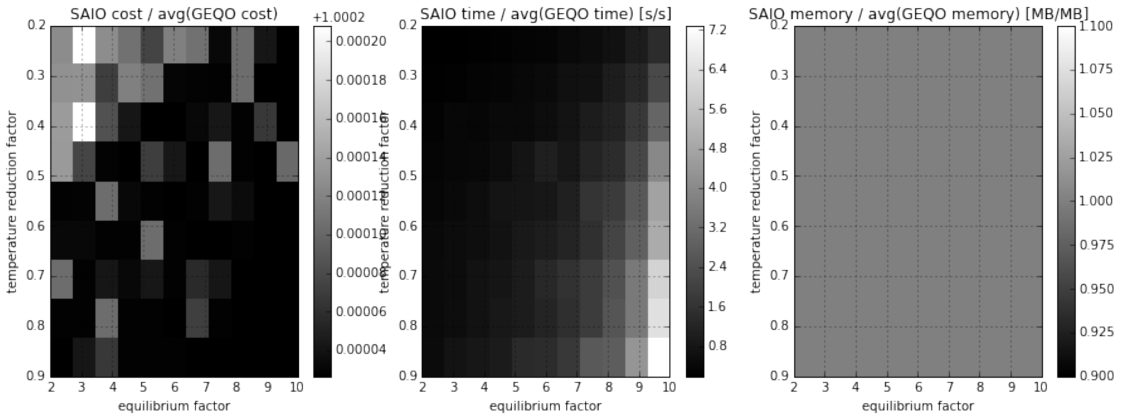

GEQO vs SAIO random_nested_query_15_joins
1e9+6.7437e13
cost
geqo cost
average saio cost
time [s]
geqo time
average saio time
memory [MB]
geqo memory
average saio memory
geqo effort

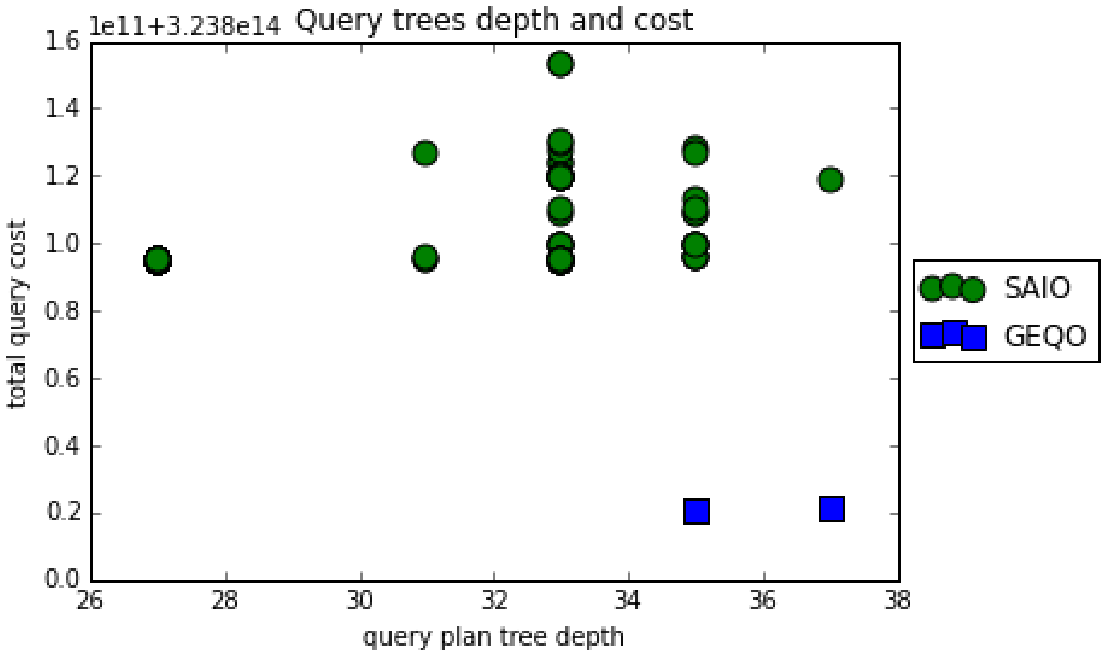

SAIO cost / avg(GEQO cost)
+9.998e-1
SAIO time / avg(GEQO time) [s/s]
SAIO memory / avg(GEQO memory) [MB/MB]
temperature reduction factor
equilibrium factor

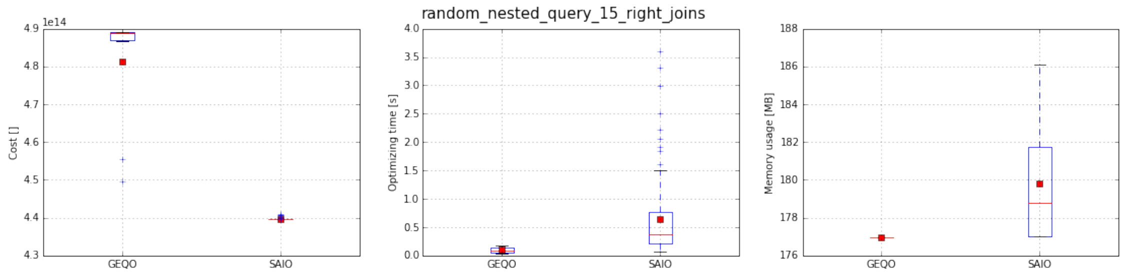

Query trees depth and cost
1e10+6.742e13
total query cost
query plan tree depth
SAIO
GEQO

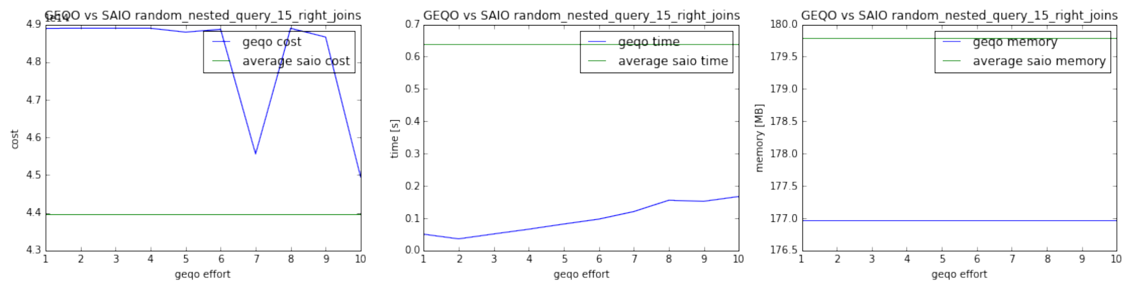

random_nested_query_15_left_joins
1e11+3.238e14
Cost []
Optimizing time [s]
Memory usage [MB]
+1.7701e2
GEQO
SAIO

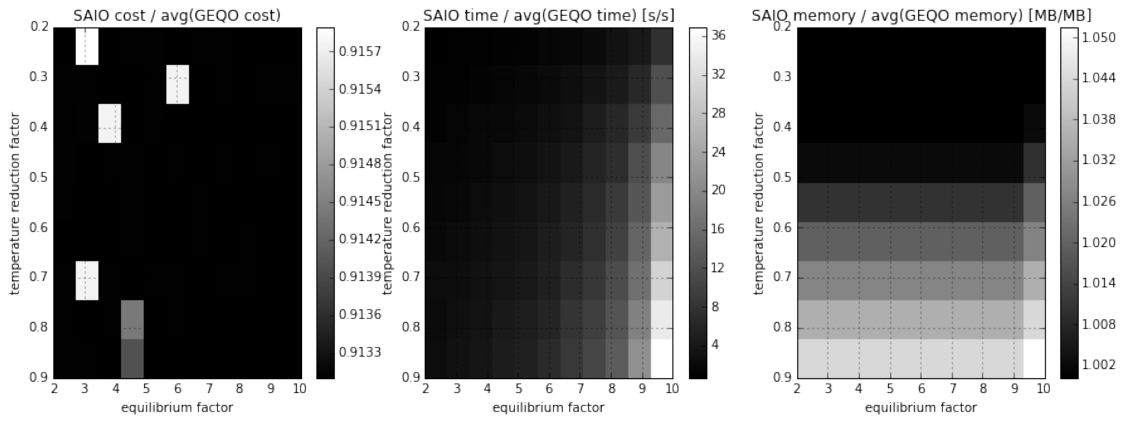

GEQO vs SAIO random_nested_query_15_left_joins
geqo cost
average saio cost
geqo time
average saio time
geqo memory
average saio memory
cost
time [s]
memory [MB]
geqo effort

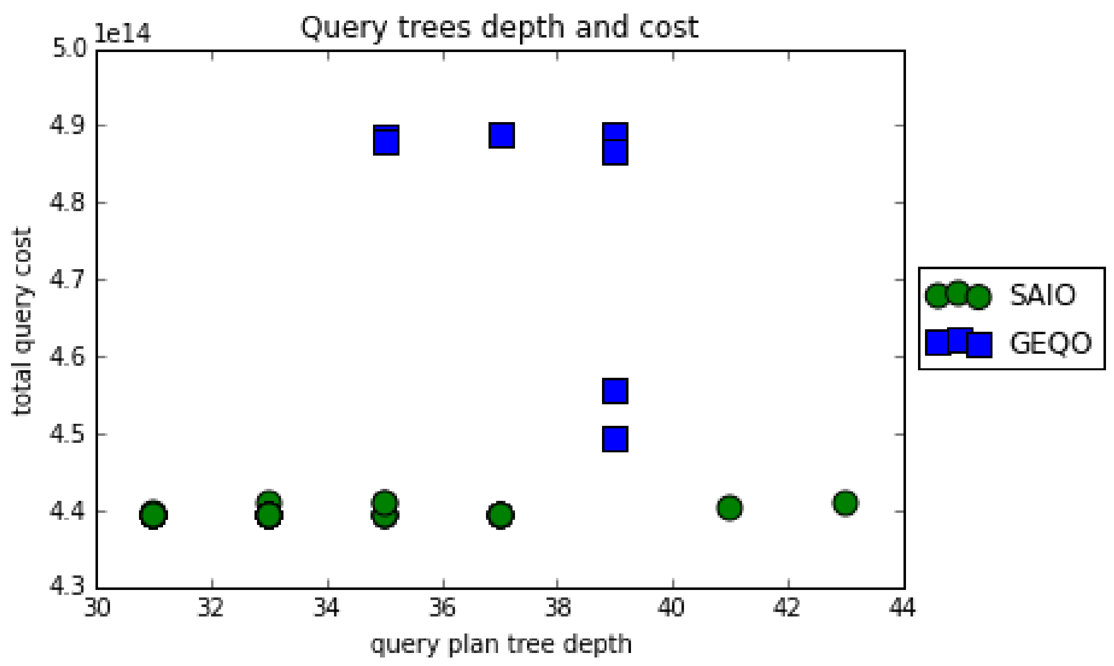

SAIO cost / avg(GEQO cost)
SAIO time / avg(GEQO time) [s/s]
SAIO memory / avg(GEQO memory) [MB/MB]
temperature reduction factor
equilibrium factor
+1.0002

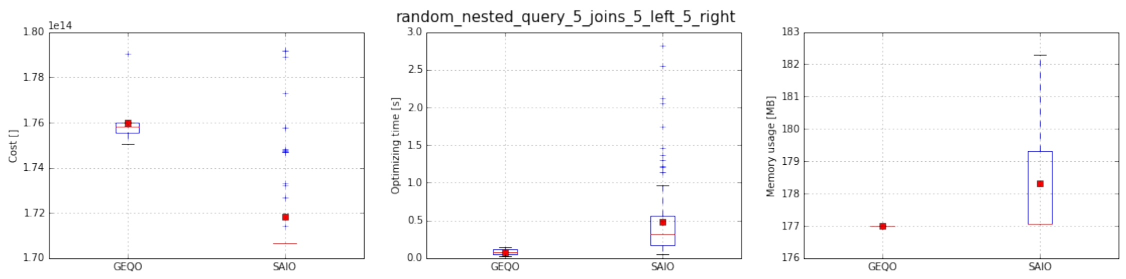

Query trees depth and cost
1e11+3.238e14
total query cost
query plan tree depth
SAIO
GEQO

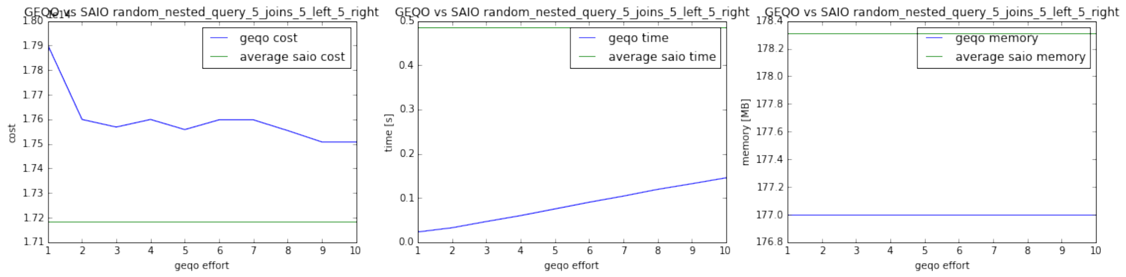

random_nested_query_15_right_joins
1e14
Cost []
Optimizing time [s]
Memory usage [MB]
GEQO
SAIO

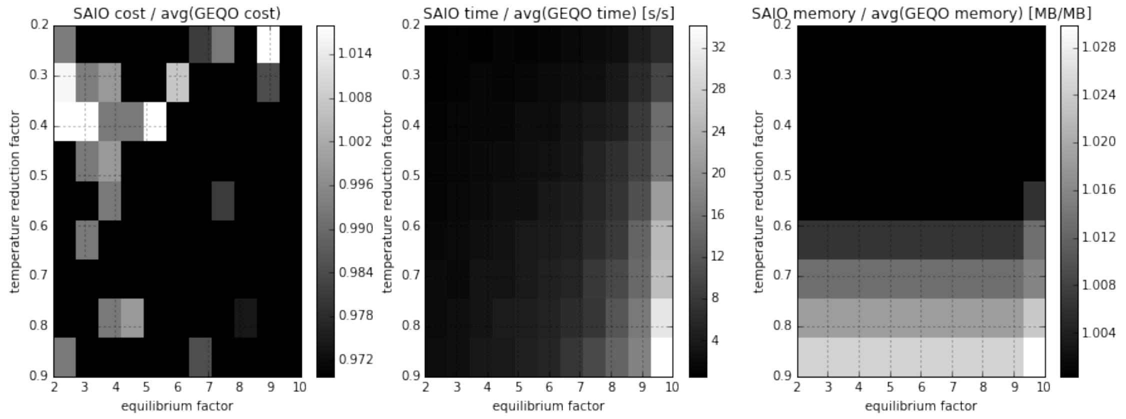

GEQO vs SAIO random_nested_query_15_right_joins
geqo cost
average saio cost
cost
geqo effort
geqo time
average saio time
time [s]
geqo memory
average saio memory
memory [MB]

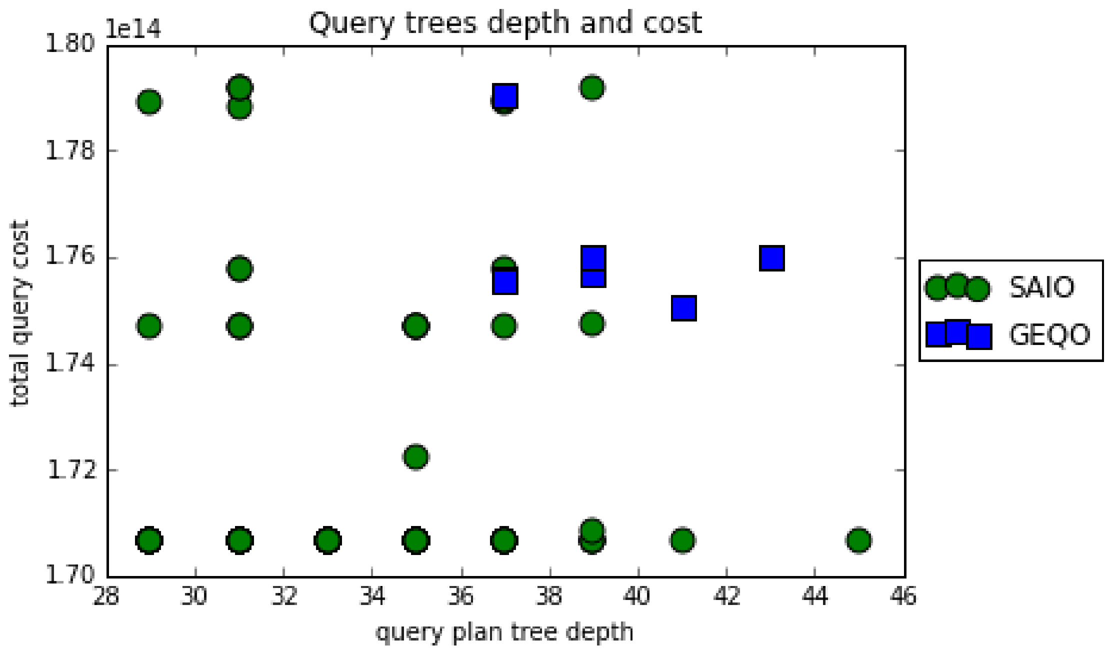

SAIO cost / avg(GEQO cost)
SAIO time / avg(GEQO time) [s/s]
SAIO memory / avg(GEQO memory) [MB/MB]
temperature reduction factor
equilibrium factor


Query trees depth and cost
1e14
total query cost
query plan tree depth
SAIO
GEQO


random_nested_query_5_joins_5_left_5_right
Cost []
1e14
Optimizing time [s]
Memory usage [MB]
GEQO
SAIO

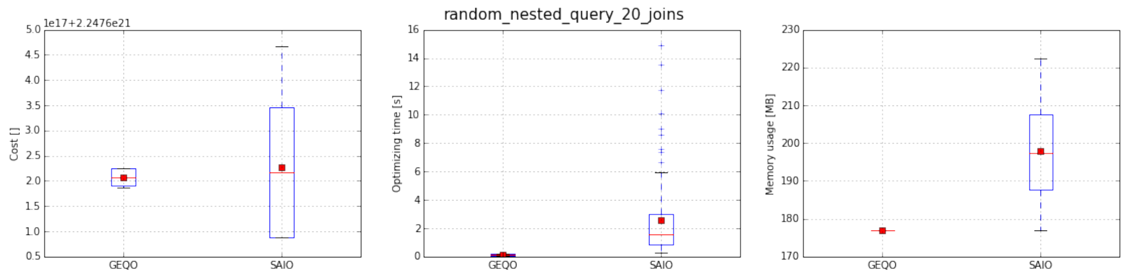

GEQO vs SAIO random_nested_query_5_joins_5_left_5_right
geqo cost
average saio cost
cost
1e14
geqo time
average saio time
time [s]
geqo memory
average saio memory
memory [MB]
geqo effort

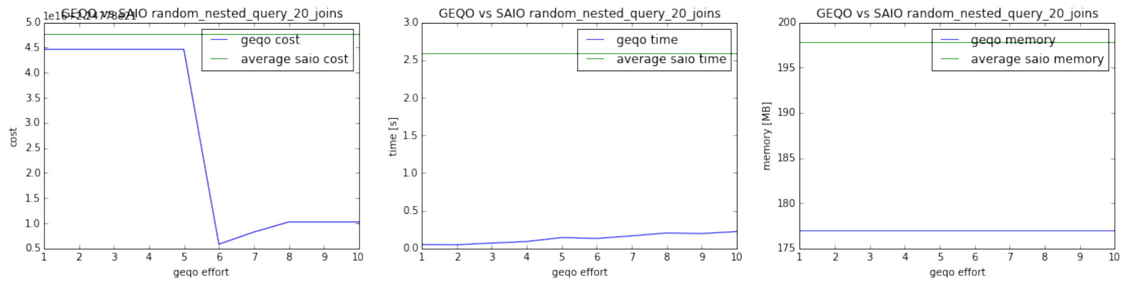

SAIO cost / avg(GEQO cost)
SAIO time / avg(GEQO time) [s/s]
SAIO memory / avg(GEQO memory) [MB/MB]
temperature reduction factor
equilibrium factor

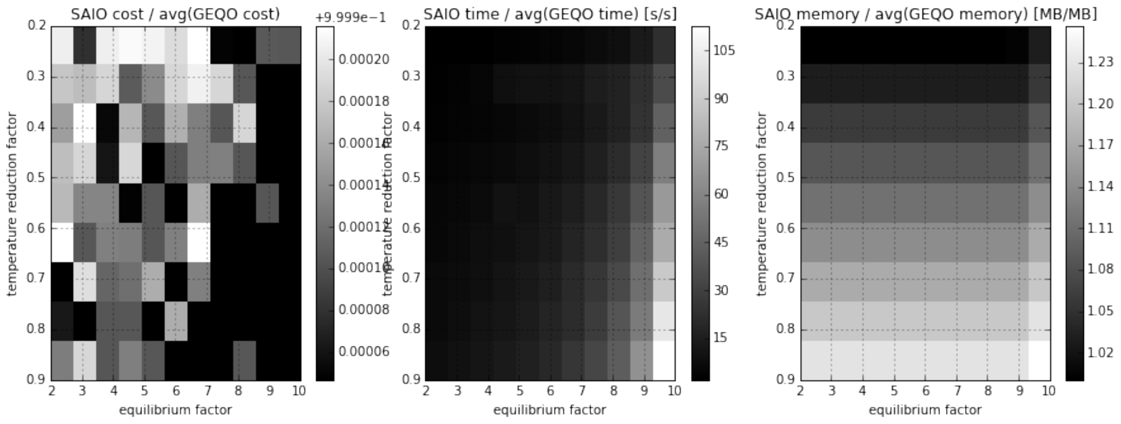


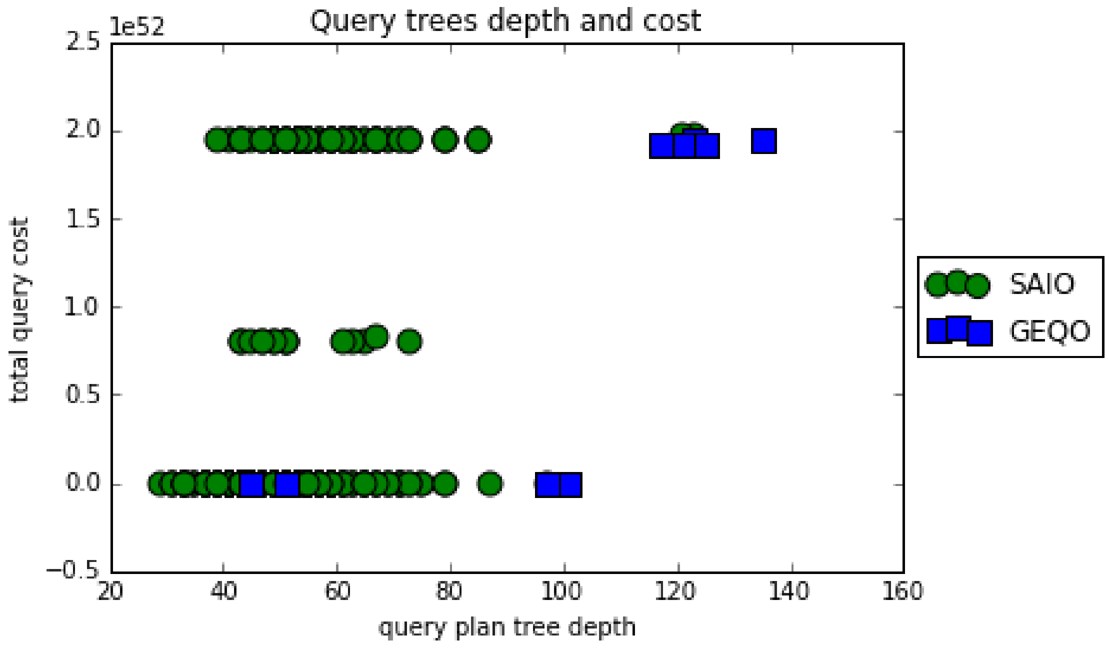
20-25 JOINS SAIO and GEQO are comparable for queries with 20-25 JOINS. GEQO wins at LEFT JOINS. SAIO wins at RIGHT JOINS.

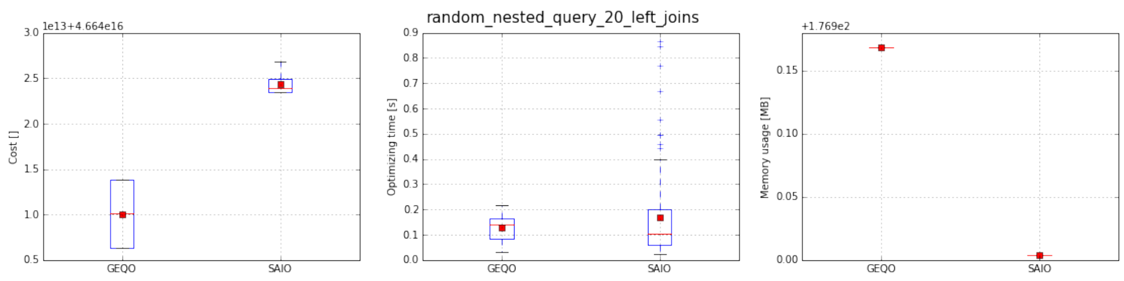
```
In [5]: display_info_for_test_case('random_nested_query_20_joins')
        display_info_for_test_case('random_nested_query_20_left_joins')
        display_info_for_test_case('random_nested_query_20_right_joins')
```

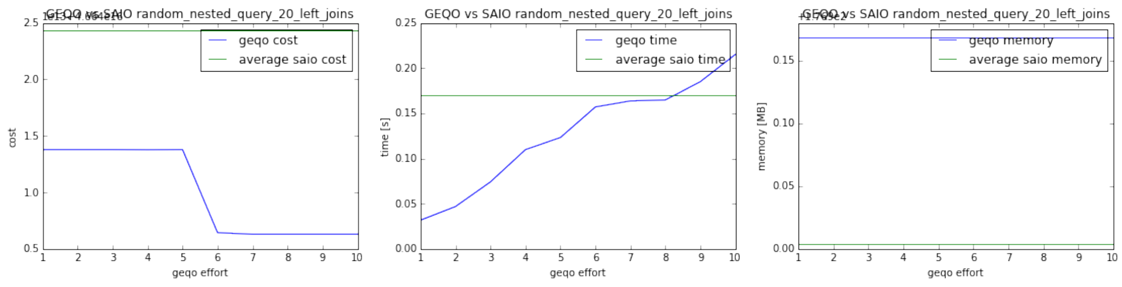


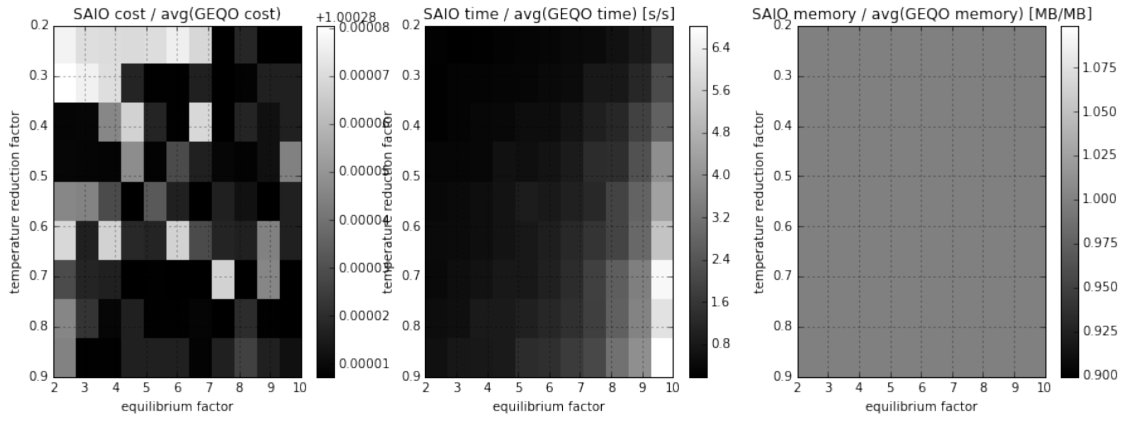


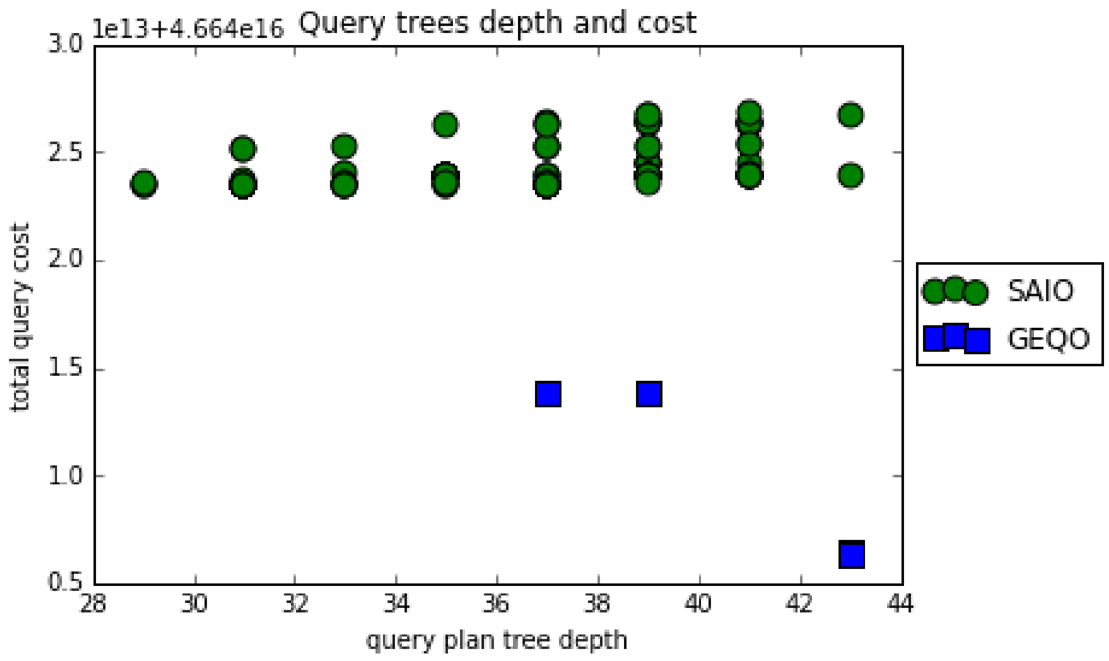

SAIO cost / avg(GEQO cost)
SAIO time / avg(GEQO time) [s/s]
SAIO memory / avg(GEQO memory) [MB/MB]
temperature reduction factor
equilibrium factor
+9.999e−1

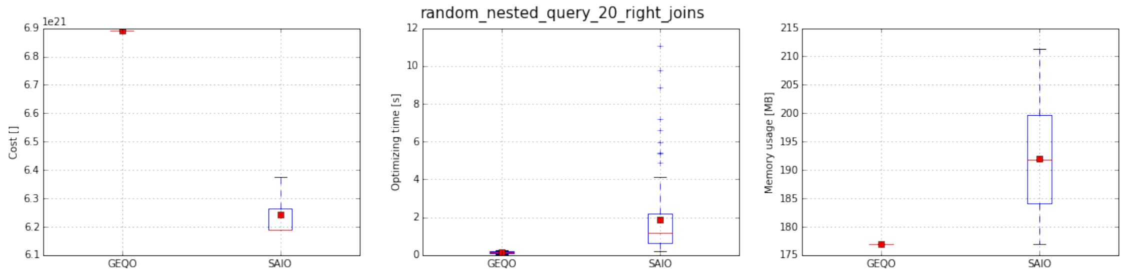

Query trees depth and cost
1e52
total query cost
query plan tree depth
SAIO
GEQO

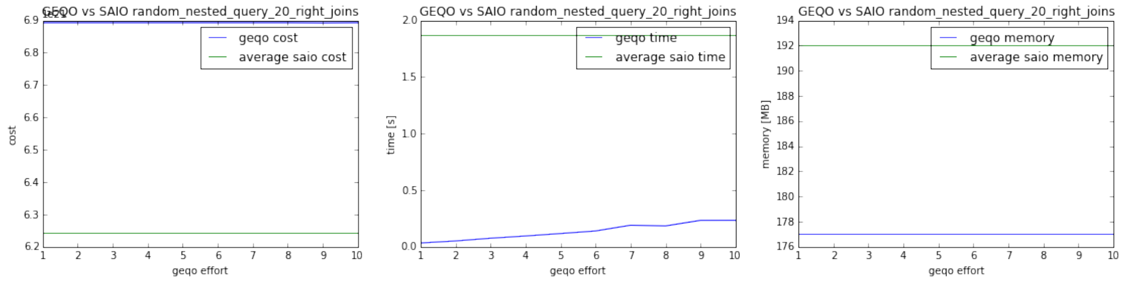

random_nested_query_20_left_joins
1e13+4.664e16
Cost []
Optimizing time [s]
+1.769e2
Memory usage [MB]
GEQO
SAIO

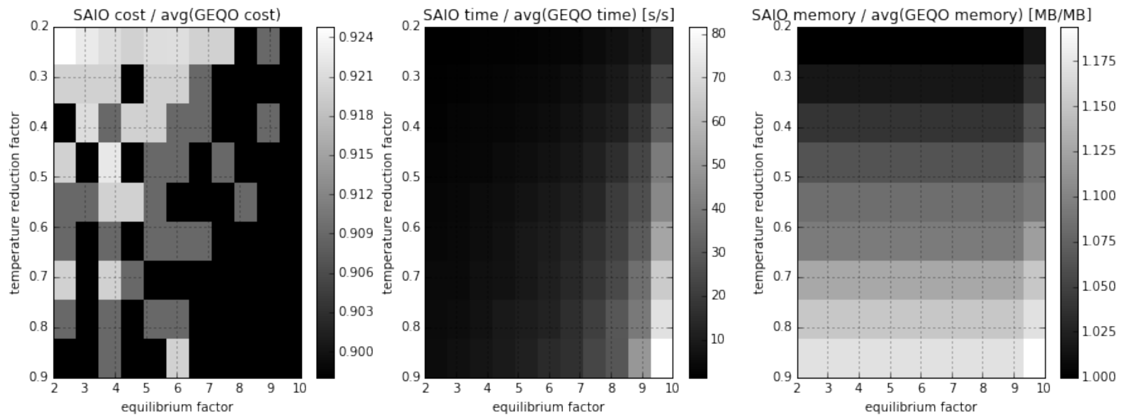

GEQO vs SAIO random_nested_query_20_left_joins
geqo cost
average saio cost
cost
geqo effort
GEQO vs SAIO random_nested_query_20_left_joins
geqo time
average saio time
time [s]
geqo effort
GEQO vs SAIO random_nested_query_20_left_joins
geqo memory
average saio memory
memory [MB]
geqo effort

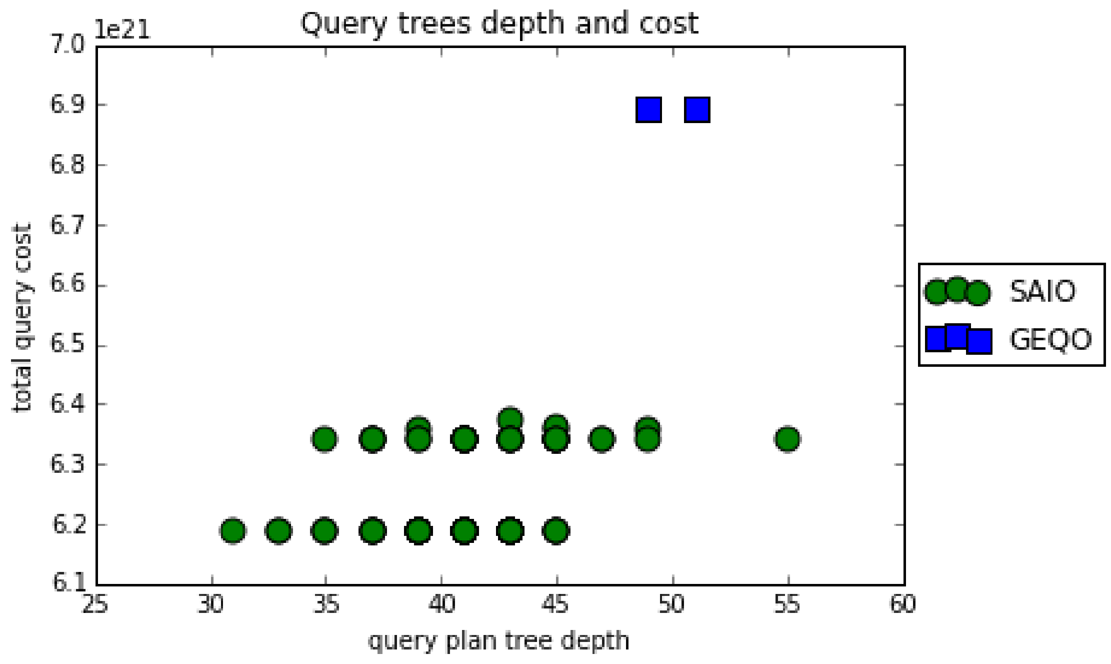

SAIO cost / avg(GEQO cost)
+1.00028
temperature reduction factor
equilibrium factor
SAIO time / avg(GEQO time) [s/s]
temperature reduction factor
equilibrium factor
SAIO memory / avg(GEQO memory) [MB/MB]
temperature reduction factor
equilibrium factor


Query trees depth and cost
1e13+4.664e16
total query cost
query plan tree depth
SAIO
GEQO


random_nested_query_20_right_joins
1e21
Cost []
GEQO
SAIO
Optimizing time [s]
Memory usage [MB]

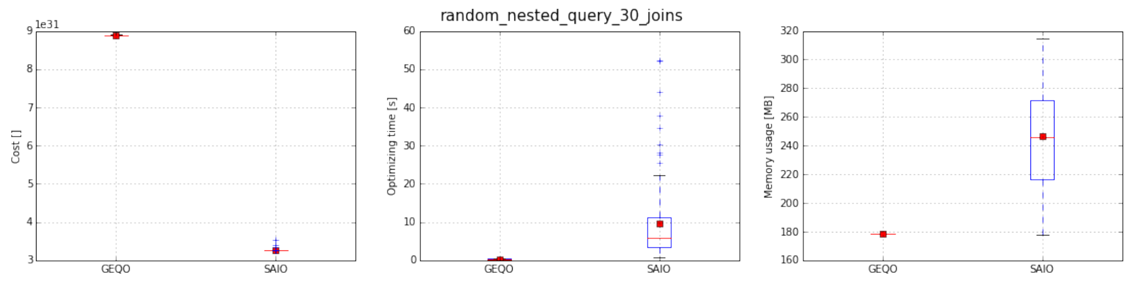

GEQO vs SAIO random_nested_query_20_right_joins
1e21
geqo cost
average saio cost
cost
geqo effort
geqo time
average saio time
time [s]
geqo memory
average saio memory
memory [MB]

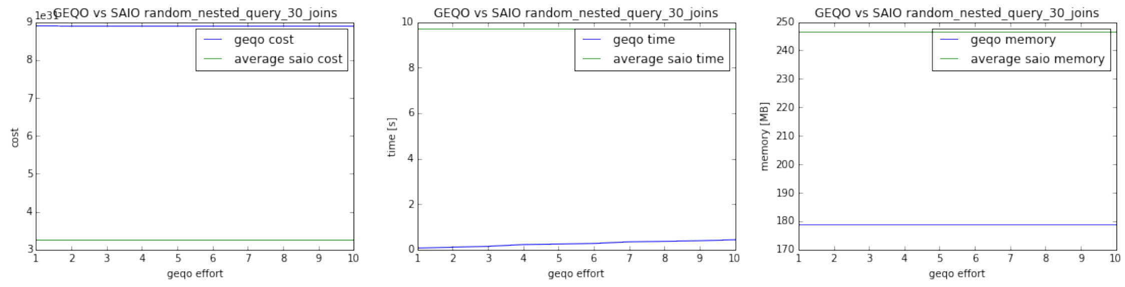

SAIO cost / avg(GEQO cost)
SAIO time / avg(GEQO time) [s/s]
SAIO memory / avg(GEQO memory) [MB/MB]
temperature reduction factor
equilibrium factor

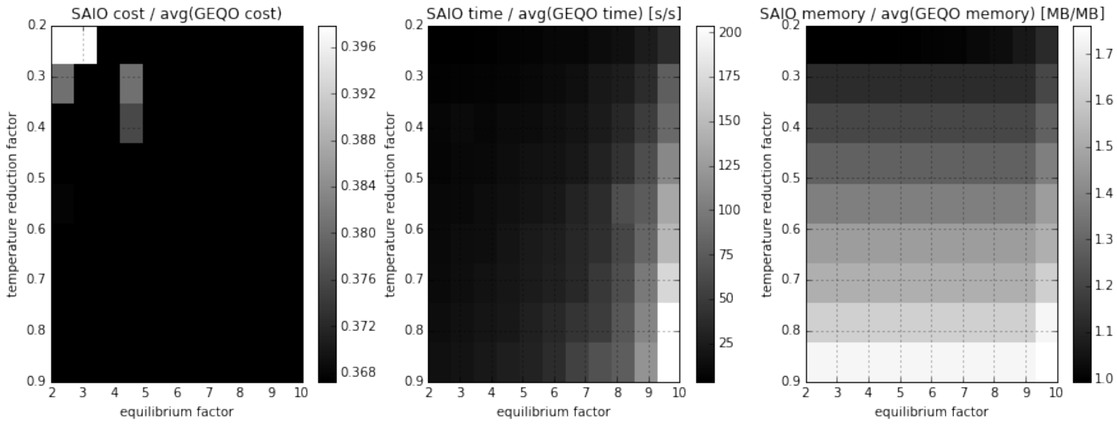


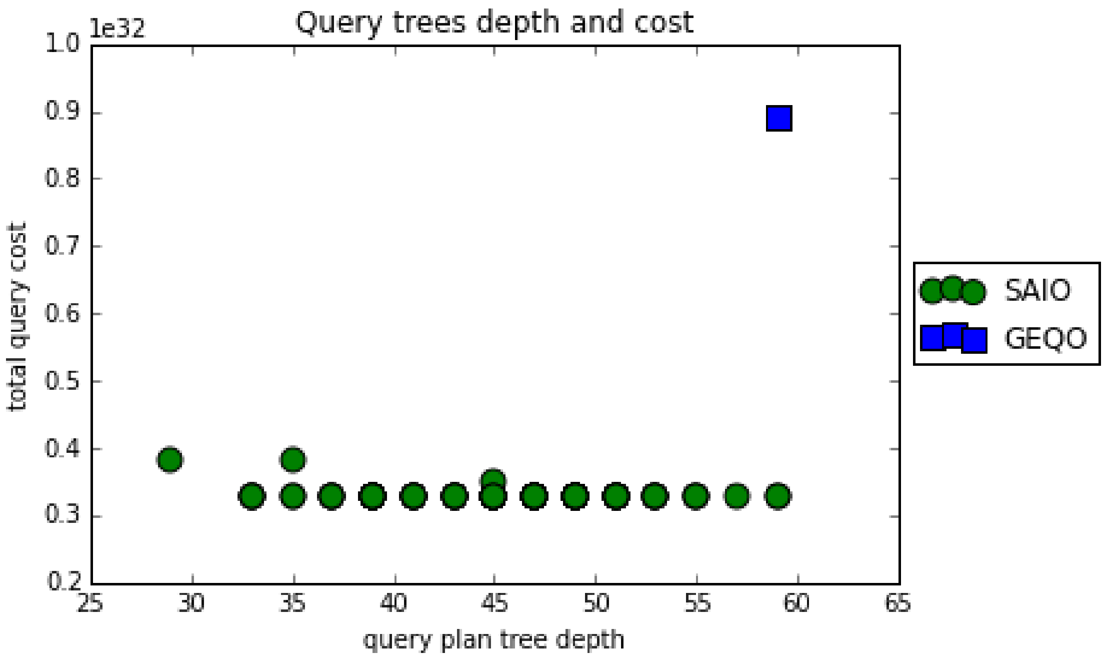
30-35 JOINS Here SAIO finds better results for queries with 30-35 JOINS, RIGHT JOINS and mixed. Solutions for queries with LEFT JOINS are comparable between GEQO and SAIO.

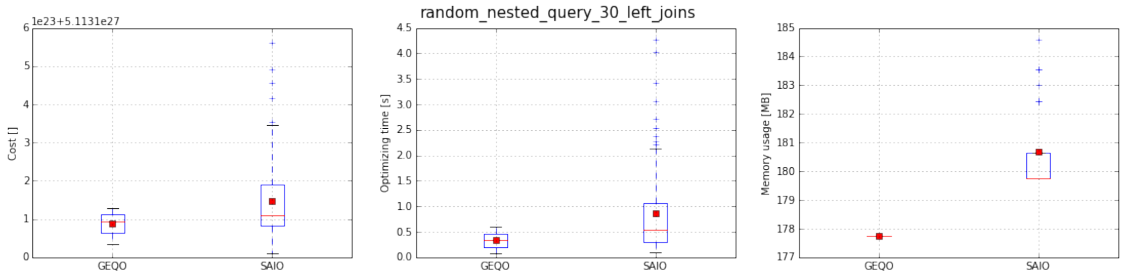
```
In [12]: display_info_for_test_case('random_nested_query_30_joins')
         display_info_for_test_case('random_nested_query_30_left_joins')
         display_info_for_test_case('random_nested_query_30_right_joins')
         display_info_for_test_case('random_nested_query_10_joins_10_left_10_right')
```

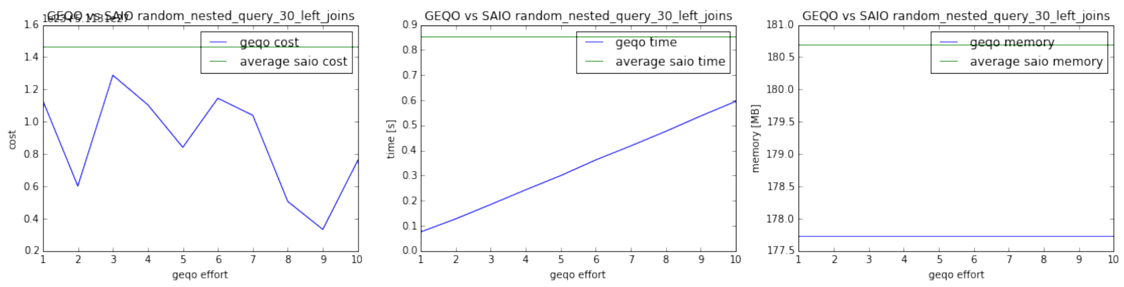


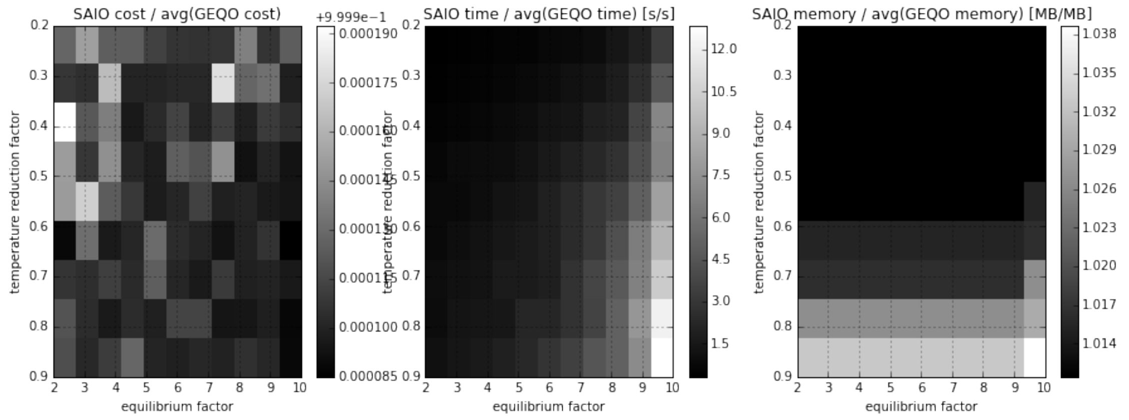


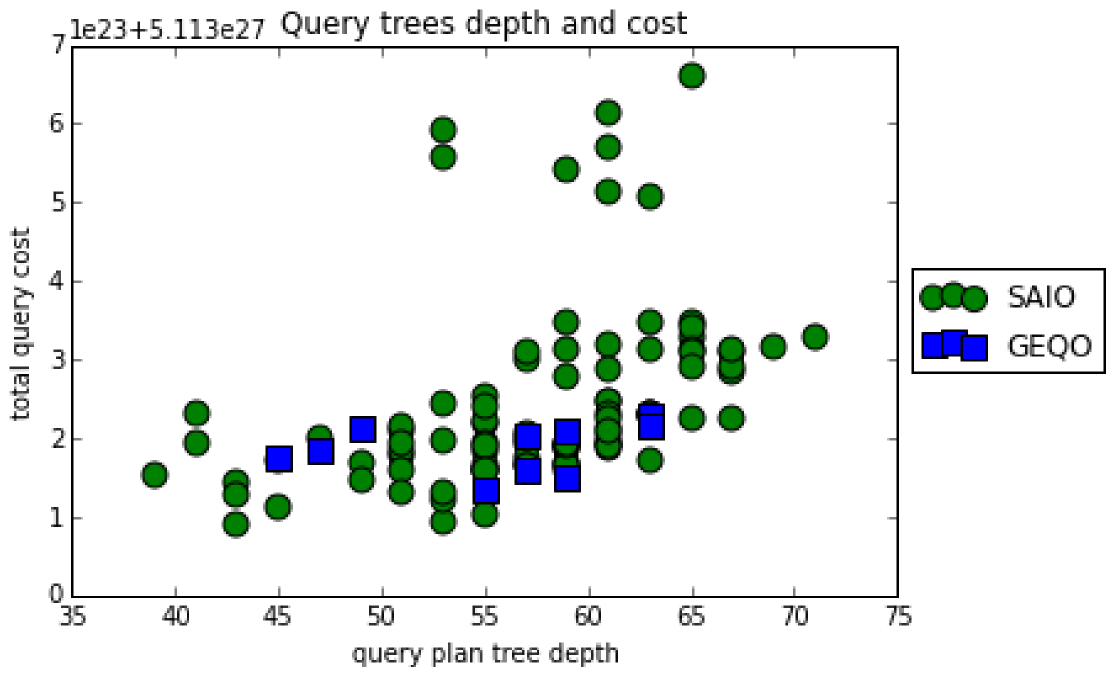

SAIO cost / avg(GEQO cost)
SAIO time / avg(GEQO time) [s/s]
SAIO memory / avg(GEQO memory) [MB/MB]
temperature reduction factor
equilibrium factor

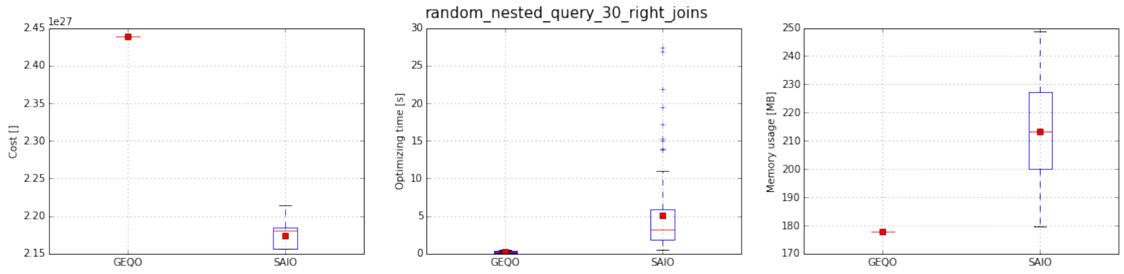

Query trees depth and cost
1e32
total query cost
query plan tree depth
SAIO
GEQO

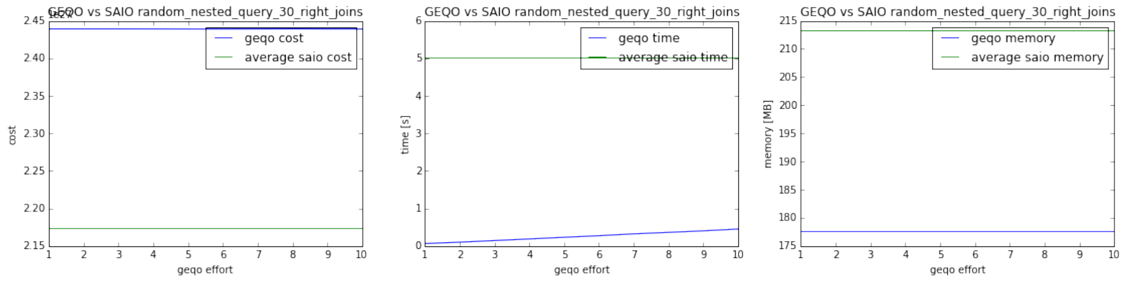

random_nested_query_30_left_joins
1e23+5.1131e27
Cost []
Optimizing time [s]
Memory usage [MB]
GEQO
SAIO

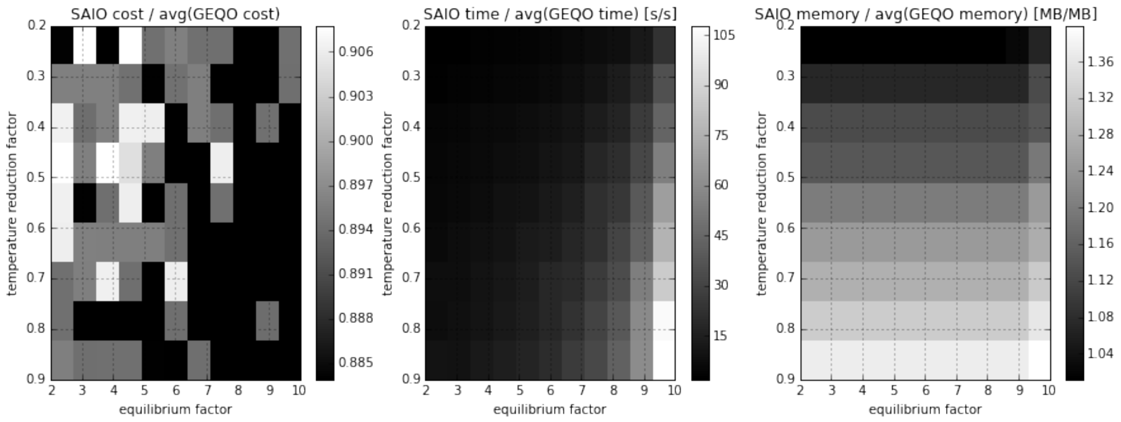

GEQO vs SAIO random_nested_query_30_left_joins
1e23+5.113e27
cost
geqo effort
geqo cost
average saio cost
GEQO vs SAIO random_nested_query_30_left_joins
time [s]
geqo effort
geqo time
average saio time
GEQO vs SAIO random_nested_query_30_left_joins
memory [MB]
geqo effort
geqo memory
average saio memory

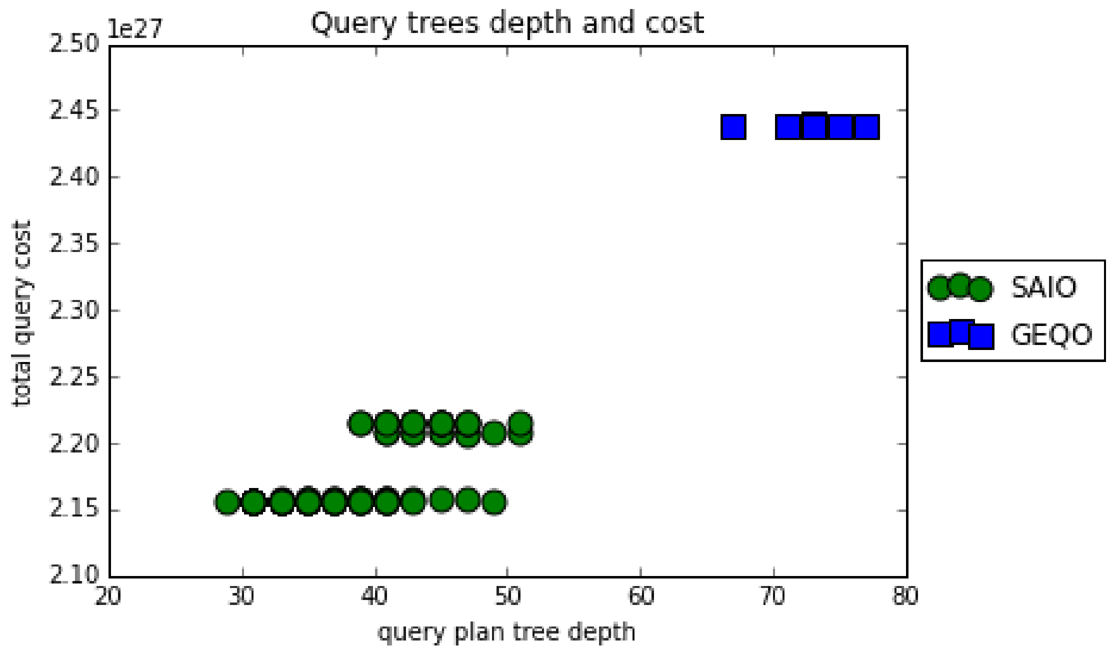

SAIO cost / avg(GEQO cost)
+9.999e−1
temperature reduction factor
equilibrium factor
SAIO time / avg(GEQO time) [s/s]
temperature reduction factor
equilibrium factor
SAIO memory / avg(GEQO memory) [MB/MB]
temperature reduction factor
equilibrium factor

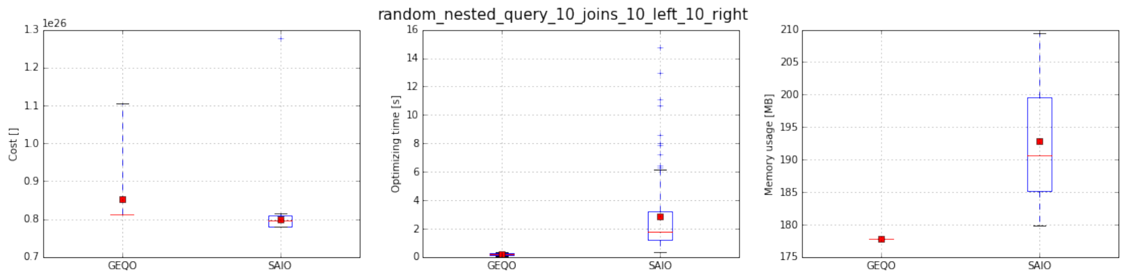

Query trees depth and cost
1e23+5.113e27
total query cost
query plan tree depth
SAIO
GEQO

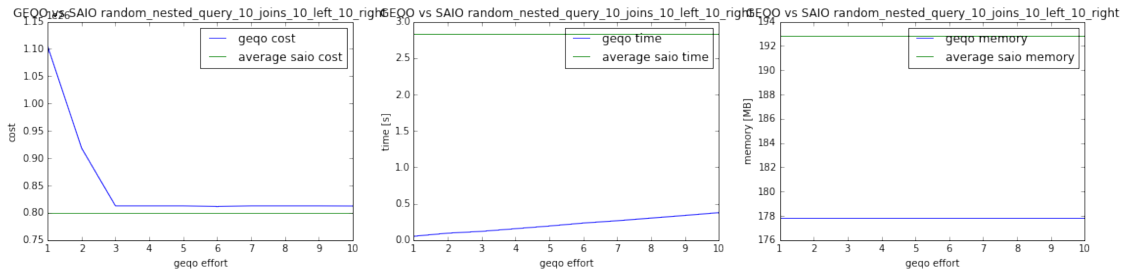

random_nested_query_30_right_joins
Cost []
1e27
GEQO
SAIO
Optimizing time [s]
Memory usage [MB]

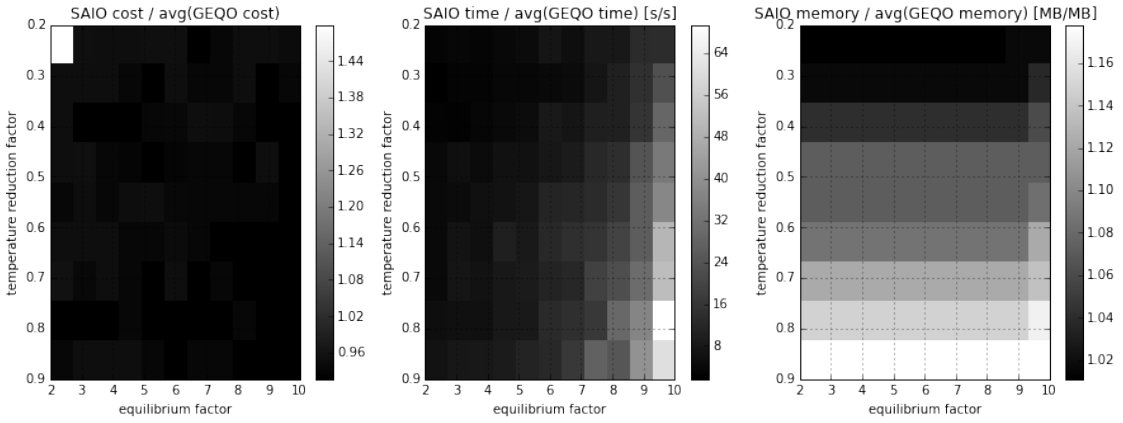

GEQO vs SAIO random_nested_query_30_right_joins
geqo cost
average saio cost
cost
geqo time
average saio time
time [s]
geqo memory
average saio memory
memory [MB]
geqo effort

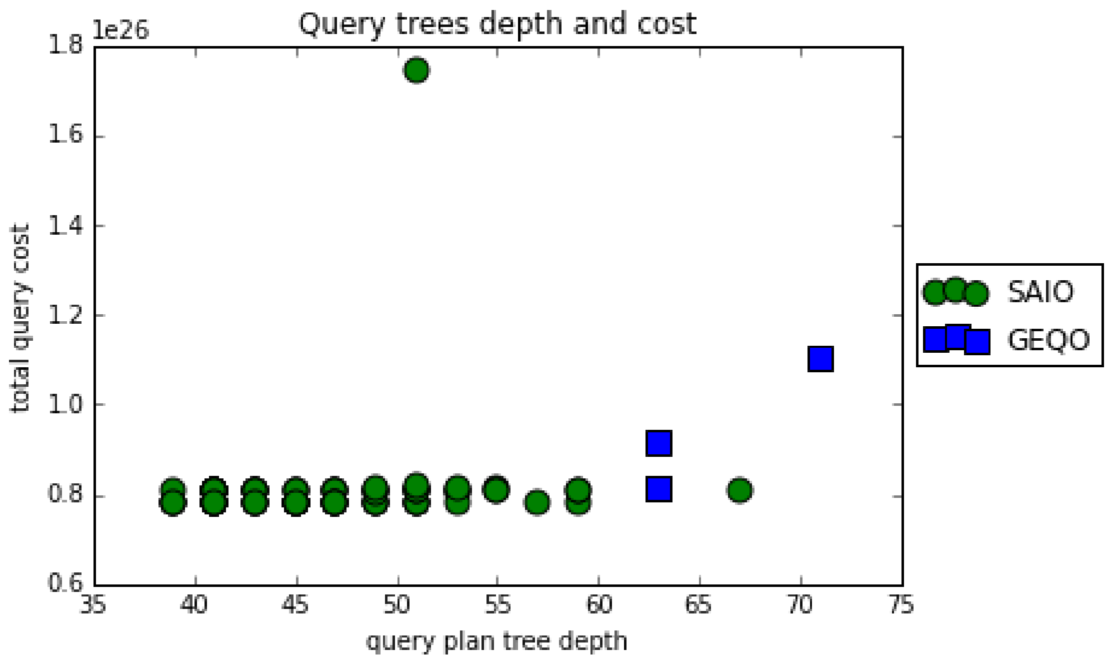

SAIO cost / avg(GEQO cost)
SAIO time / avg(GEQO time) [s/s]
SAIO memory / avg(GEQO memory) [MB/MB]
temperature reduction factor
equilibrium factor


Query trees depth and cost
1e27
total query cost
query plan tree depth
SAIO
GEQO
2.50
2.45
2.40
2.35
2.30
2.25
2.20
2.15
2.10
20
30
40
50
60
70
80


random_nested_query_10_joins_10_left_10_right
1e26
Cost []
Optimizing time [s]
Memory usage [MB]
GEQO
SAIO

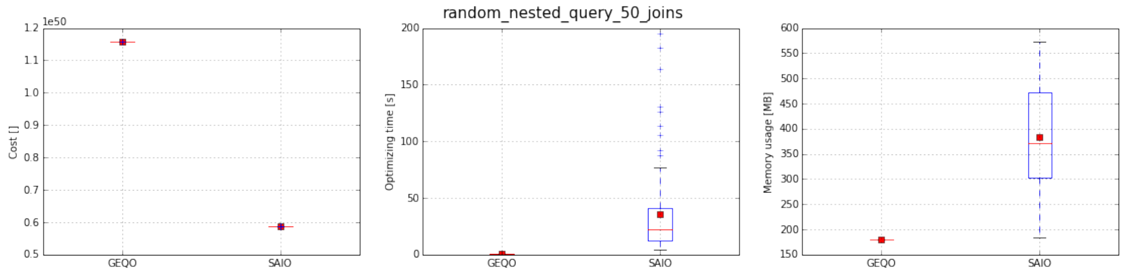

GEQO vs SAIO random_nested_query_10_joins_10_left_10_right
1e26
cost
time [s]
memory [MB]
geqo effort
geqo cost
average saio cost
geqo time
average saio time
geqo memory
average saio memory

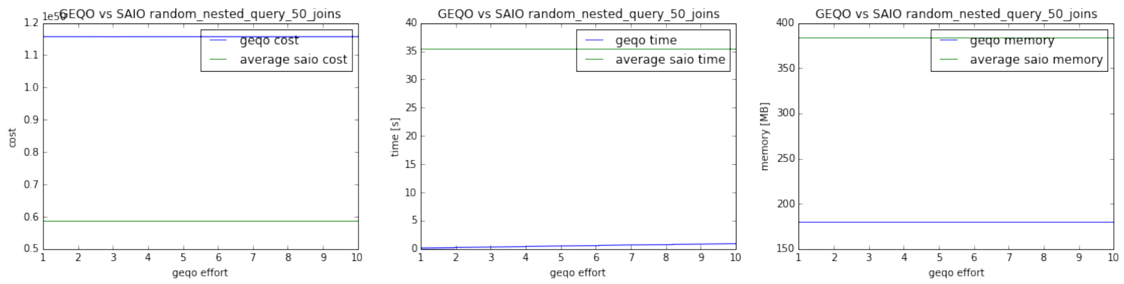


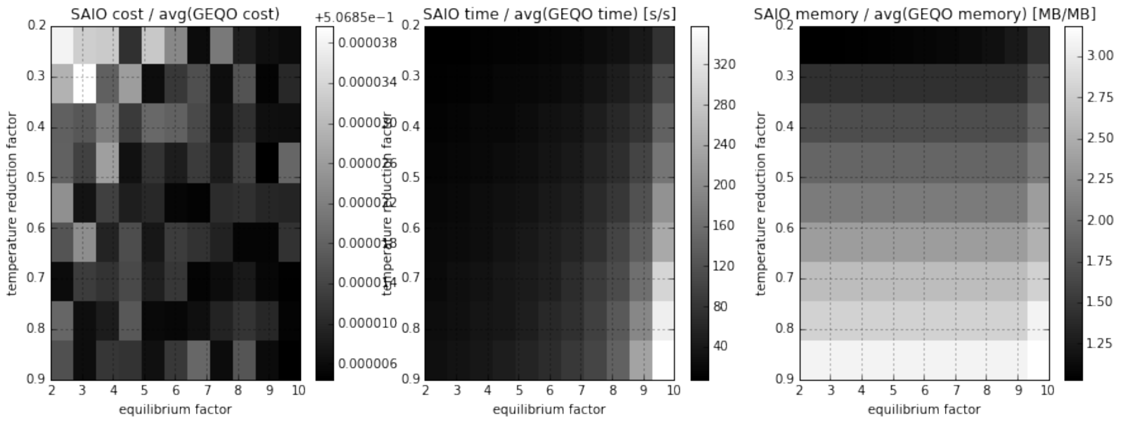


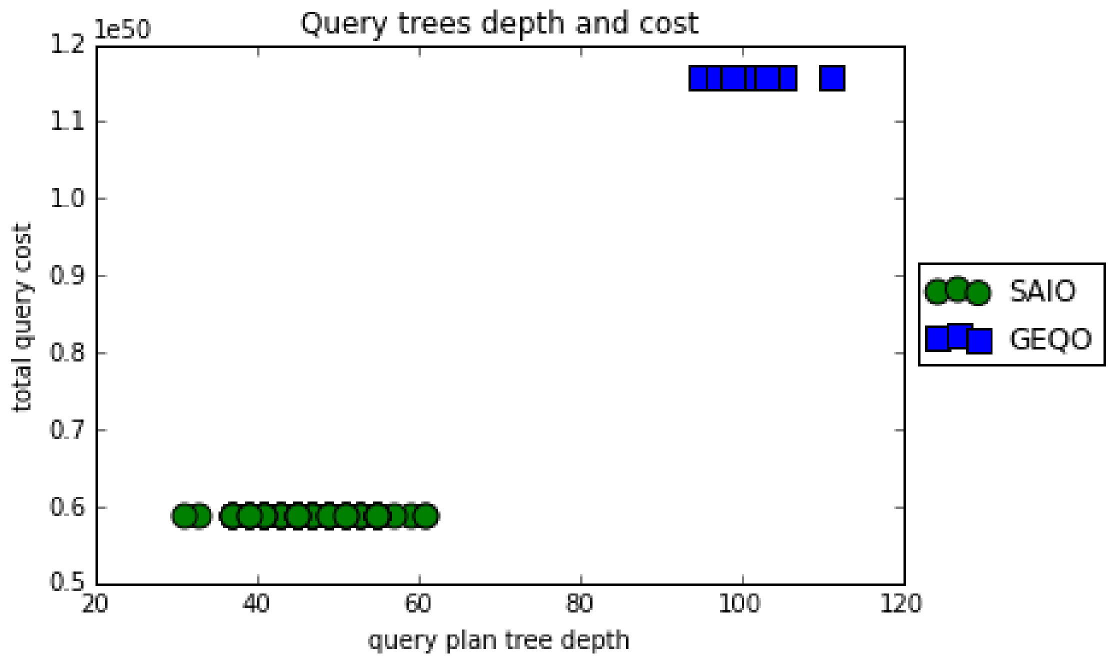
50-60 joins Looking at the costs, these queries would take almost forever to complete. But still it is visible that GEQO wins for queries that contain LEFT JOINS while SAIO wins at regular JOINS, RIGHT JOINS and mixed queries.

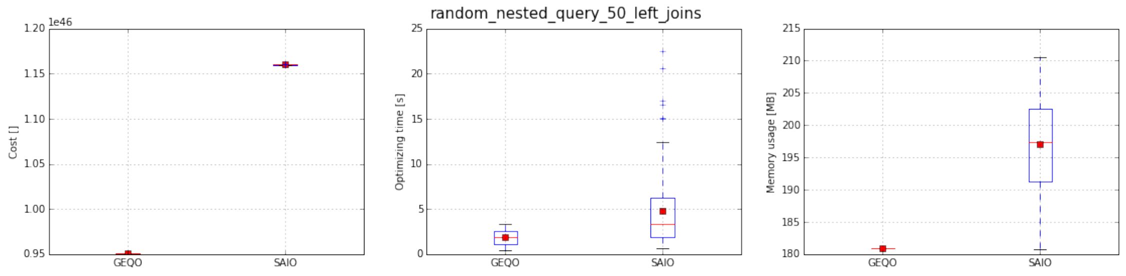
```
In [24]: display_info_for_test_case('random_nested_query_50_joins')
         display_info_for_test_case('random_nested_query_50_left_joins')
         display_info_for_test_case('random_nested_query_50_right_joins')
         display_info_for_test_case('random_nested_query_20_joins_15_left_15_right_to_atri2')
```

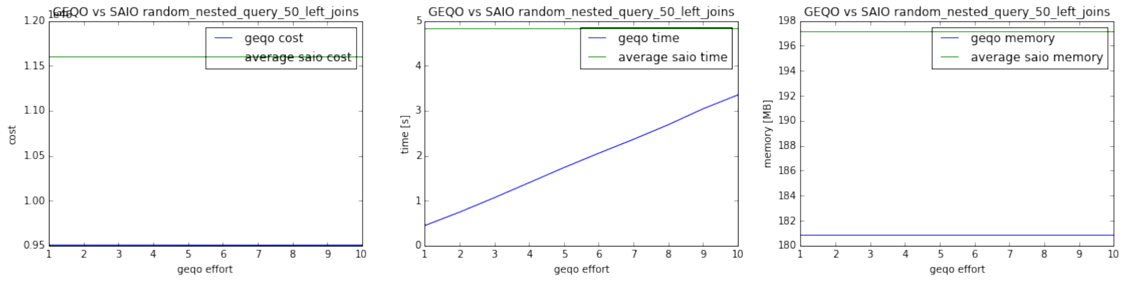

random_nested_query_50_joins
1e50
Cost []
Optimizing time [s]
Memory usage [MB]
GEQO
SAIO

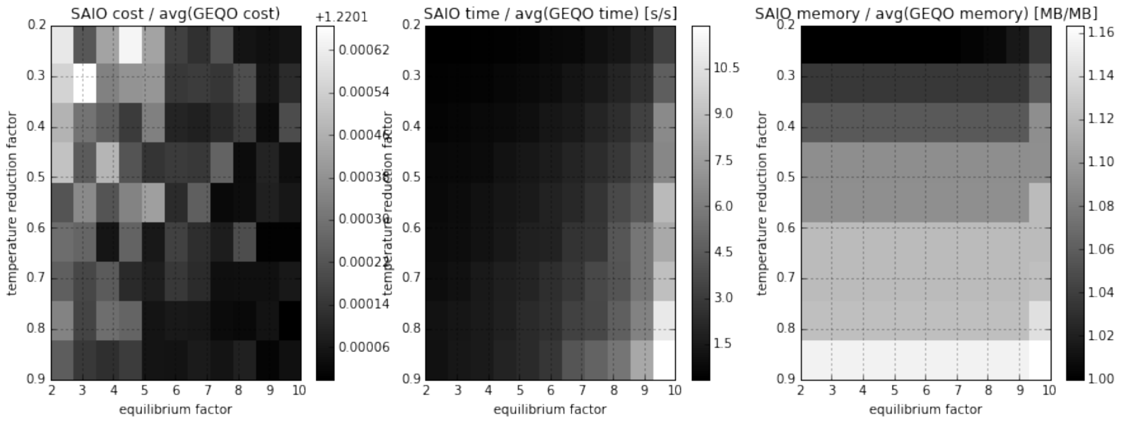

GEQO vs SAIO random_nested_query_50_joins
1e50
geqo cost
average saio cost
geqo time
average saio time
geqo memory
average saio memory
cost
time [s]
memory [MB]
geqo effort

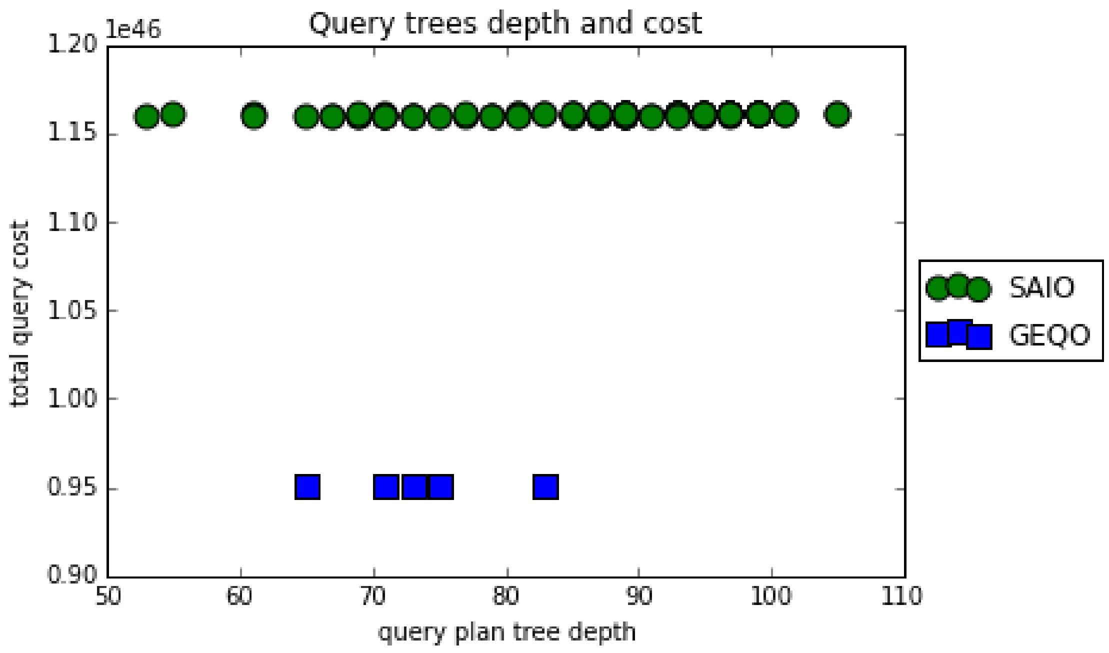

SAIO cost / avg(GEQO cost)
+5.0685e−1
SAIO time / avg(GEQO time) [s/s]
SAIO memory / avg(GEQO memory) [MB/MB]
temperature reduction factor
equilibrium factor

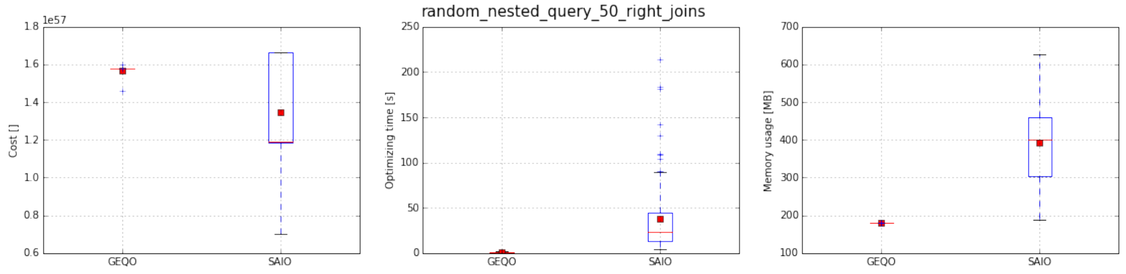

Query trees depth and cost
1e50
total query cost
query plan tree depth
SAIO
GEQO

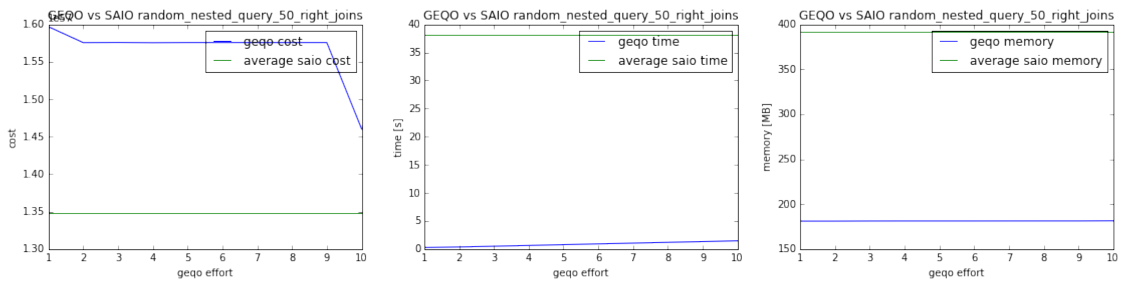

random_nested_query_50_left_joins
1e46
Cost []
Optimizing time [s]
Memory usage [MB]
GEQO
SAIO

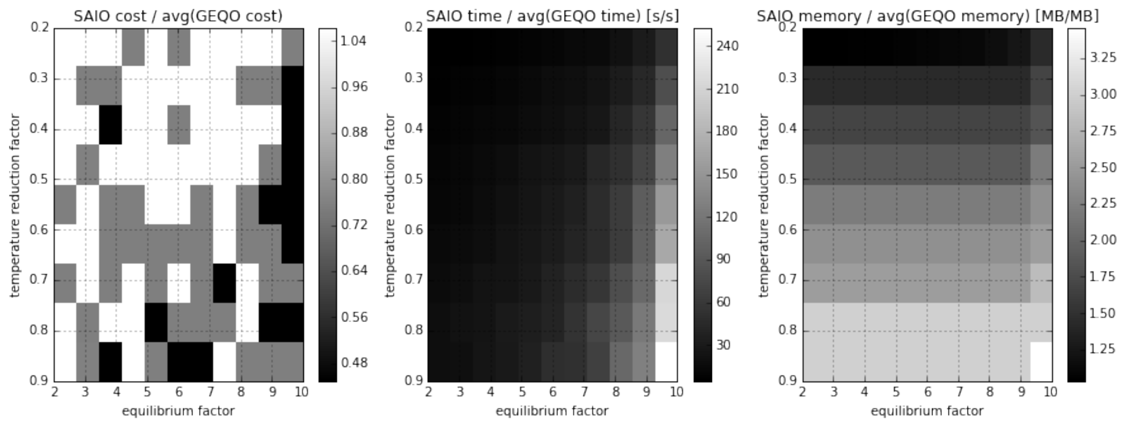

GEQO vs SAIO random_nested_query_50_left_joins
1e46
cost
geqo cost
average saio cost
time [s]
geqo time
average saio time
memory [MB]
geqo memory
average saio memory
geqo effort

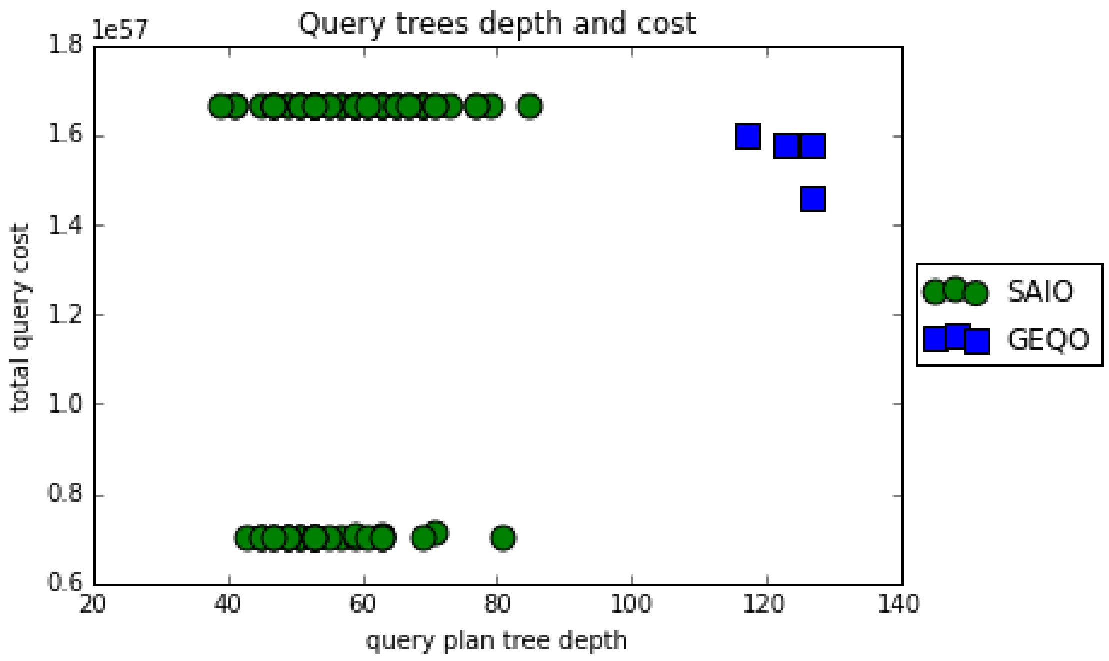

SAIO cost / avg(GEQO cost)
SAIO time / avg(GEQO time) [s/s]
SAIO memory / avg(GEQO memory) [MB/MB]
temperature reduction factor
equilibrium factor
+1.2201

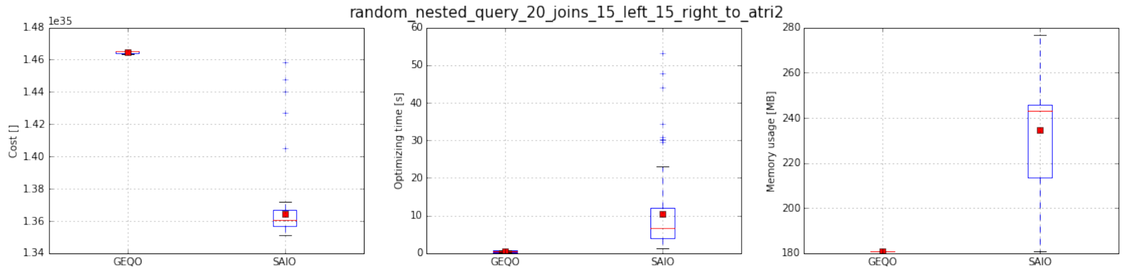

Query trees depth and cost
1e46
total query cost
query plan tree depth
SAIO
GEQO

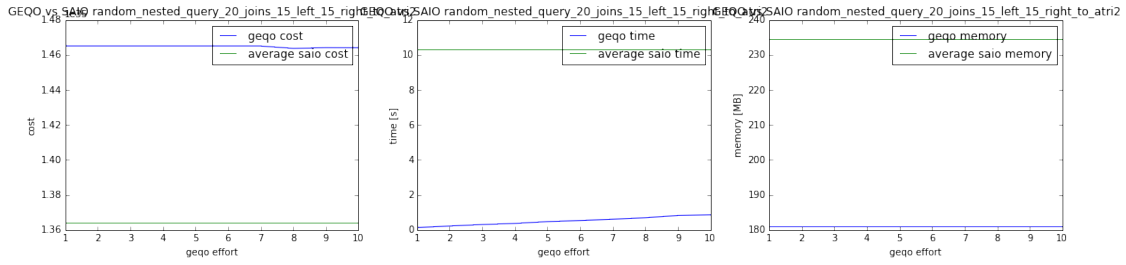

random_nested_query_50_right_joins
1e57
Cost []
Optimizing time [s]
Memory usage [MB]
GEQO
SAIO

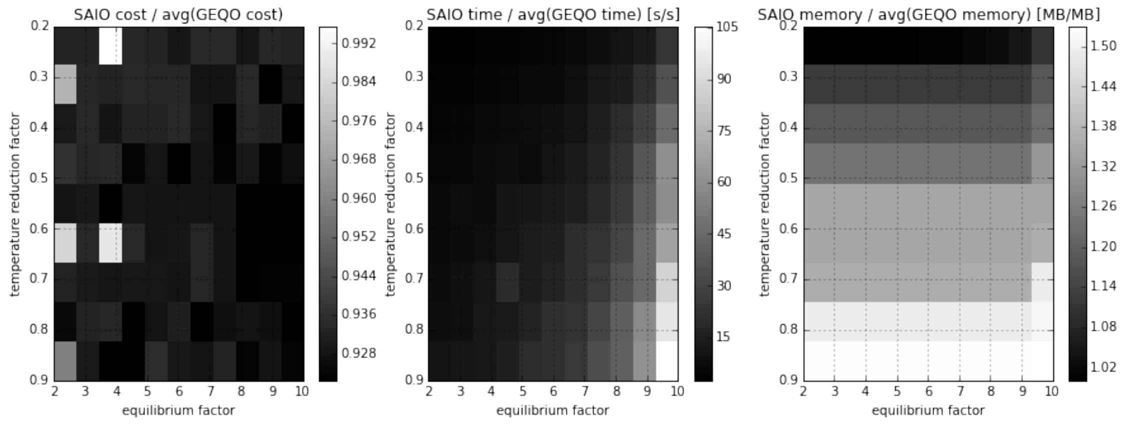

GEQO vs SAIO random_nested_query_50_right_joins
1e57
cost
geqo effort
geqo cost
average saio cost
GEQO vs SAIO random_nested_query_50_right_joins
time [s]
geqo effort
geqo time
average saio time
GEQO vs SAIO random_nested_query_50_right_joins
memory [MB]
geqo effort
geqo memory
average saio memory

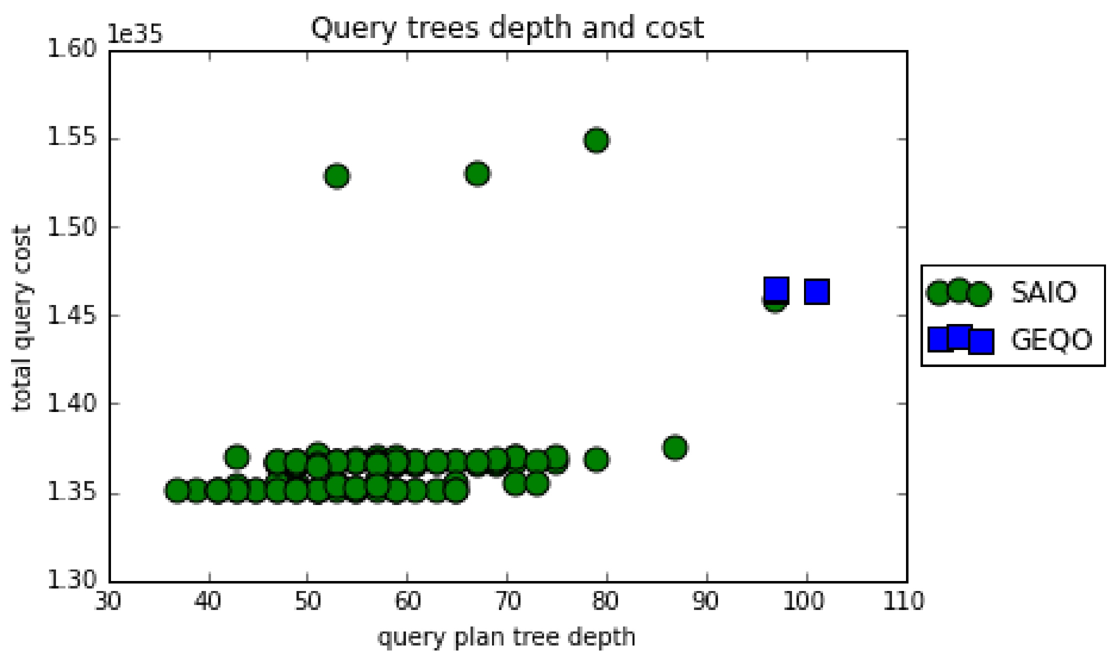

SAIO cost / avg(GEQO cost)
temperature reduction factor
equilibrium factor
SAIO time / avg(GEQO time) [s/s]
temperature reduction factor
equilibrium factor
SAIO memory / avg(GEQO memory) [MB/MB]
temperature reduction factor
equilibrium factor


Query trees depth and cost
1e57
total query cost
query plan tree depth
SAIO
GEQO


random_nested_query_20_joins_15_left_15_right_to_atri2
1e35
Cost []
GEQO
SAIO
Optimizing time [s]
Memory usage [MB]

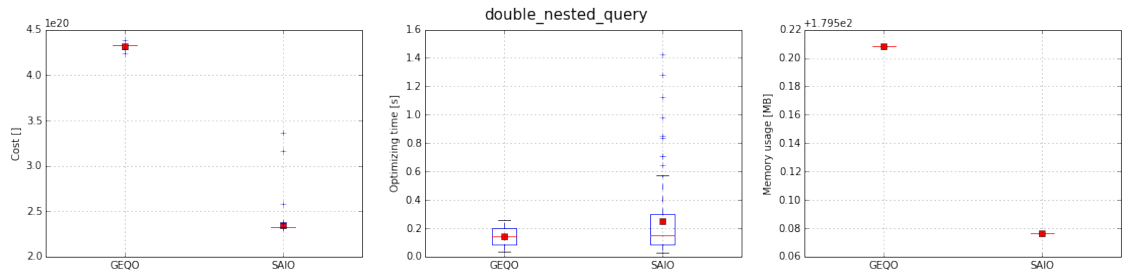

GEQO vs SAIO random_nested_query_20_joins_15_left_15_right_to_atri2
geqo cost
average saio cost
cost
geqo effort
geqo time
average saio time
time [s]
geqo memory
average saio memory
memory [MB]

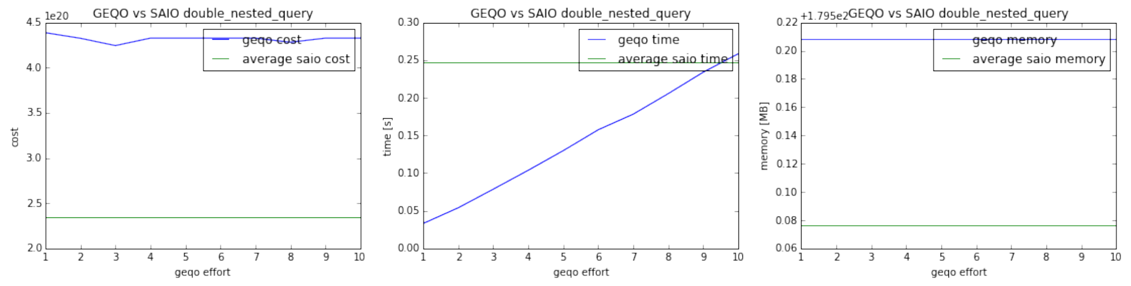

SAIO cost / avg(GEQO cost)
SAIO time / avg(GEQO time) [s/s]
SAIO memory / avg(GEQO memory) [MB/MB]
temperature reduction factor
equilibrium factor

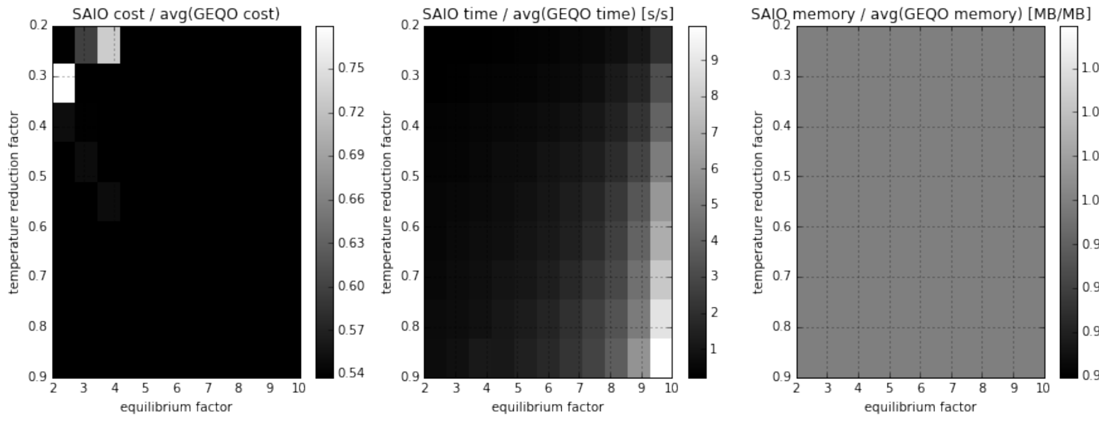


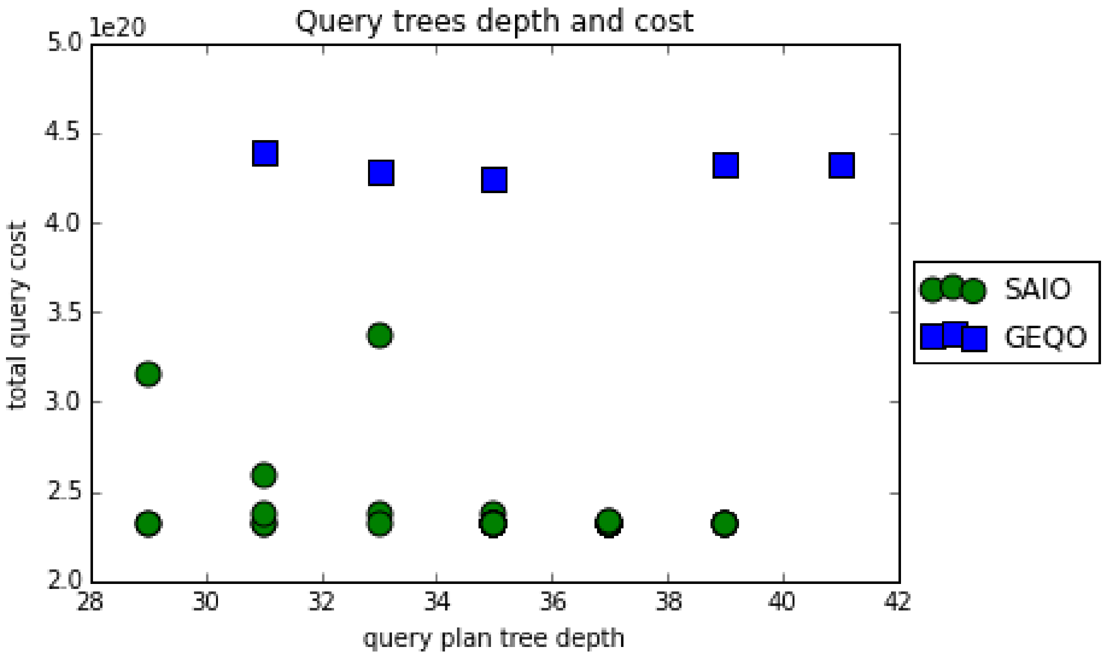
Two level nested query For this query that has subqueries within subqueries and 4 joins, 3 left joins, 15 right joins in total, it is visible that SAIO easily beats GEQO finding a query with lower cost and using less memory. For the correct set of SAIO parameters, the computing time of SAIO may be also shorter than that of GEQO.

```
In [31]: display_info_for_test_case('double_nested_query')
```









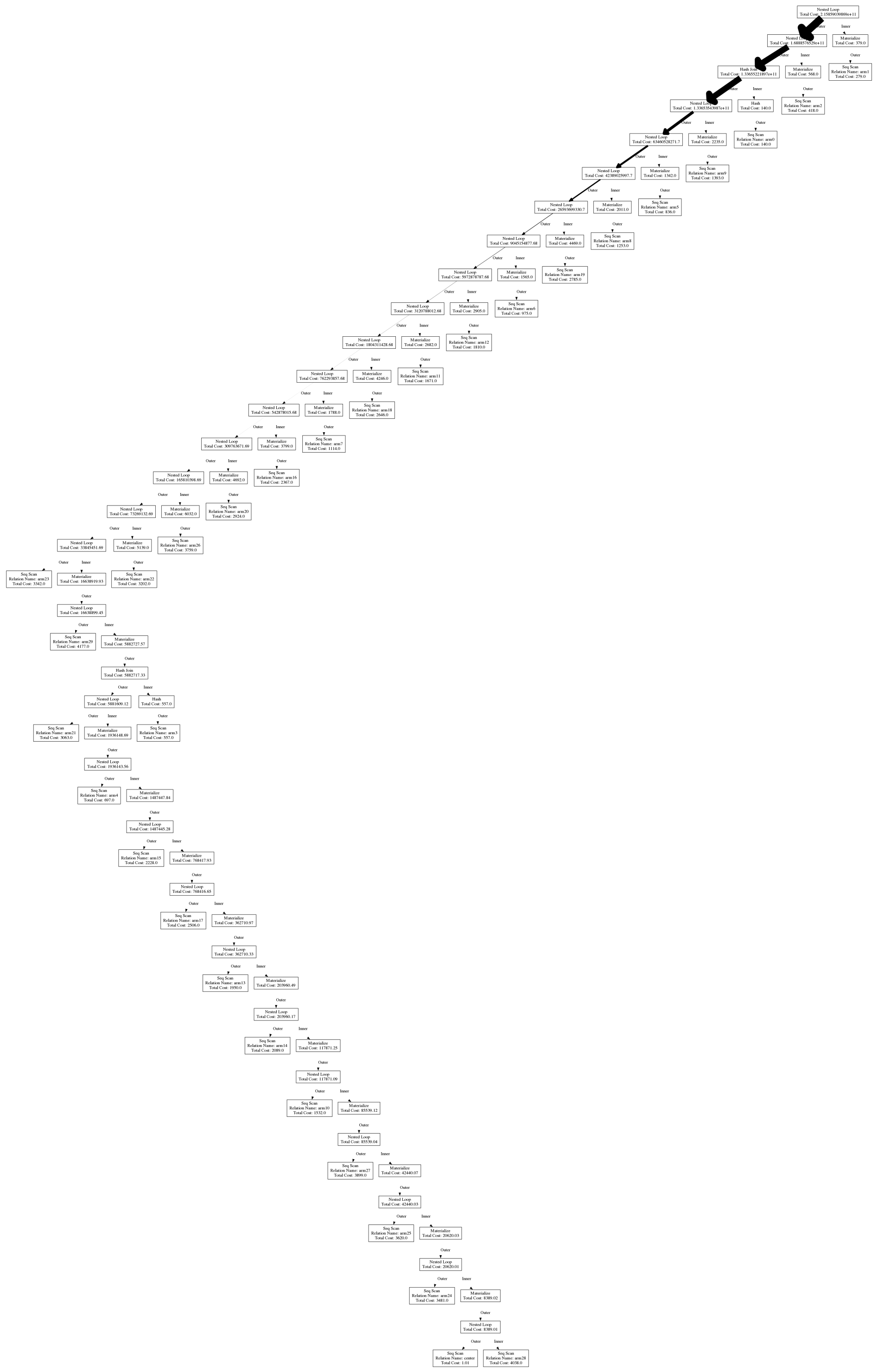
4.3 Sample query trees

To illustrate solutions created by GEQO and SAIO, a few smaller query trees are included

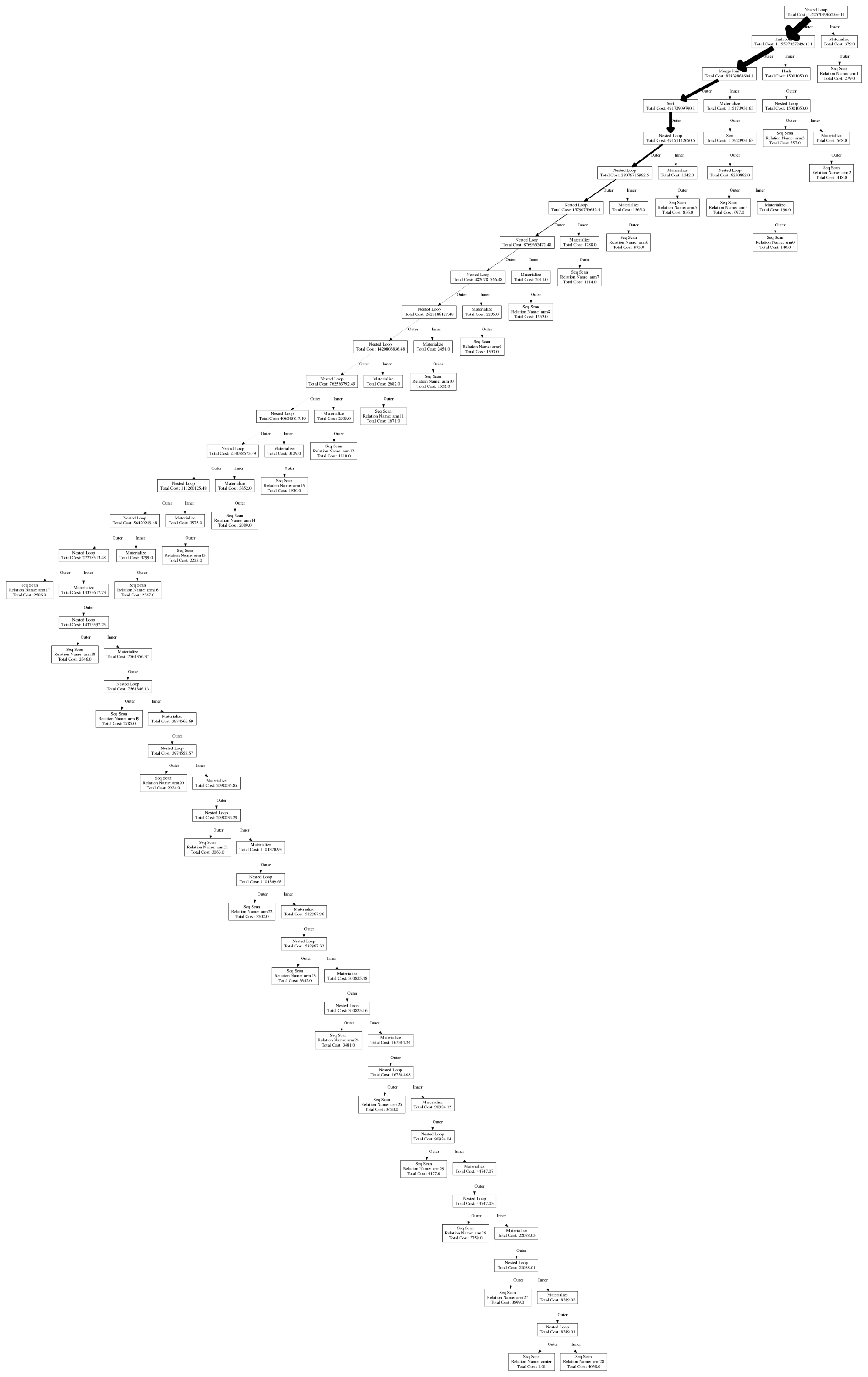
4.3.1 Query trees for cartesians

For queries with cartesian joins we can observe that both GEQO and SAIO produce trees of very similar topology.

GEQO



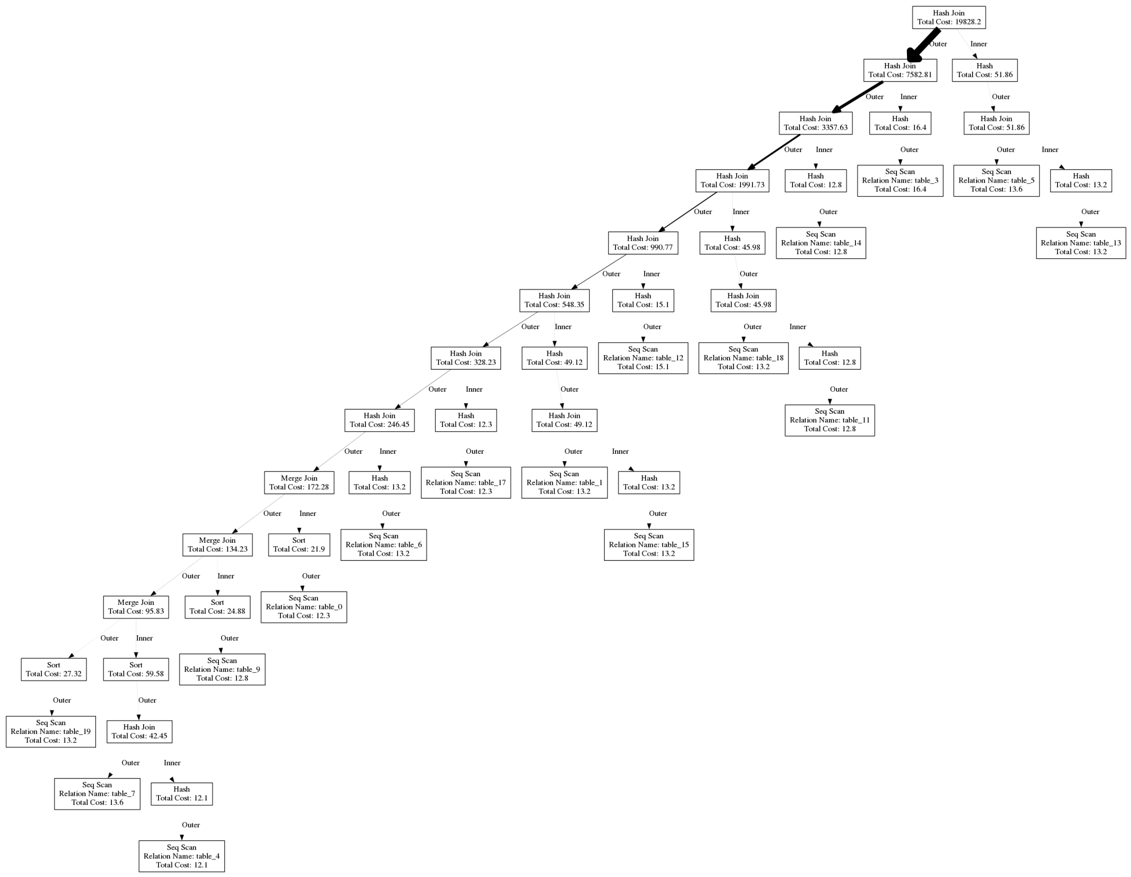
SAIO


Nested Loop Total Cost: 1.62570196528e+11
Hash Join Total Cost: 1.15597327249e+11
Materialize Total Cost: 379.0
Seq Scan Relation Name: arm1 Total Cost: 279.0
Merge Join Total Cost: 82839861604.1
Hash Total Cost: 15001050.0
Nested Loop Total Cost: 15001050.0
Seq Scan Relation Name: arm3 Total Cost: 557.0
Materialize Total Cost: 568.0
Seq Scan Relation Name: arm2 Total Cost: 418.0
Sort Total Cost: 49172909790.1
Materialize Total Cost: 115173931.63
Sort Total Cost: 113923931.63
Nested Loop Total Cost: 6250862.0
Seq Scan Relation Name: arm4 Total Cost: 697.0
Materialize Total Cost: 190.0
Seq Scan Relation Name: arm0 Total Cost: 140.0
Nested Loop Total Cost: 49151142650.5
Nested Loop Total Cost: 28079716992.5
Materialize Total Cost: 1342.0
Seq Scan Relation Name: arm5 Total Cost: 836.0
Nested Loop Total Cost: 15790759652.5
Materialize Total Cost: 1565.0
Seq Scan Relation Name: arm6 Total Cost: 975.0
Nested Loop Total Cost: 8769652472.48
Materialize Total Cost: 1788.0
Seq Scan Relation Name: arm7 Total Cost: 1114.0
Nested Loop Total Cost: 4820781566.48
Materialize Total Cost: 2011.0
Seq Scan Relation Name: arm8 Total Cost: 1253.0
Nested Loop Total Cost: 2627186127.48
Materialize Total Cost: 2235.0
Seq Scan Relation Name: arm9 Total Cost: 1393.0
Nested Loop Total Cost: 1420806836.48
Materialize Total Cost: 2458.0
Seq Scan Relation Name: arm10 Total Cost: 1532.0
Nested Loop Total Cost: 762563792.49
Materialize Total Cost: 2682.0
Seq Scan Relation Name: arm11 Total Cost: 1671.0
Nested Loop Total Cost: 406045817.49
Materialize Total Cost: 2905.0
Seq Scan Relation Name: arm12 Total Cost: 1810.0
Nested Loop Total Cost: 214088573.49
Materialize Total Cost: 3129.0
Seq Scan Relation Name: arm13 Total Cost: 1950.0
Nested Loop Total Cost: 111260125.48
Materialize Total Cost: 3352.0
Seq Scan Relation Name: arm14 Total Cost: 2089.0
Nested Loop Total Cost: 56420249.48
Materialize Total Cost: 3575.0
Seq Scan Relation Name: arm15 Total Cost: 2228.0
Nested Loop Total Cost: 27278513.48
Materialize Total Cost: 3799.0
Seq Scan Relation Name: arm16 Total Cost: 2367.0
Seq Scan Relation Name: arm17 Total Cost: 2506.0
Materialize Total Cost: 14373617.73
Nested Loop Total Cost: 14373597.25
Seq Scan Relation Name: arm18 Total Cost: 2646.0
Materialize Total Cost: 7561356.37
Nested Loop Total Cost: 7561346.13
Seq Scan Relation Name: arm19 Total Cost: 2785.0
Materialize Total Cost: 3974563.69
Nested Loop Total Cost: 3974558.57
Seq Scan Relation Name: arm20 Total Cost: 2924.0
Materialize Total Cost: 2090035.85
Nested Loop Total Cost: 2090033.29
Seq Scan Relation Name: arm21 Total Cost: 3063.0
Materialize Total Cost: 1101370.93
Nested Loop Total Cost: 1101369.65
Seq Scan Relation Name: arm22 Total Cost: 3202.0
Materialize Total Cost: 582967.96
Nested Loop Total Cost: 582967.32
Seq Scan Relation Name: arm23 Total Cost: 3342.0
Materialize Total Cost: 310825.48
Nested Loop Total Cost: 310825.16
Seq Scan Relation Name: arm24 Total Cost: 3481.0
Materialize Total Cost: 167344.24
Nested Loop Total Cost: 167344.08
Seq Scan Relation Name: arm25 Total Cost: 3620.0
Materialize Total Cost: 90924.12
Nested Loop Total Cost: 90924.04
Seq Scan Relation Name: arm29 Total Cost: 4177.0
Materialize Total Cost: 44747.07
Nested Loop Total Cost: 44747.03
Seq Scan Relation Name: arm26 Total Cost: 3759.0
Materialize Total Cost: 22088.03
Nested Loop Total Cost: 22088.01
Seq Scan Relation Name: arm27 Total Cost: 3899.0
Materialize Total Cost: 8389.02
Nested Loop Total Cost: 8389.01
Seq Scan Relation Name: center Total Cost: 1.01
Seq Scan Relation Name: arm28 Total Cost: 4038.0
Outer
Inner

4.3.2 Query trees for joins

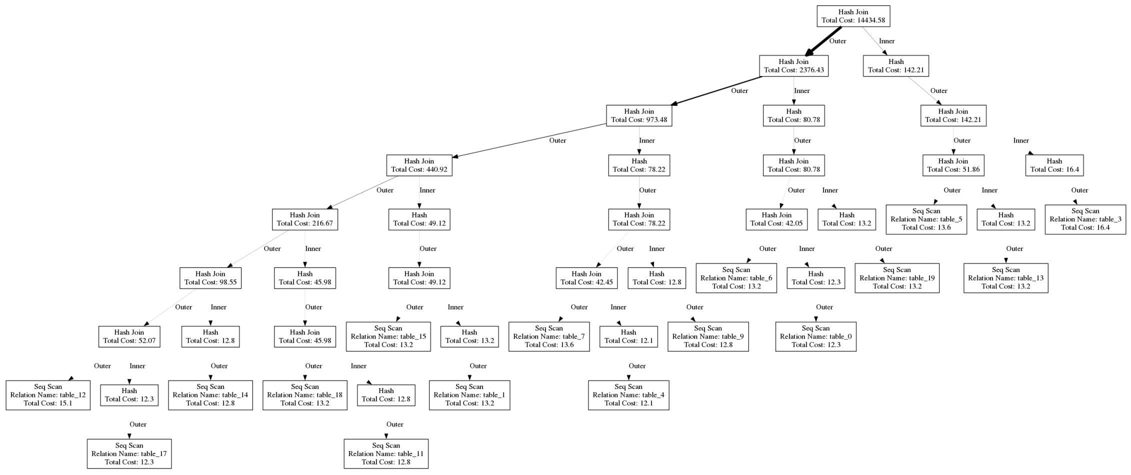
For queries with joins the trees produced by GEQO and SAIO have similar topology.

GEQO



Hash Join Total Cost: 19828.2
Outer
Inner
Hash Join Total Cost: 7582.81
Hash Total Cost: 51.86
Hash Join Total Cost: 3357.63
Hash Total Cost: 16.4
Hash Join Total Cost: 51.86
Hash Join Total Cost: 1991.73
Hash Total Cost: 12.8
Seq Scan Relation Name: table_3 Total Cost: 16.4
Seq Scan Relation Name: table_5 Total Cost: 13.6
Hash Total Cost: 13.2
Hash Join Total Cost: 990.77
Hash Total Cost: 45.98
Seq Scan Relation Name: table_14 Total Cost: 12.8
Seq Scan Relation Name: table_13 Total Cost: 13.2
Hash Join Total Cost: 548.35
Hash Total Cost: 15.1
Hash Join Total Cost: 45.98
Hash Join Total Cost: 328.23
Hash Total Cost: 49.12
Seq Scan Relation Name: table_12 Total Cost: 15.1
Seq Scan Relation Name: table_18 Total Cost: 13.2
Hash Total Cost: 12.8
Hash Join Total Cost: 246.45
Hash Total Cost: 12.3
Hash Join Total Cost: 49.12
Seq Scan Relation Name: table_11 Total Cost: 12.8
Merge Join Total Cost: 172.28
Hash Total Cost: 13.2
Seq Scan Relation Name: table_17 Total Cost: 12.3
Seq Scan Relation Name: table_1 Total Cost: 13.2
Hash Total Cost: 13.2
Merge Join Total Cost: 134.23
Sort Total Cost: 21.9
Seq Scan Relation Name: table_6 Total Cost: 13.2
Seq Scan Relation Name: table_15 Total Cost: 13.2
Merge Join Total Cost: 95.83
Sort Total Cost: 24.88
Seq Scan Relation Name: table_0 Total Cost: 12.3
Sort Total Cost: 27.32
Sort Total Cost: 59.58
Seq Scan Relation Name: table_9 Total Cost: 12.8
Seq Scan Relation Name: table_19 Total Cost: 13.2
Hash Join Total Cost: 42.45
Seq Scan Relation Name: table_7 Total Cost: 13.6
Hash Total Cost: 12.1
Seq Scan Relation Name: table_4 Total Cost: 12.1

SAIO

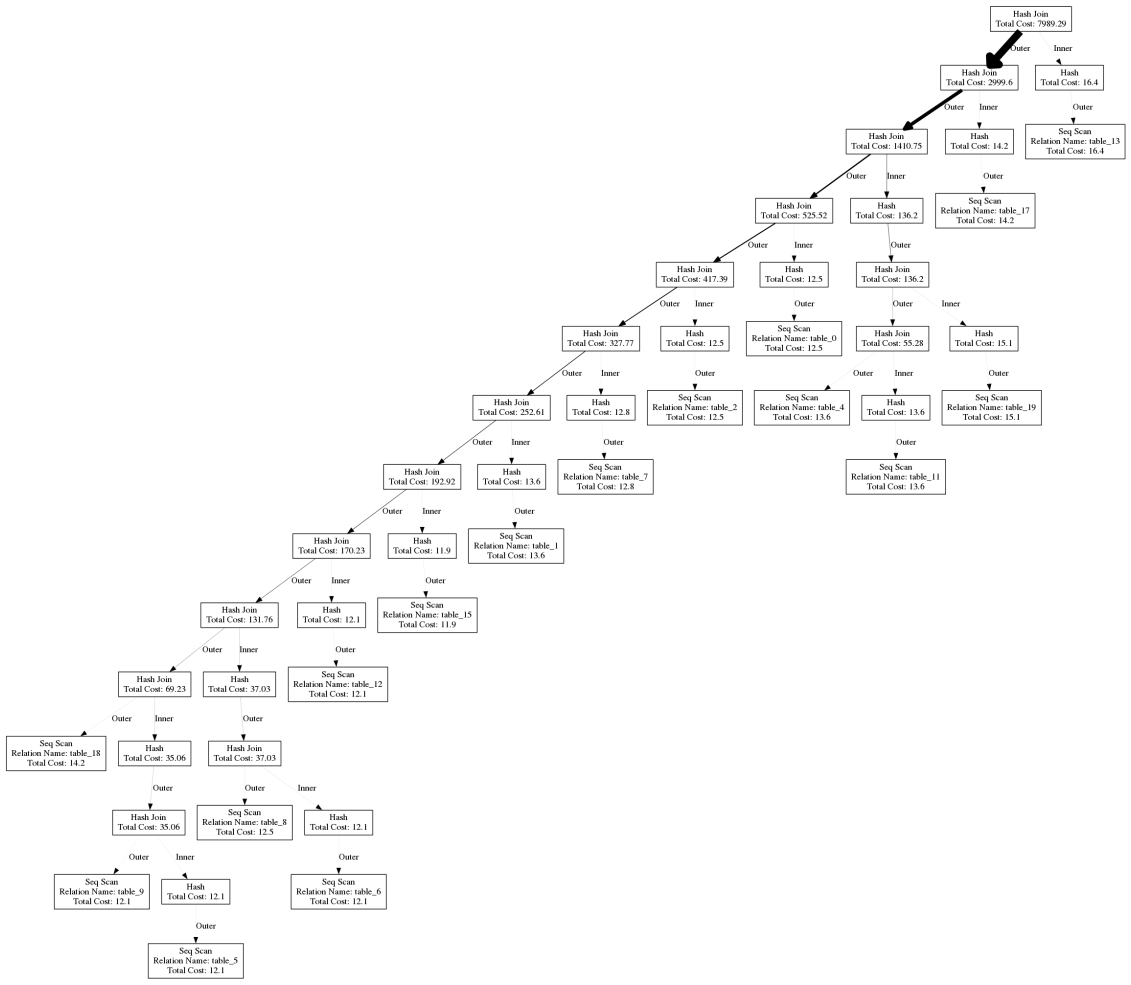


Hash Join Total Cost: 14434.58
Outer
Inner
Hash Join Total Cost: 2376.43
Hash Total Cost: 142.21
Hash Join Total Cost: 973.48
Hash Total Cost: 80.78
Hash Join Total Cost: 142.21
Hash Join Total Cost: 440.92
Hash Total Cost: 78.22
Hash Join Total Cost: 80.78
Hash Join Total Cost: 51.86
Hash Total Cost: 16.4
Hash Join Total Cost: 216.67
Hash Total Cost: 49.12
Hash Join Total Cost: 78.22
Hash Join Total Cost: 42.05
Hash Total Cost: 13.2
Seq Scan Relation Name: table_5 Total Cost: 13.6
Hash Total Cost: 13.2
Seq Scan Relation Name: table_3 Total Cost: 16.4
Hash Join Total Cost: 98.55
Hash Total Cost: 45.98
Hash Join Total Cost: 49.12
Hash Join Total Cost: 42.45
Hash Total Cost: 12.8
Seq Scan Relation Name: table_6 Total Cost: 13.2
Hash Total Cost: 12.3
Seq Scan Relation Name: table_19 Total Cost: 13.2
Seq Scan Relation Name: table_13 Total Cost: 13.2
Hash Join Total Cost: 52.07
Hash Total Cost: 12.8
Hash Join Total Cost: 45.98
Seq Scan Relation Name: table_15 Total Cost: 13.2
Hash Total Cost: 13.2
Seq Scan Relation Name: table_7 Total Cost: 13.6
Hash Total Cost: 12.1
Seq Scan Relation Name: table_9 Total Cost: 12.8
Seq Scan Relation Name: table_0 Total Cost: 12.3
Seq Scan Relation Name: table_12 Total Cost: 15.1
Hash Total Cost: 12.3
Seq Scan Relation Name: table_14 Total Cost: 12.8
Seq Scan Relation Name: table_18 Total Cost: 13.2
Hash Total Cost: 12.8
Seq Scan Relation Name: table_1 Total Cost: 13.2
Seq Scan Relation Name: table_4 Total Cost: 12.1
Seq Scan Relation Name: table_17 Total Cost: 12.3
Seq Scan Relation Name: table_11 Total Cost: 12.8

4.3.3 Query trees for left joins

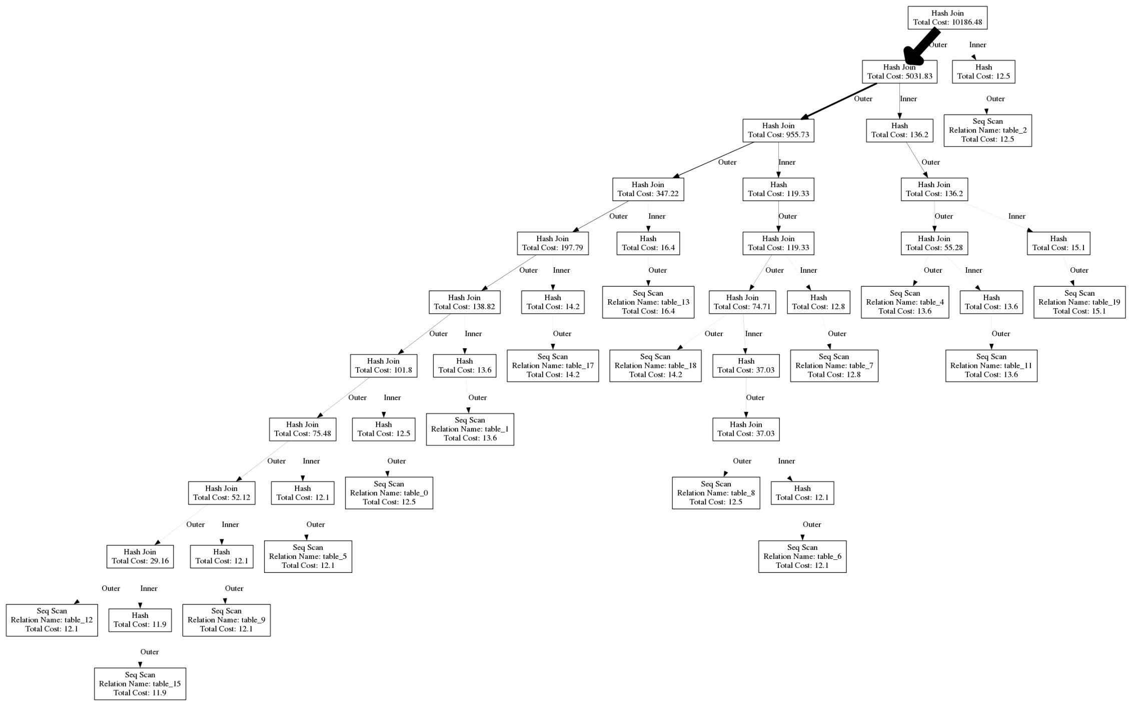
For queries with left joins the trees produced by GEQO and SAIO have similar topology.

GEQO



Hash Join Total Cost: 7989.29
Outer
Inner
Hash Join Total Cost: 2999.6
Hash Total Cost: 16.4
Seq Scan Relation Name: table_13 Total Cost: 16.4
Hash Join Total Cost: 1410.75
Hash Total Cost: 14.2
Seq Scan Relation Name: table_17 Total Cost: 14.2
Hash Join Total Cost: 525.52
Hash Total Cost: 136.2
Hash Join Total Cost: 417.39
Hash Total Cost: 12.5
Hash Join Total Cost: 136.2
Seq Scan Relation Name: table_0 Total Cost: 12.5
Hash Join Total Cost: 327.77
Hash Total Cost: 12.5
Hash Join Total Cost: 55.28
Hash Total Cost: 15.1
Hash Join Total Cost: 252.61
Hash Total Cost: 12.8
Seq Scan Relation Name: table_2 Total Cost: 12.5
Seq Scan Relation Name: table_4 Total Cost: 13.6
Hash Total Cost: 13.6
Seq Scan Relation Name: table_19 Total Cost: 15.1
Hash Join Total Cost: 192.92
Hash Total Cost: 13.6
Seq Scan Relation Name: table_7 Total Cost: 12.8
Seq Scan Relation Name: table_11 Total Cost: 13.6
Hash Join Total Cost: 170.23
Hash Total Cost: 11.9
Seq Scan Relation Name: table_1 Total Cost: 13.6
Hash Join Total Cost: 131.76
Hash Total Cost: 12.1
Seq Scan Relation Name: table_15 Total Cost: 11.9
Hash Join Total Cost: 69.23
Hash Total Cost: 37.03
Seq Scan Relation Name: table_12 Total Cost: 12.1
Seq Scan Relation Name: table_18 Total Cost: 14.2
Hash Total Cost: 35.06
Hash Join Total Cost: 37.03
Hash Join Total Cost: 35.06
Seq Scan Relation Name: table_8 Total Cost: 12.5
Hash Total Cost: 12.1
Seq Scan Relation Name: table_9 Total Cost: 12.1
Hash Total Cost: 12.1
Seq Scan Relation Name: table_6 Total Cost: 12.1
Seq Scan Relation Name: table_5 Total Cost: 12.1

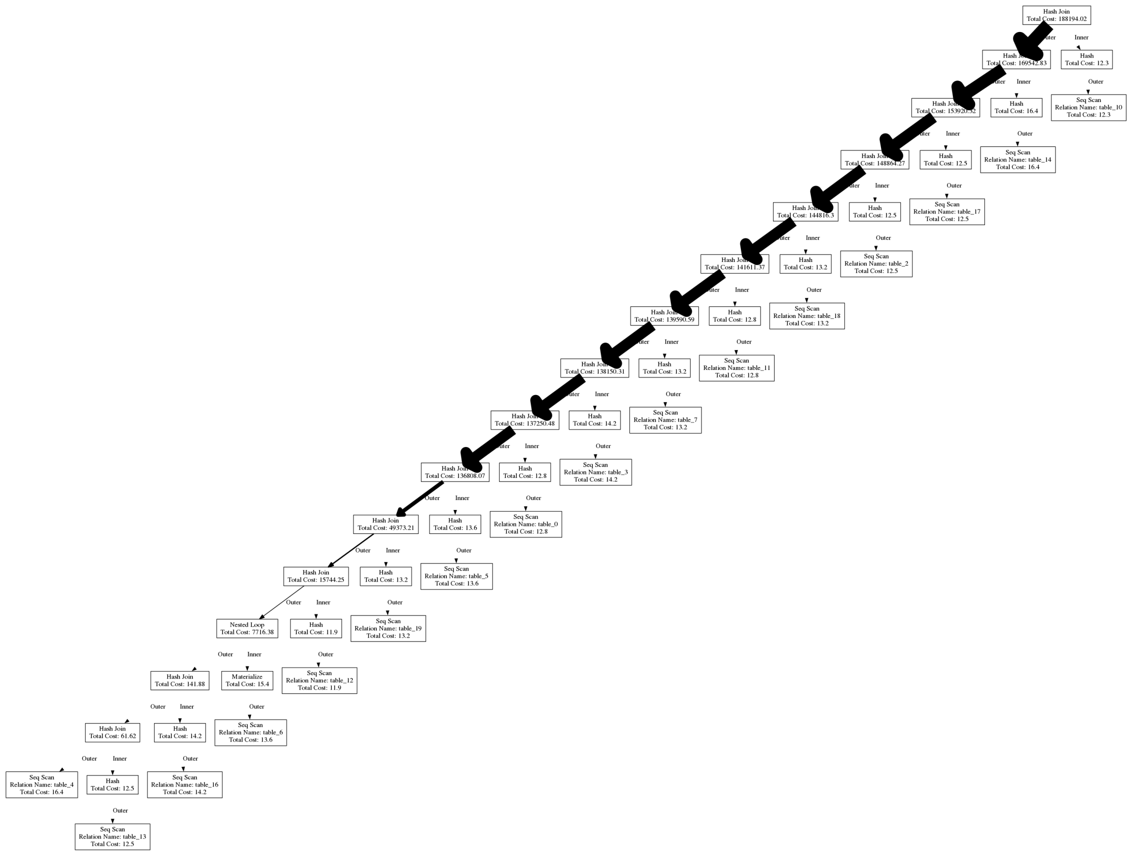
SAIO


Hash Join Total Cost: 10186.48
Outer
Inner
Hash Join Total Cost: 5031.83
Hash Total Cost: 12.5
Outer
Inner
Outer
Hash Join Total Cost: 955.73
Hash Total Cost: 136.2
Seq Scan Relation Name: table_2 Total Cost: 12.5
Outer
Inner
Outer
Hash Join Total Cost: 347.22
Hash Total Cost: 119.33
Hash Join Total Cost: 136.2
Outer
Inner
Outer
Outer
Inner
Hash Join Total Cost: 197.79
Hash Total Cost: 16.4
Hash Join Total Cost: 119.33
Hash Join Total Cost: 55.28
Hash Total Cost: 15.1
Outer
Inner
Outer
Outer
Inner
Outer
Inner
Outer
Hash Join Total Cost: 138.82
Hash Total Cost: 14.2
Seq Scan Relation Name: table_13 Total Cost: 16.4
Hash Join Total Cost: 74.71
Hash Total Cost: 12.8
Seq Scan Relation Name: table_4 Total Cost: 13.6
Hash Total Cost: 13.6
Seq Scan Relation Name: table_19 Total Cost: 15.1
Outer
Inner
Outer
Outer
Inner
Outer
Outer
Hash Join Total Cost: 101.8
Hash Total Cost: 13.6
Seq Scan Relation Name: table_17 Total Cost: 14.2
Seq Scan Relation Name: table_18 Total Cost: 14.2
Hash Total Cost: 37.03
Seq Scan Relation Name: table_7 Total Cost: 12.8
Seq Scan Relation Name: table_11 Total Cost: 13.6
Outer
Inner
Outer
Outer
Hash Join Total Cost: 75.48
Hash Total Cost: 12.5
Seq Scan Relation Name: table_1 Total Cost: 13.6
Hash Join Total Cost: 37.03
Outer
Inner
Outer
Outer
Inner
Hash Join Total Cost: 52.12
Hash Total Cost: 12.1
Seq Scan Relation Name: table_0 Total Cost: 12.5
Seq Scan Relation Name: table_8 Total Cost: 12.5
Hash Total Cost: 12.1
Outer
Inner
Outer
Outer
Hash Join Total Cost: 29.16
Hash Total Cost: 12.1
Seq Scan Relation Name: table_5 Total Cost: 12.1
Seq Scan Relation Name: table_6 Total Cost: 12.1
Outer
Inner
Outer
Seq Scan Relation Name: table_12 Total Cost: 12.1
Hash Total Cost: 11.9
Seq Scan Relation Name: table_9 Total Cost: 12.1
Outer
Seq Scan Relation Name: table_15 Total Cost: 11.9

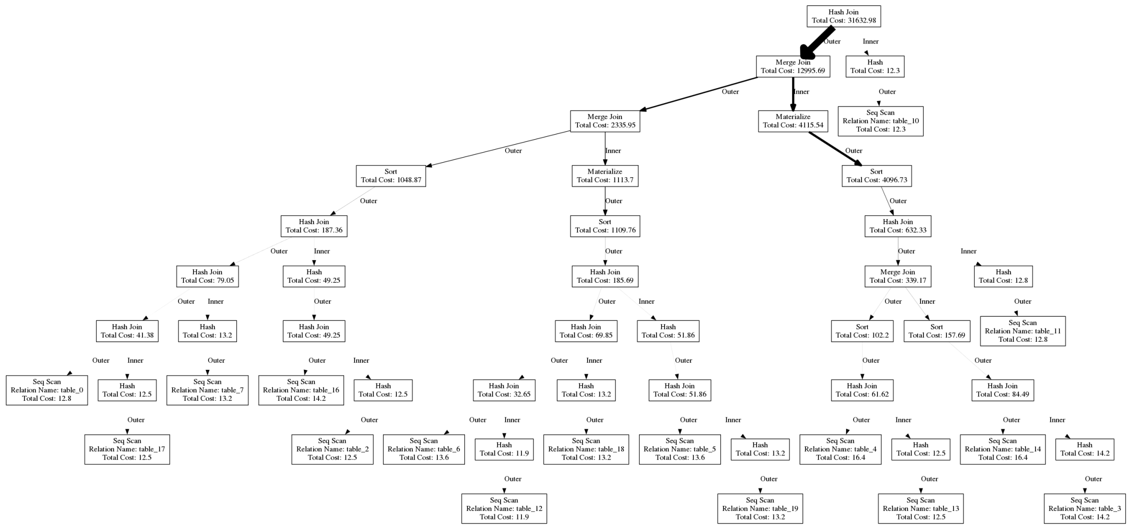
4.3.4 Query trees for right joins

For queries with right joins we can observe that the trees produced by GEQO and SAIO look much different. The GEQO tree is left-deep and SAIO tree is bushy.

GEQO



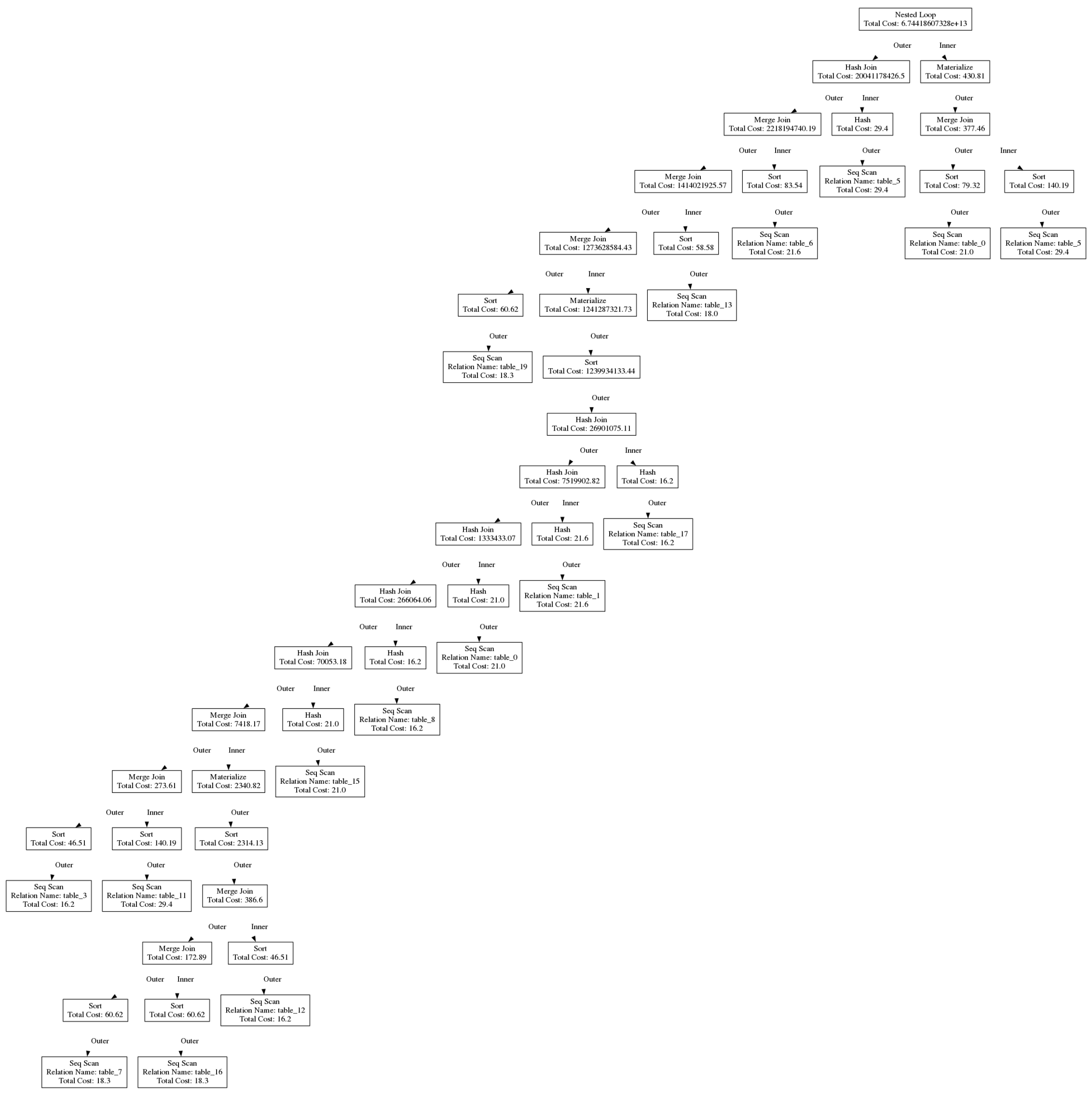
SAIO



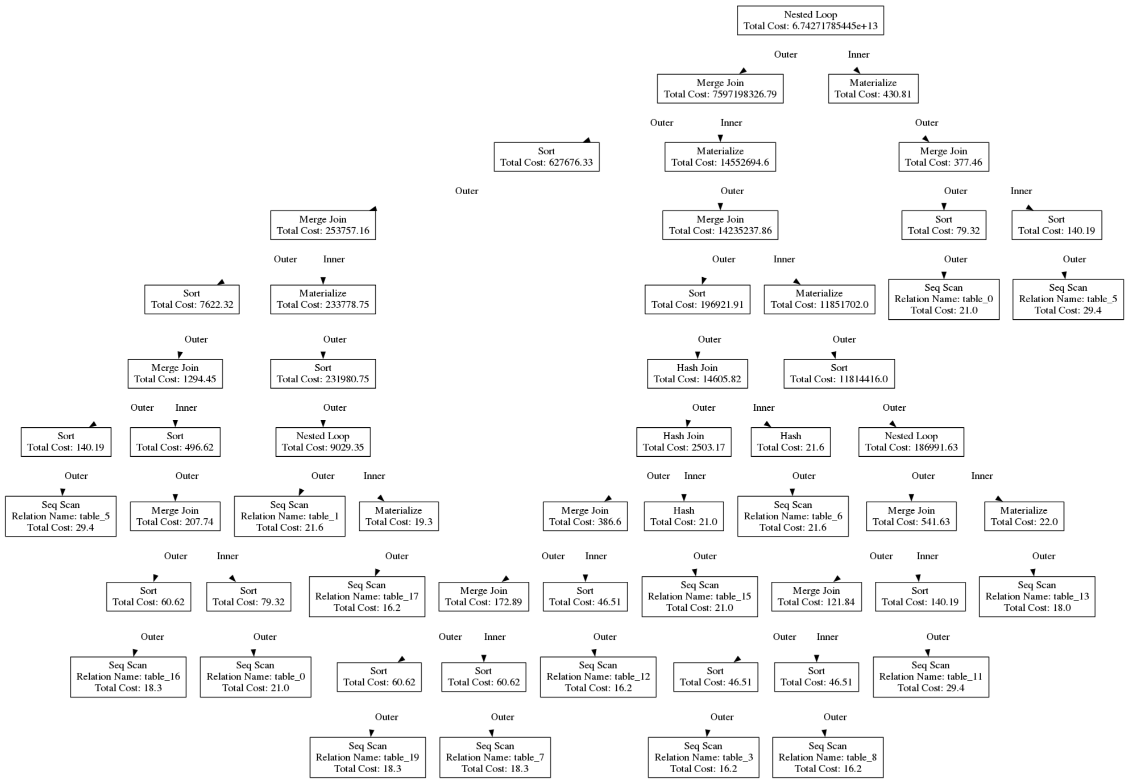
4.3.5 Query trees for nested queries

Here we can observe that trees produced by GEQO and SAIO have different topologies again. This is because we have some right joins here.

GEQO


Nested Loop Total Cost: 6.74418607328e+13
Hash Join Total Cost: 20041178426.5
Materialize Total Cost: 430.81
Merge Join Total Cost: 2218194740.19
Hash Total Cost: 29.4
Merge Join Total Cost: 377.46
Merge Join Total Cost: 1414021925.57
Sort Total Cost: 83.54
Seq Scan Relation Name: table_5 Total Cost: 29.4
Sort Total Cost: 79.32
Sort Total Cost: 140.19
Merge Join Total Cost: 1273628584.43
Sort Total Cost: 58.58
Seq Scan Relation Name: table_6 Total Cost: 21.6
Seq Scan Relation Name: table_0 Total Cost: 21.0
Seq Scan Relation Name: table_5 Total Cost: 29.4
Sort Total Cost: 60.62
Materialize Total Cost: 1241287321.73
Seq Scan Relation Name: table_13 Total Cost: 18.0
Seq Scan Relation Name: table_19 Total Cost: 18.3
Sort Total Cost: 1239934133.44
Hash Join Total Cost: 26901075.11
Hash Join Total Cost: 7519902.82
Hash Total Cost: 16.2
Hash Join Total Cost: 1333433.07
Hash Total Cost: 21.6
Seq Scan Relation Name: table_17 Total Cost: 16.2
Hash Join Total Cost: 266064.06
Hash Total Cost: 21.0
Seq Scan Relation Name: table_1 Total Cost: 21.6
Hash Join Total Cost: 70053.18
Hash Total Cost: 16.2
Seq Scan Relation Name: table_0 Total Cost: 21.0
Merge Join Total Cost: 7418.17
Hash Total Cost: 21.0
Seq Scan Relation Name: table_8 Total Cost: 16.2
Merge Join Total Cost: 273.61
Materialize Total Cost: 2340.82
Seq Scan Relation Name: table_15 Total Cost: 21.0
Sort Total Cost: 46.51
Sort Total Cost: 140.19
Sort Total Cost: 2314.13
Seq Scan Relation Name: table_3 Total Cost: 16.2
Seq Scan Relation Name: table_11 Total Cost: 29.4
Merge Join Total Cost: 386.6
Merge Join Total Cost: 172.89
Sort Total Cost: 46.51
Sort Total Cost: 60.62
Sort Total Cost: 60.62
Seq Scan Relation Name: table_12 Total Cost: 16.2
Seq Scan Relation Name: table_7 Total Cost: 18.3
Seq Scan Relation Name: table_16 Total Cost: 18.3

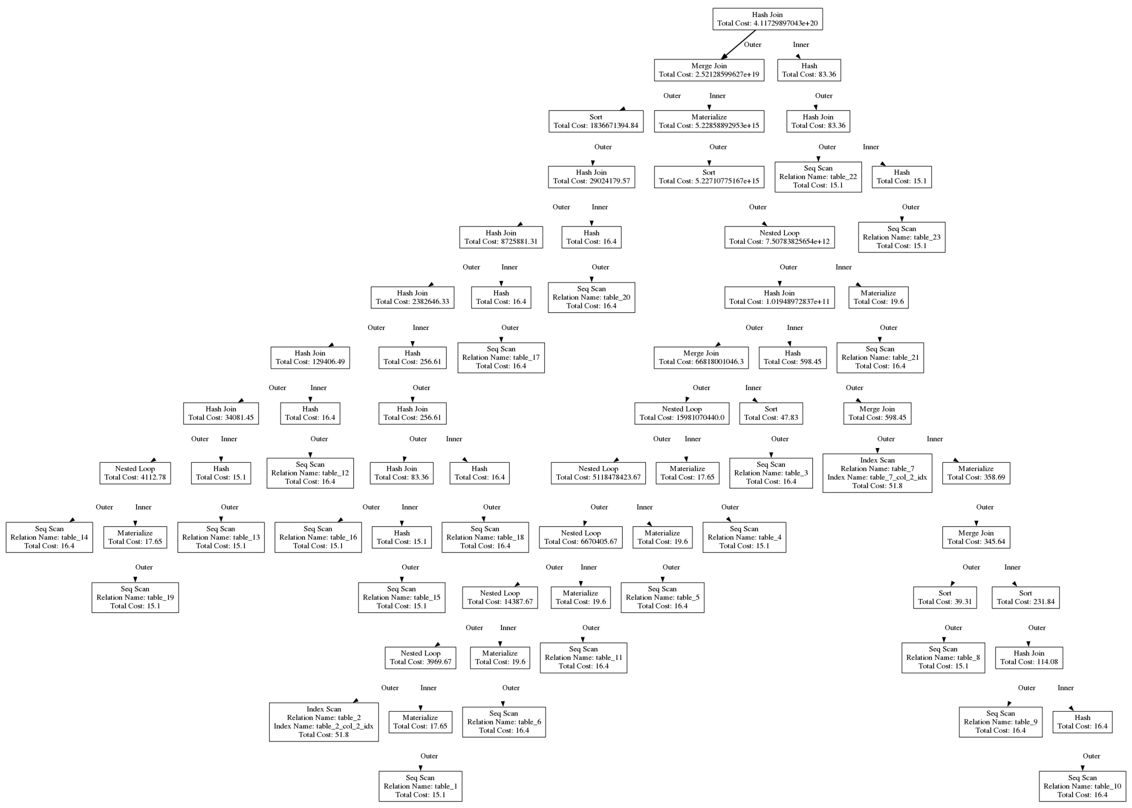
SAIO


Nested Loop
Total Cost: 6.74271785445e+13
Outer
Inner
Merge Join
Total Cost: 7597198326.79
Materialize
Total Cost: 430.81
Outer
Inner
Outer
Sort
Total Cost: 627676.33
Materialize
Total Cost: 14552694.6
Merge Join
Total Cost: 377.46
Outer
Outer
Outer
Inner
Merge Join
Total Cost: 253757.16
Merge Join
Total Cost: 14235237.86
Sort
Total Cost: 79.32
Sort
Total Cost: 140.19
Outer
Inner
Outer
Inner
Outer
Outer
Sort
Total Cost: 7622.32
Materialize
Total Cost: 233778.75
Sort
Total Cost: 196921.91
Materialize
Total Cost: 11851702.0
Seq Scan
Relation Name: table_0
Total Cost: 21.0
Seq Scan
Relation Name: table_5
Total Cost: 29.4
Outer
Outer
Outer
Outer
Merge Join
Total Cost: 1294.45
Sort
Total Cost: 231980.75
Hash Join
Total Cost: 14605.82
Sort
Total Cost: 11814416.0
Outer
Inner
Outer
Outer
Inner
Outer
Sort
Total Cost: 140.19
Sort
Total Cost: 496.62
Nested Loop
Total Cost: 9029.35
Hash Join
Total Cost: 2503.17
Hash
Total Cost: 21.6
Nested Loop
Total Cost: 186991.63
Outer
Outer
Outer
Inner
Outer
Inner
Outer
Outer
Inner
Seq Scan
Relation Name: table_5
Total Cost: 29.4
Merge Join
Total Cost: 207.74
Seq Scan
Relation Name: table_1
Total Cost: 21.6
Materialize
Total Cost: 19.3
Merge Join
Total Cost: 386.6
Hash
Total Cost: 21.0
Seq Scan
Relation Name: table_6
Total Cost: 21.6
Merge Join
Total Cost: 541.63
Materialize
Total Cost: 22.0
Outer
Inner
Outer
Outer
Inner
Outer
Outer
Inner
Outer
Sort
Total Cost: 60.62
Sort
Total Cost: 79.32
Seq Scan
Relation Name: table_17
Total Cost: 16.2
Merge Join
Total Cost: 172.89
Sort
Total Cost: 46.51
Seq Scan
Relation Name: table_15
Total Cost: 21.0
Merge Join
Total Cost: 121.84
Sort
Total Cost: 140.19
Seq Scan
Relation Name: table_13
Total Cost: 18.0
Outer
Outer
Outer
Inner
Outer
Outer
Inner
Outer
Seq Scan
Relation Name: table_16
Total Cost: 18.3
Seq Scan
Relation Name: table_0
Total Cost: 21.0
Sort
Total Cost: 60.62
Sort
Total Cost: 60.62
Seq Scan
Relation Name: table_12
Total Cost: 16.2
Sort
Total Cost: 46.51
Sort
Total Cost: 46.51
Seq Scan
Relation Name: table_11
Total Cost: 29.4
Outer
Outer
Outer
Outer
Seq Scan
Relation Name: table_19
Total Cost: 18.3
Seq Scan
Relation Name: table_7
Total Cost: 18.3
Seq Scan
Relation Name: table_3
Total Cost: 16.2
Seq Scan
Relation Name: table_8
Total Cost: 16.2

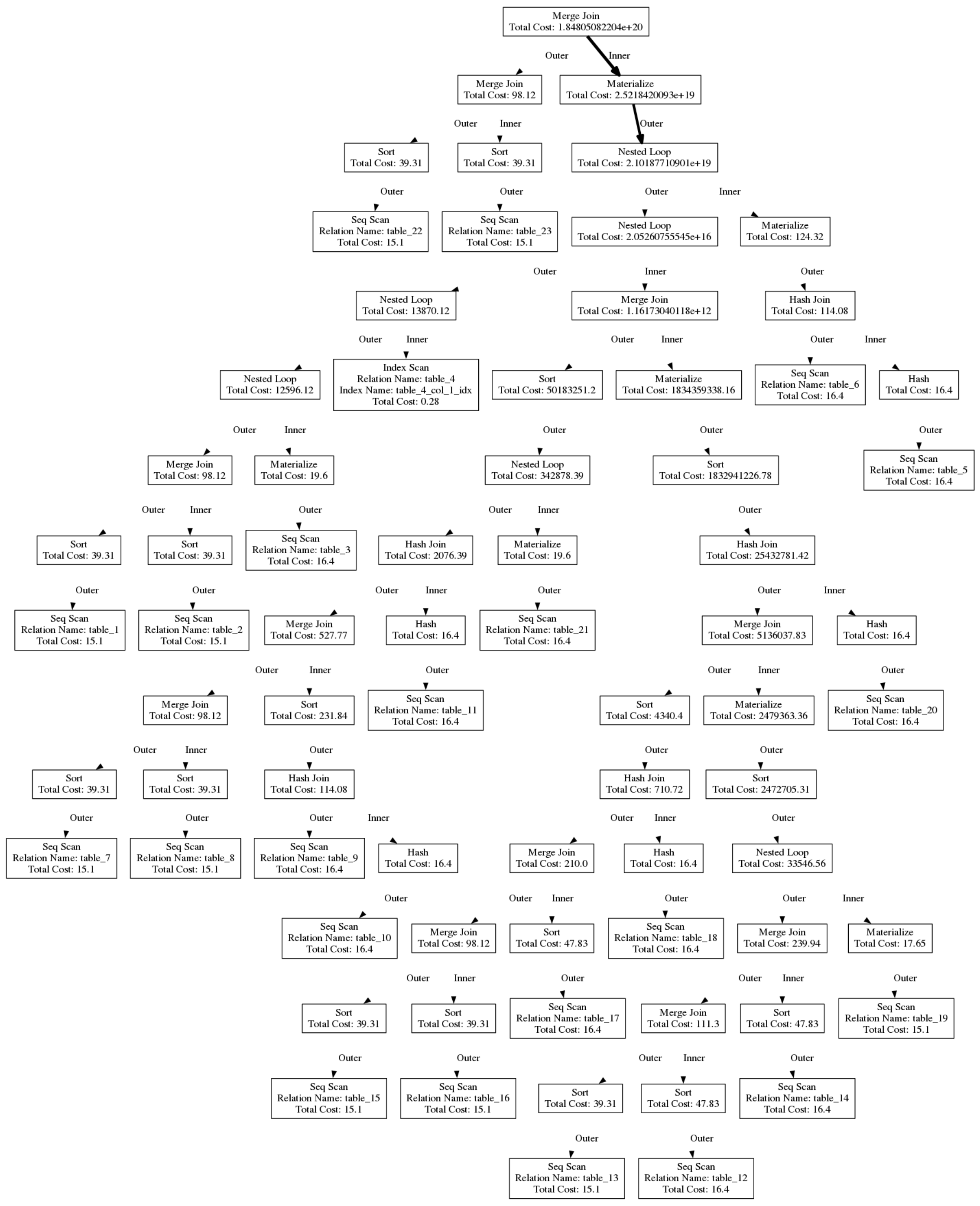
4.3.6 Query trees for double nested query

Here both trees look pretty bushy.

GEQO



SAIO



Merge Join Total Cost: 1.84805082204e+20
Merge Join Total Cost: 98.12
Materialize Total Cost: 2.5218420093e+19
Sort Total Cost: 39.31
Sort Total Cost: 39.31
Nested Loop Total Cost: 2.10187710901e+19
Seq Scan Relation Name: table_22 Total Cost: 15.1
Seq Scan Relation Name: table_23 Total Cost: 15.1
Nested Loop Total Cost: 2.05260755545e+16
Materialize Total Cost: 124.32
Nested Loop Total Cost: 13870.12
Merge Join Total Cost: 1.16173040118e+12
Hash Join Total Cost: 114.08
Nested Loop Total Cost: 12596.12
Index Scan Relation Name: table_4 Index Name: table_4_col_1_idx Total Cost: 0.28
Sort Total Cost: 50183251.2
Materialize Total Cost: 1834359338.16
Seq Scan Relation Name: table_6 Total Cost: 16.4
Hash Total Cost: 16.4
Merge Join Total Cost: 98.12
Materialize Total Cost: 19.6
Nested Loop Total Cost: 342878.39
Sort Total Cost: 1832941226.78
Seq Scan Relation Name: table_5 Total Cost: 16.4
Sort Total Cost: 39.31
Sort Total Cost: 39.31
Seq Scan Relation Name: table_3 Total Cost: 16.4
Hash Join Total Cost: 2076.39
Materialize Total Cost: 19.6
Hash Join Total Cost: 25432781.42
Seq Scan Relation Name: table_1 Total Cost: 15.1
Seq Scan Relation Name: table_2 Total Cost: 15.1
Merge Join Total Cost: 527.77
Hash Total Cost: 16.4
Seq Scan Relation Name: table_21 Total Cost: 16.4
Merge Join Total Cost: 5136037.83
Hash Total Cost: 16.4
Merge Join Total Cost: 98.12
Sort Total Cost: 231.84
Seq Scan Relation Name: table_11 Total Cost: 16.4
Sort Total Cost: 4340.4
Materialize Total Cost: 2479363.36
Seq Scan Relation Name: table_20 Total Cost: 16.4
Sort Total Cost: 39.31
Sort Total Cost: 39.31
Hash Join Total Cost: 114.08
Hash Join Total Cost: 710.72
Sort Total Cost: 2472705.31
Seq Scan Relation Name: table_7 Total Cost: 15.1
Seq Scan Relation Name: table_8 Total Cost: 15.1
Seq Scan Relation Name: table_9 Total Cost: 16.4
Hash Total Cost: 16.4
Merge Join Total Cost: 210.0
Hash Total Cost: 16.4
Nested Loop Total Cost: 33546.56
Seq Scan Relation Name: table_10 Total Cost: 16.4
Merge Join Total Cost: 98.12
Sort Total Cost: 47.83
Seq Scan Relation Name: table_18 Total Cost: 16.4
Merge Join Total Cost: 239.94
Materialize Total Cost: 17.65
Sort Total Cost: 39.31
Sort Total Cost: 39.31
Seq Scan Relation Name: table_17 Total Cost: 16.4
Merge Join Total Cost: 111.3
Sort Total Cost: 47.83
Seq Scan Relation Name: table_19 Total Cost: 15.1
Seq Scan Relation Name: table_15 Total Cost: 15.1
Seq Scan Relation Name: table_16 Total Cost: 15.1
Sort Total Cost: 39.31
Sort Total Cost: 47.83
Seq Scan Relation Name: table_14 Total Cost: 16.4
Seq Scan Relation Name: table_13 Total Cost: 15.1
Seq Scan Relation Name: table_12 Total Cost: 16.4
Outer
Inner