REPORT

The Fair Baskets Problem

Abstract

An insight to the program implementation and it’s evaluation

220026989

- Variables, domains and Constraints

Since most of the variables to be used in the program was given, the next step would be to define the range of the variables in the eprime file.

❖ Given:

```
given n_producers : int given n_products : int given n_weeks : int
given product_category : matrix indexed by [int(1..n_products)] of
int(1..3) given product_type : matrix indexed by [int(1..n_products)]
of int(1..n_products) given product_price : matrix indexed by
[int(1..n_products)] of int(..) given product_producer : matrix indexed
by [int(1..n_products)] of int(1..n_producers) given product_quantity :
matrix indexed by [int(1..n_products)] of int(..) given basket_price :
matrix indexed by [int(1..3)] of int(..) given max_type : int
given same_type : bool
```

Since n_products, n_producers, n_weeks, max_type are single value integers, they are just defined as integer variables. The rest of the variables(excluding same_type – Boolean variable) are matrices and are defined with a range:

The product_category, product_type, product_price, product_producer, product_quantity contain n_product number of elements,as the incorporate details of n_products. The domain range of product_category is from 1 to 3, as there are only 3 categories. The domain range of product type is defined from 1 to n_products as there should be atleast one product_type and there can only be a maximum of n_product types(when each of the product has different values). The range of product_price and product_quantity is not defined within a particular limit as there is no limit to both. The values in the product_producer matrix can only vary between 1 and n_producers value and is therefore defined as such.

❖ Lettings:

```
letting max_price = max(basket_price) letting
individual_max_price = max(product_price)
letting max_type_product = max(product_type)
letting min_price = min(product_price)
 letting CATEGORY be domain int(1..3) letting
WEEKS be domain int(1..n_weeks) letting
PRODUCTS be domain int(1..n_products) letting
PRODUCERS be domain int(1..n_producers)
letting TYPE be domain
int(1..max_type_product)
```

The max variables are defined as to be used as the range on further lettings. CATEGORY, WEEKS, PRODUCTS, PRODUCERS, TYPE are defined as domains for categories, n_weeks, n_products, n_producers and max_type_product respectively.

```
find b : matrix indexed by [int(1..3), int(1..n_weeks), int(1..n_products)] of
int(0..max_type) find optimal : int(0..(max_price-min_price)*n_weeks*3)
```

The b matrix is the result required. The range of each element in the b matrix is from 0 to max_type. This is because the matrix skeleton is of the matrix is defined in such a way that all those elements (product quantities) that do not belong to a given category(row) is zero. The maximum quantity of a given type has been restricted to max_type by the administrator and thus restricting the value for each element.

The optimal value can range between 0 and a (maximum of max_price-min_price)*n_weeks*3, where max_price and min_price are defined above. The maximum range of the optimal is defined according to the final optimal function, which subtracts the basket value from product value and summed to n_weeks and then summed to max_category.

- Constraints used

```
   $separating out categories in the b matrix
         forAll i : CATEGORY.
          forAll j : WEEKS.
           forAll k : PRODUCTS.
            (product_category[k] != i) -> b[i,j,k] = 0,

        forAll i : CATEGORY.
         forAll j : WEEKS.
          forAll k : PRODUCTS.
           (product_category[k] = i) -> ((b[i,j,k] >= 0) /\ (b[i,j,k] <=
max_type) /\ (b[i,j,k] <= product_quantity[k])),

  $atleast 3 products in a basket
     forAll i : CATEGORY.
      forAll j : WEEKS.
        3 <= sum k : PRODUCTS. (b[i,j,k] != 0),

   $max_type constraint
    forAll i : CATEGORY.
      forAll j : WEEKS.
```

```
            forAll k : TYPE.
             max_type >=  sum l : PRODUCTS.(product_type[l] = k) * (b[i,j,l]),

        $a basket should not contain just the same type
         forAll i : CATEGORY.
          forAll j : WEEKS.
           forAll k : TYPE.
             1 <= sum l : PRODUCTS. ((b[i,j,l] != 0) /\ (product_type[l] != k)),

        $same producer constraint
         (same_type) ->
           forAll i : CATEGORY.
            forAll j : WEEKS.
             forAll k : PRODUCTS.(b[i,j,k] != 0) -> (product_type[k] =
product_producer[k]) ->
              forAll l : PRODUCTS. (product_type[l] = product_type[k]) ->
(product_producer[l] = product_producer[k]),


       $atleast one producer constraint
         forAll j : WEEKS.
           forAll k : PRODUCERS.
            1 <= sum i : CATEGORY.
             sum l : PRODUCTS.
             ((b[i,j,l] != 0) /\ product_producer[l] = k),

       $product quantity constraint
         forAll i : PRODUCTS.
          product_quantity[i] >= sum j : CATEGORY.
                          sum k : WEEKS. (b[j,k,i]),

       $symmetry breaking
        forAll i : CATEGORY.
         forAll j : int(1..n_weeks-1).
          forAll k : int(j+1..n_weeks).
           b[i,j,..] <=lex b[i,k,..],


       $price and final optimize
         optimal = sum i : CATEGORY.
                    sum k : WEEKS.
                     |basket_price[i]-sum l : PRODUCTS.
                       (b[i,k,l]*product_price[l])|,

            $Here the basket price is subtracted from the sum of the products for
each basket, each week. The subtracted basket price is then added to the optimal
which is then optimized.
```

- Symmetry Breaking Constraint

```
forAll i : CATEGORY.  forAll j : int(1..n_weeks-1).
forAll k : int(j+1..n_weeks).    b[i,j,..] <=lex
b[i,k,..],
```

A simple lexicographical ordering between the weeks for a given category. The product matrix of the weeks will be arranged in an ascending order inorder to break the row symmetry between weeks.

- Basic Empirical Evaluation

Param	Optimal	SolverNodes	SolverSolveTime (s)	SavileRowTotalTime (s)	SolverTotalTime (s)
basic	10	19	0.00E+00	0.151	0
easy1	26	468	0.00172	0.229	0.002958
easy2	7	11166	0.03648	0.226	0.037815
easy3	91	3149	0.010885	0.219	0.011394
med1	7	2587	0.00715	0.233	0.008474
med2	20	775148	0.00715	0.233	3.49283
med3	5	712851	2.52365	0.435	2.52487
hard1	17	117582439	604.641	0.222	604.642
hard2	37	37479	0.178072	0.22	0.179358
hard3	99	1117262685	3974.29	0.209	3974.29

- b matrix for each file:

  - basic.param

    [[[1, 5, 5, 0, 0, 0, 0, 0, 0],
    [3, 3, 1, 0, 0, 0, 0, 0, 0]],
    [[0, 0, 0, 1, 4, 1, 0, 0, 0],
    [0, 0, 0, 1, 4, 1, 0, 0, 0]],
    [[0, 0, 0, 0, 0, 0, 6, 1, 5],
    [0, 0, 0, 0, 0, 0, 6, 5, 1]]]

- easy1.param

```
[[[0, 0, 2, 1, 0, 0, 0, 0, 0, 3],
[0, 0, 3, 1, 0, 0, 0, 0, 0, 3],
[0, 0, 3, 2, 0, 0, 0, 0, 0, 3],
[0, 0, 3, 3, 0, 0, 0, 0, 0, 3]],
[[0, 2, 0, 0, 1, 0, 0, 1, 1, 0],
[0, 2, 0, 0, 1, 0, 0, 1, 1, 0],
[0, 2, 0, 0, 1, 0, 0, 1, 1, 0],
[0, 2, 0, 0, 1, 0, 0, 1, 1, 0]],
[[2, 0, 0, 0, 0, 2, 1, 0, 0, 0],
[2, 0, 0, 0, 0, 2, 1, 0, 0, 0],
[2, 0, 0, 0, 0, 2, 1, 0, 0, 0],
[2, 0, 0, 0, 0, 2, 1, 0, 0, 0]]]
```

- easy2.param

```
[[[1, 1, 0, 2, 0, 0, 0, 0, 0, 0],
[1, 1, 0, 2, 0, 0, 0, 0, 0, 0],
[1, 1, 0, 2, 0, 0, 0, 0, 0, 0],
[1, 1, 0, 2, 0, 0, 0, 0, 0, 0]],
[[0, 0, 0, 0, 1, 4, 2, 0, 0, 0],
[0, 0, 0, 0, 2, 1, 2, 0, 0, 0],
[0, 0, 0, 0, 2, 1, 2, 0, 0, 0],
[0, 0, 0, 0, 2, 1, 2, 0, 0, 0]],
[[0, 0, 0, 0, 0, 0, 0, 1, 1, 2],
[0, 0, 0, 0, 0, 0, 0, 1, 1, 2],
[0, 0, 0, 0, 0, 0, 0, 1, 1, 2],
[0, 0, 0, 0, 0, 0, 0, 1, 1, 2]]]
```

- easy3.param

```
[[[1, 1, 1, 0, 0, 0, 0, 0, 0, 0],
[1, 1, 1, 0, 0, 0, 0, 0, 0, 0],
[1, 1, 2, 0, 0, 0, 0, 0, 0, 0],
[1, 2, 2, 0, 0, 0, 0, 0, 0, 0]],
[[0, 0, 0, 1, 1, 1, 0, 0, 0, 0],
[0, 0, 0, 1, 1, 1, 0, 0, 0, 0],
[0, 0, 0, 2, 1, 1, 0, 0, 0, 0],
[0, 0, 0, 2, 1, 1, 0, 0, 0, 0]],
[[0, 0, 0, 0, 0, 0, 0, 2, 1, 1],
[0, 0, 0, 0, 0, 0, 1, 2, 2, 2],
[0, 0, 0, 0, 0, 0, 2, 2, 0, 1],
```

[0, 0, 0, 0, 0, 0, 2, 2, 1, 0]]]

- med1.param

[[[1, 3, 1, 0, 0, 0, 0, 0, 0, 0],
[2, 2, 2, 0, 0, 0, 0, 0, 0, 0],
[2, 2, 2, 0, 0, 0, 0, 0, 0, 0],
[2, 2, 2, 0, 0, 0, 0, 0, 0, 0]],
[[0, 0, 0, 1, 3, 3, 0, 0, 0, 0],
[0, 0, 0, 1, 3, 3, 0, 0, 0, 0],
[0, 0, 0, 1, 3, 3, 0, 0, 0, 0],
[0, 0, 0, 3, 2, 2, 0, 0, 0, 0]],
[[0, 0, 0, 0, 0, 0, 0, 2, 2, 3],
[0, 0, 0, 0, 0, 0, 0, 2, 3, 3], [0,
0, 0, 0, 0, 0, 3, 1, 3, 3],
[0, 0, 0, 0, 0, 0, 3, 3, 0, 2]]]

- med2.param

[[[0, 2, 1, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
[0, 2, 1, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
[0, 2, 1, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
[0, 2, 1, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]],
[[0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0],
[0, 0, 0, 0, 2, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0],
[0, 0, 0, 0, 2, 2, 1, 1, 0, 0, 0, 0, 0, 0, 0],
[0, 0, 0, 0, 2, 2, 1, 2, 0, 0, 0, 0, 0, 0, 0]],
[[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 2, 1, 2],
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 2, 1, 2],
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 2, 2, 0],
[0, 0, 0, 0, 0, 0, 0, 0, 2, 2, 1, 2, 0, 2, 0]]]

- med3.param

[[[0, 0, 0, 3, 3, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
[1, 0, 2, 3, 3, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
[3, 3, 0, 1, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
[3, 3, 2, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]],
[[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 3, 0, 0, 2, 0, 0, 0, 0, 0],
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 1, 0, 0, 1, 0, 0, 0, 0, 0],
[0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 2, 0, 0, 0, 0, 0],
[0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 2, 0, 0, 0, 0, 0]],
[[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1],
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1],

[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1],
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1]]]

- hard1.param

```
[[[0, 0, 0, 1, 0, 0, 1, 0, 4, 0],
[0, 0, 0, 1, 0, 0, 1, 0, 4, 0],
[0, 0, 0, 1, 0, 0, 2, 0, 1, 0],
[0, 0, 0, 1, 0, 0, 2, 0, 1, 0]],
[[0, 0, 0, 0, 1, 0, 0, 3, 0, 4],
[0, 0, 0, 0, 3, 0, 0, 3, 0, 2],  [0,
0, 0, 0, 4, 0, 0, 3, 0, 1],
[0, 0, 3, 0, 1, 0, 0, 2, 0, 4]],  [[1,
1, 0, 0, 0, 3, 0, 0, 0, 0],
[3, 3, 0, 0, 0, 4, 0, 0, 0, 0],
[4, 1, 0, 0, 0, 4, 0, 0, 0, 0],
[4, 4, 0, 0, 0, 3, 0, 0, 0, 0]]]
```

- hard2.param

```
[[[0, 0, 1, 0, 0, 0, 1, 0, 0, 1],
[0, 2, 1, 0, 0, 0, 2, 0, 0, 3],
[0, 3, 1, 0, 0, 0, 2, 0, 0, 3],
[0, 3, 2, 0, 0, 0, 3, 0, 0, 1]],
[[0, 0, 0, 1, 0, 0, 0, 2, 2, 0],
[0, 0, 0, 1, 0, 0, 0, 2, 2, 0],
[0, 0, 0, 1, 0, 0, 0, 2, 2, 0],
[0, 0, 0, 1, 0, 0, 0, 2, 2, 0]],
[[2, 0, 0, 0, 2, 1, 0, 0, 0, 0],
[3, 0, 0, 0, 2, 1, 0, 0, 0, 0],
[3, 0, 0, 0, 2, 1, 0, 0, 0, 0],
[3, 0, 0, 0, 2, 1, 0, 0, 0, 0]]]
```

- hard3.param

```
[[[3, 0, 0, 1, 0, 0, 0, 1, 0, 0],
[3, 0, 0, 1, 0, 0, 0, 1, 0, 0],
[3, 0, 0, 1, 0, 0, 0, 1, 0, 0],
[3, 0, 0, 1, 0, 0, 0, 3, 0, 0]],
[[0, 0, 0, 0, 2, 1, 0, 0, 0, 1],
[0, 0, 0, 0, 4, 2, 0, 0, 0, 4],
[0, 0, 0, 0, 4, 2, 0, 0, 0, 4],
[0, 0, 0, 0, 4, 3, 0, 0, 0, 2]],
[[0, 1, 0, 0, 0, 0, 1, 0, 3, 0],
```

[0, 3, 0, 0, 0, 0, 1, 0, 3, 0],
[0, 4, 0, 0, 0, 0, 1, 0, 3, 0],
[0, 4, 0, 0, 0, 0, 1, 0, 3, 0]]]

- Additional Empirical Evaluation

  - Symmetry breaking

    Without the lexical constraint:

Param	Optimal	SolverNodes	SolverTotalTime
basic	10	26	1.00E-06
easy1	26	1083	0.003128
easy2	7	371170	0.827261
easy3	91	1726607	6.34677
med1	7	679300	1.29144
med2	20	40784088	155.54
med3	5	12659676	40.2063
hard1	17	X	>5400
hard2	37	X	>5400
hard3	99	X	>5400

  - Without using custom search heuristic

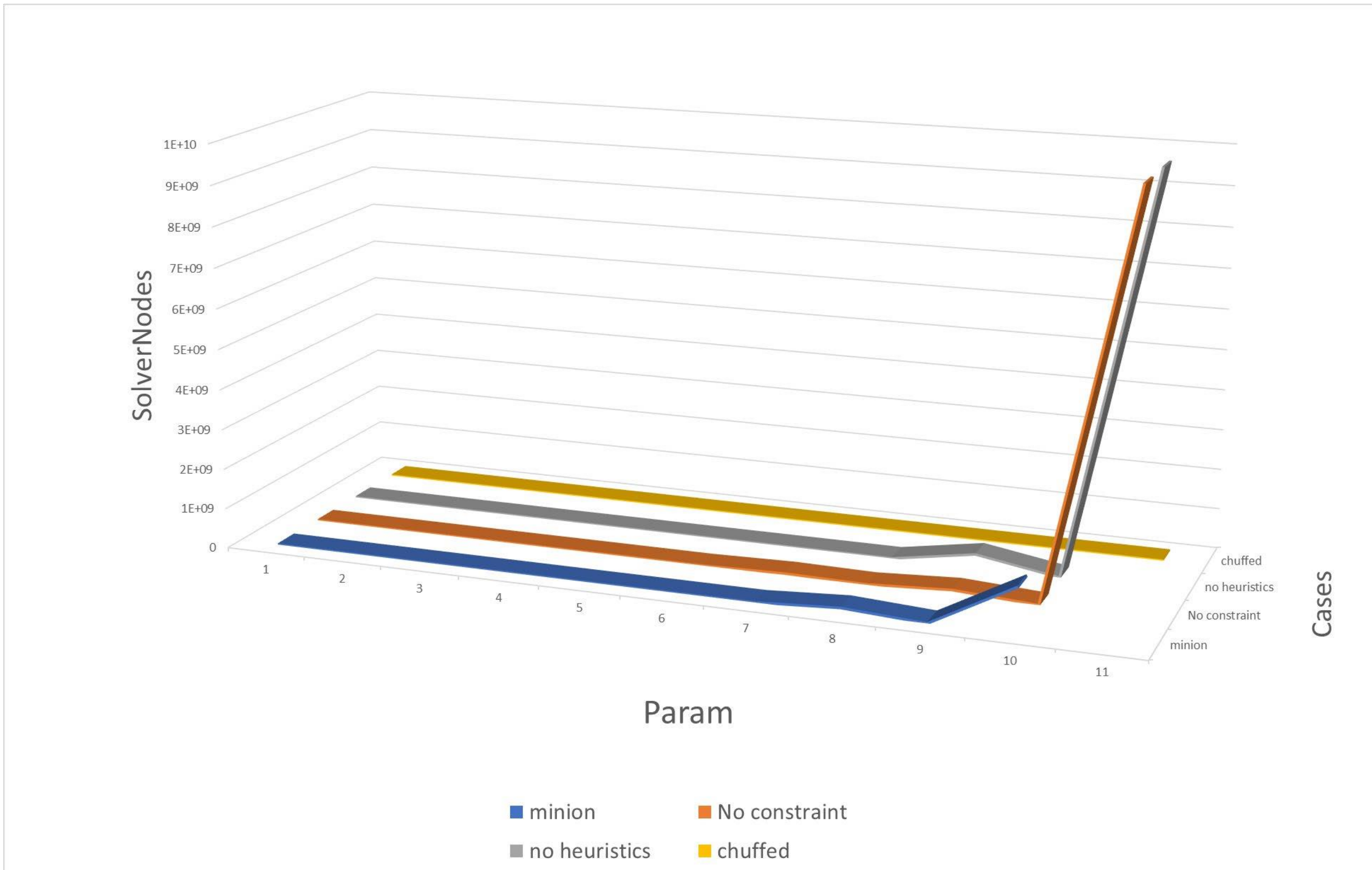
    (Without using branching on):

Param	Optimal	SolverNodes	SolverTotalTime
basic	10	823	0.002024
easy1	26	920	0.004446
easy2	7	16006	0.053656
easy3	91	2889	0.010642
med1	7	97246	0.239011
med2	20	840263	3.59721
med3	5	403959	1.43398
hard1	17	337346072	1644
hard2	37	6594	0.030579
hard3	99	X	>5400

- Using -chuffed

Param	Optimal	SolverNodes	SolverTotalTime
basic	10	25	3.00E-03
easy1	26	3891	0.021
easy2	7	136	0.005
easy3	91	585	0.007
med1	7	77	0.226
med2	20	13345	0.095
med3	5	365944	16.899
hard1	17	11493	0.06
hard2	37	8885	0.094
hard3	99	22719	0.202

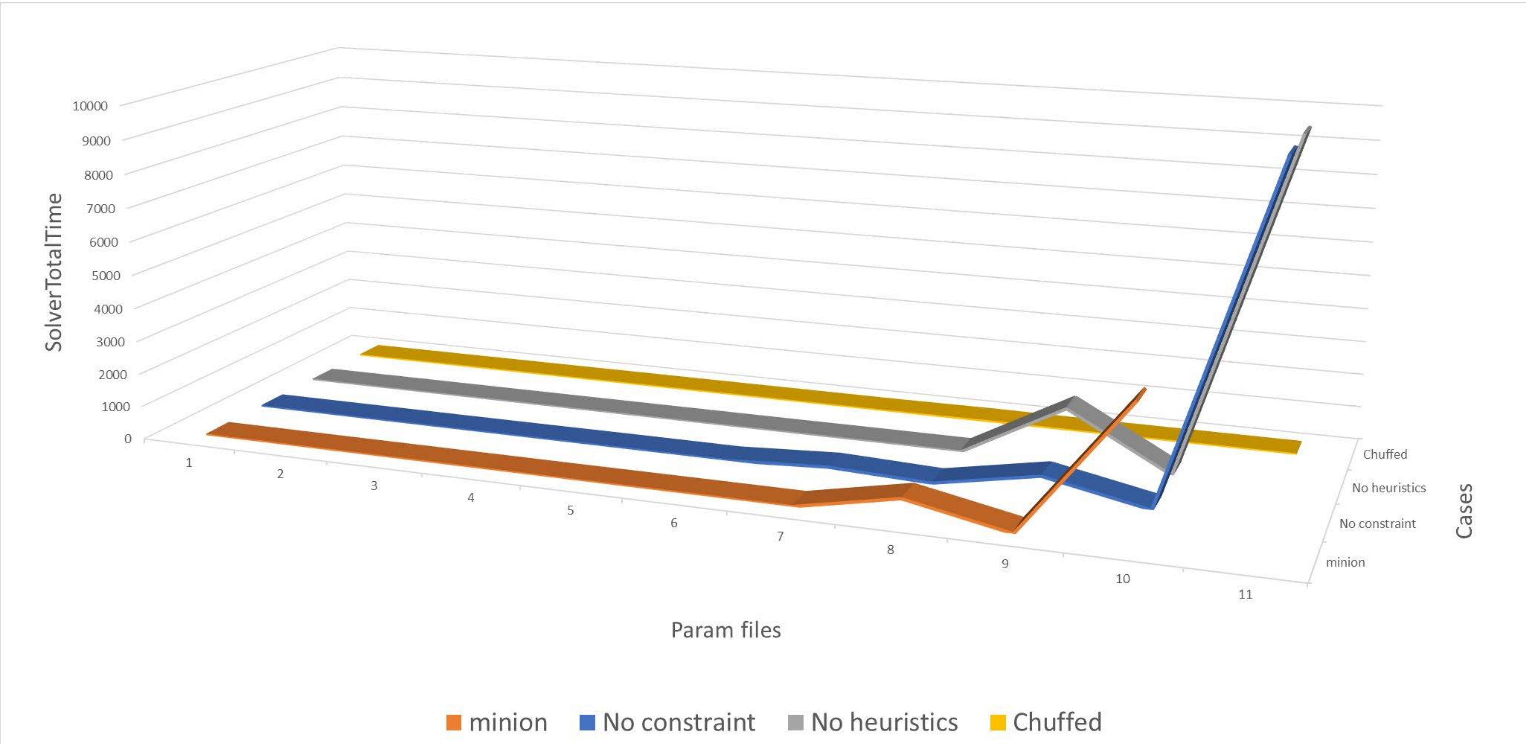
❖ Comparing all the 4 cases:
  - ✓ Solver nodes comparison



Review:

As we can see from the graph the solverNodes are very high for No-constraints and No heuristics which is heavy for the solver and time consuming (see the next graph). Furthermore we can see that the best optimized (solverNodes case) for the written eprime code is with chuffed.

✓ Comparing solver Time



Review:

As we can see from the graph the solverTotalTime are very high for No-constraints and No heuristics as it is heavy for the solver . Furthermore we can see that the best optimized (Time case) for the written eprime code is with -chuffed, which compared to the minion time is flat.