Fish-SAT

Josiah Blaisdell

June 2020

1 Introduction

Changing ocean conditions due to global climate change make fishery science and oceanography a hot big data topic. Scientists are interested in understanding how the fish respond to the changing conditions. Hundreds of millions of dollars worth of fish are shipped all over the world from the Gulf of Alaska every year. Many jobs depend on the fish being available every year, even under changing conditions. As a part of the National Research Traineeship (NRT), I am building a tool to use machine learning and remotely sensed data to create "stress-scapes" that segment the GOA through time and space according to regions that stress fish in similar ways.

My collaborator has various models for fish that describe the growth rate as a function of sea surface temperature, oxygen content, ocean acidity and other variables. We consider a fish to be stressed if its growth rate is far from optimal. We would like to determine if there are values for SST, O2 and pH for which the maximum growth rate (under linear constraints) for all the models is above some threshold. Most of the models we have worked with so far are "quadratic" in the independent variables.

I will use the PySMT library to solve the problem of Quadratic Programming with Linear Constraints as it can be applied to fish growth rate models. A quadratic polynomial of degree 2 with k variables is made of of $\binom{2+k}{2}$ terms. For example, a polynomial of degree 2 in 2 variables has the following form:

$$p(x, y) = Ax^2 + By^2 + Cxy + Dx + Ey + F$$

where, F consists of the sum of the x^0 term and the y^0 term. Notice that there are $\binom{4}{2=6}$ terms. This polynomial can be represented by a matrix:

$$p(x, y) = d^t x + x^t M x + F$$

where d is the vector of linear terms coefficients. For example, the polynomial $p(x, y) = x^2 - 2xy + y^2 - 5x - 2y$ can be represented as follows:

$$\begin{bmatrix} x & y \end{bmatrix} \begin{pmatrix} 1 & -1 \\ -1 & 1 \end{pmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} x & y \end{bmatrix} \begin{bmatrix} -5 \\ -2 \end{bmatrix}$$

In general, to find the matrix representation, separate the constant and linear terms, then find a matrix that, when conjugated by a vector representation of the remaining terms, gives the original equation. This can be a little bit tricky, but for polynomials of degree two in 2 variables the matrix is:

$$\begin{pmatrix} A & C/2 \\ C/2 & B \end{pmatrix}$$

Since:

$$\begin{bmatrix} x & y \end{bmatrix} \begin{pmatrix} A & C/2 \\ C/2 & B \end{pmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} x & y \end{bmatrix} \begin{bmatrix} Ax + Cy/2 \\ Cx/2 + By \end{bmatrix} = Ax^2 + By^2 + Cxy$$

It turns out that in general, the matrix is the Hessian (as seen below). If the (Hessian) matrix M is positive semi-definite, the problem of finding quadratic programming is NP, but in general, the problem is NP-complete. It is not known whether or not fish bioenergetic models have positive semi-definite matrix representations, but this is highly unlikely given the number of variables in the ocean and the amount of co-variance between the different variables and growth rates.

2 Problem Definition

There are three formulations of the quadratic programming problem with linear constraints. One formulation stipulates that the matrix representation of the objective function is positive semi-definite. This ensures that all the eigenvalues are positive. Solving QP problems where the matrix is positive semidefinite can be done in polynomial time via LU-Factorization and then finding the directions of greatest and least change at each local minima and maxima. The problem gets more complicated when the matrix representation of the objective function is not positive semi-definite. By "not positive semi-definite" I mean that the matrix representation has both positive and negative eigenvalues. In this case, numerical methods do not converge and the problem is NP-COMPLETE.

2.1 Related: ILP Problem

The integer linear programming problem asks, given a system of inequalities with integer coefficients, whether there are integer values that the variables in the inequalities can take that will satisfy all expressions in the system. For example, the following is a system on inequalities:

$$2x - 5y \geq 8$$
$$3x + 9y \geq -12$$

This problem is related in the sense that it helped me to understand how one might implement a solver for the quadratic programming problem with

linear constraints. In particular, the addition and multiplication of numbers and variables are converted to binary. Bitwise addition and multiplication is used to convert the mathematical expression into a series of and's and or's on the individual bits that make up the variables and the constants in the system of inequalities. The SAT solver determines if there is a set of ones and zeroes that can be assigned to the bits of the variables to make the expression evaluate to true. In order to better understand the integer linear programming problem and its conversion to a SAT I refer the interested reader to (Warner, 1998) and (Li, 2004).

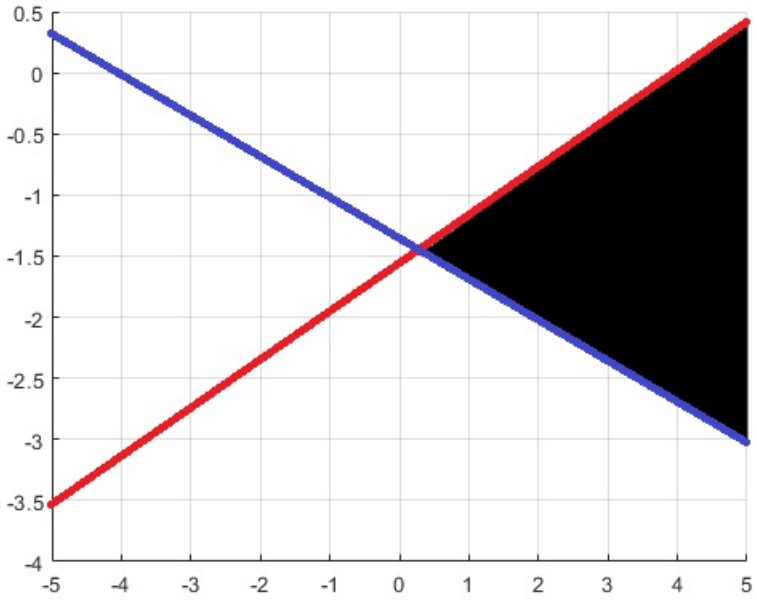

Figure 1: A plot of the two equations and the solution space is given, the red line is the first equation, the blue line is the second equation, the black is the solution space. ILP asks whether there exists integer values in the solution space.

2.2 Reduction from Clique to QP1NE

Given a graph $G = (V, E)$, a subset $Q = \{V_i\}$ of vertices is called a clique if and only if there is an edge between any two vertices in Q. The clique problem asks, given $k \in \mathbf{N}$ whether there is a clique of size k in G. Clique is an NP-HARD problem, this was shown in [Kar72] via reduction from 3-SAT to Vertex cover to Clique.

The problem QP1NE describes all quadratic programming problems where the objective function has one negative eigenvalue. Clique is relevant because it can be reduced to QP1NE [PV91]. In the reduction, variables in the QP1NE objective function are created from the vertices of the CLIQUE problem. For all pairs of vertices where an edge is not present a linear constraint is added $x_i + x_j \leq 1$. Finally, the last constraint is added $x_i + ... + x_n = k$. The idea is that there is a click of size k if and only if the optimum of the QP1NE instance is zero. The proof in the $\rightarrow$ direction relies on the definition of the "binary feasible point". For the proof in the $\leftarrow$ direction assumes that the optimum value of QP1NE is zero, so it must be that $x_i \in \{0, 1\}$ for all i hence there is clearly x_i such that $x_i = 1$ form a k-clique.

2.3 Reduction from Knapsack to general QP

The general quadratic programming problem considers polynomials whose hessian is not positive semi definite (1 or more negative eigenvalues). In the Knapsack problem, we are given a knapsack W. W has a fixed maximum weight that it can carry. We are also given a collection of items, numbered from 1 to n. Each item we are given has a weight w_i and a value v_i. We are asked to maximize the objective function

$$\sum_{i=1}^{n} v_i x_i$$

subject to the constraint

$$\sum_{i=1}^{n} w_i x_i \leq W$$

In the simplest case we take x_i to be the number of instances of item i that appear in the knapsack, and typically this is assumed to be either 0 or 1 but bounding the number of copies, or letting the number of copies of an item be unbounded is does not change the NP-Hardness of the knapsack problem. The knapsack problem is shown to be NP-hard via reduction from 3-SAT to SUBSET-SUM to Knapsack in [Kar72].

The knapsack problem is related to the general quadratic programming problem. In the general quadratic programming problem we are given an objective function that has one or more negative eigenvalues in the matrix representation. The knapsack problem is reduced to the general quadratic programming problem in [Sah74].

3 Reduction from QP to 3-SAT

3.1 Introduction

For my final project I am given a collection of fish models that describe growth rate as a function of various ocean variables. For the sake of brevity I will used

x_j for the variables and g^i to refer to the growth models. For example, for 3 variables we are given

$$\begin{aligned} g^1(x_1,x_2) &= a_{11}x_1^2 + a_{12}x_2^2 + a_{13}x_1x_2 + a_{14}x_1 + a_{15}x_2 + a_{16}+ \\ g^2(x_1,x_2) &= a_{21}x_1^2 + a_{22}x_2^2 + a_{23}x_1x_2 + a_{24}x_1 + a_{25}x_2 + a_{26}+ \\ &\ldots \\ g^n(x_1,x_2) &= a_{n1}x_1^2 + a_{n2}x_2^2 + a_{n3}x_1x_2 + a_{n4}x_1 + a_{n5}x_2 + a_{n6}+ \end{aligned}$$

Each of these functions has matrix representations, in particular, if we define g_1 to be the derivative of g with respect to x_1 and x_2, to be g_1 and g_2 respectively and we define g_{12} and g_{21} to be the partial derivatives of g in the usual way then we can define the matrix representation as follows

$$g^i(x_1,x_2) = \begin{bmatrix} x_1 & x_2 \end{bmatrix} \begin{pmatrix} g^i_{11} & g^i_{12} \\ g^i_{21} & g^i_{22} \end{pmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} x_1 & x_2 \end{bmatrix} \begin{bmatrix} a_{in} \\ -2 \end{bmatrix}$$

For example, in 2 variables, with second degree polynomials we have that:

$$g^i(x_1,x_2) = a_{i1}x_1^2 + a_{i2}x_2^2 + a_{i3}x_1x_2 + a_{i4}x_1 + a_{i5}x_2 + a_{i6}$$

We have that:

$$\begin{aligned} \frac{\partial^2 g^i}{\partial x_1^2} &= 2a_{i1} \\ \frac{\partial g^i}{\partial x_1}\frac{\partial g^i}{\partial x_2} &= a_{i3} \\ \frac{\partial g^i}{\partial x_2}\frac{\partial g^i}{\partial x_1} &= a_{i3} \\ \frac{\partial^2 g^i}{\partial x_2^2} &= 2a_{i2} \end{aligned}$$

The matrix representation of g^i becomes:

$$\begin{aligned} g^i(x_1,x_2) &= \frac{1}{2}\begin{bmatrix} x_1 & x_2 \end{bmatrix} \begin{pmatrix} 2a_{i1} & a_{i3} \\ a_{i3} & 2a_{i2} \end{pmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} x_1 & x_2 \end{bmatrix} \begin{bmatrix} a_{i4} \\ a_{i5} \end{bmatrix} + a_{i6} \\ &= \begin{bmatrix} x_1 & x_2 \end{bmatrix} \begin{bmatrix} a_{i1}x_1 + \frac{1}{2}a_{i3}x_2 \\ \frac{1}{2}a_{i3}x_1 + a_{i2}x_2 \end{bmatrix} + \begin{bmatrix} x_1 & x_2 \end{bmatrix} \begin{bmatrix} a_{i4} \\ a_{i5} \end{bmatrix} + a_{i6} \\ &= x_1^2 + 2x_1x_2 + x_1 + x_2 + 1 \end{aligned}$$

We can define polynomials of degree two whose matrix representation has a negative eigenvalue. For example consider the objective function:

$$\text{MINIMIZE: } g^i(x_1,x_2) = x_1^2 + 2x_1x_2$$

The matrix representing this polynomial is

$$M = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$$

This matrix has eigenvalues $\{-.618, 1.618\}$. A plot of this polynomial is given in figure 2.

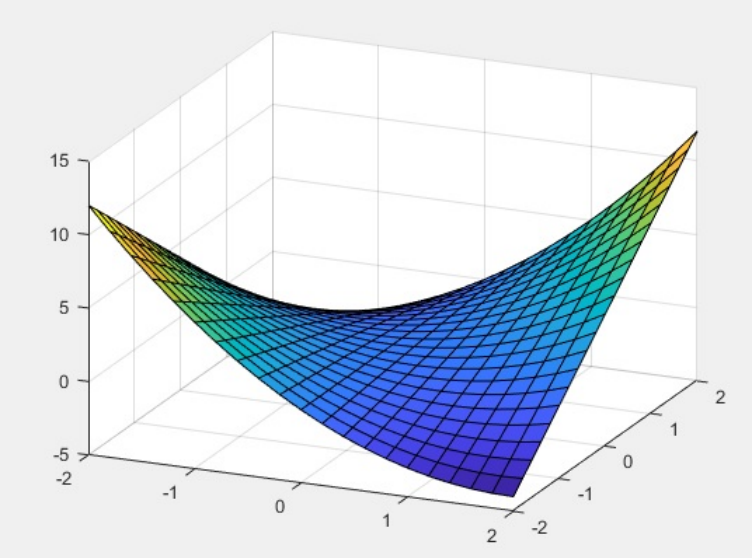

Figure 2: A plot of $g^i(x_1, x_2) = x_1^2 + 2x_1x_2$ in MATLAB.

Consider the linear constraint $x_1 + x_2 \leq 1$. Figure 3 below colors all points satisfying this linear constraint yellow.

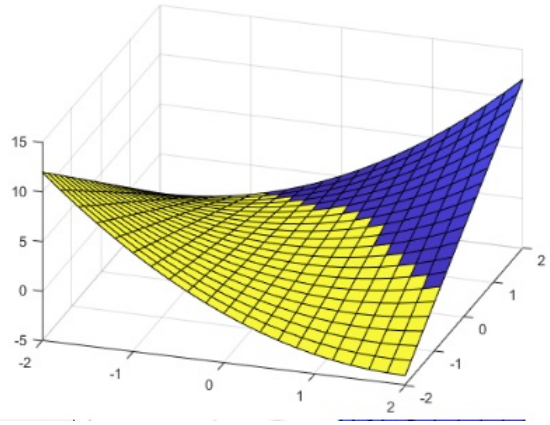

Figure 3: A plot of $g^i(x_1, x_2) = x_1^2 + 2x_1x_2$ under the linear constraint $x_1 + x_2 \leq 1$, the yellow points are the ones satisfying the constraint.

As the number of functions, variables, degree of the polynomial and con-

straints increases, it becomes more and more difficult to find the regions satisfying the constraint.

PySMT offers a way to find points satisfying the objective function all constraints across all functions.

3.2 Program Input

The input to my program will be a file that describes the quadratic programming file. Figure 4 provides an example. The format of the file will be

1. list of variables
2. number of variables number of objective functions
3. list of objective functions
4. list of constraints

Each element of the list of objective functions is defined as follows:

1. Minimum value of expression
2. x_0^2 coefficient
3. $x_0 * x1$ coefficient
4. ...
5. constant term
6. Maximum value of expression

In the following example we see 2 objective functions. Each objective function is of 2 variables: x_0 and x_1. The first line lists the variables, the second line lists the number of variables followed by the number of expressions. Lines 3 and 4 are the expressions. The first value in lines 3 and 4 are the minimum values for the expression. The last value in lines 3 and 4 are the maximum values for the expression. Lines 5-8 describe the linear constraints. The first value is the minimum for the linear constraint, the last value is the maximum for the linear constraint. The first expression at line 3 can be written as follows:

$$
\begin{aligned}
\textit{objective}: \quad & 0.608x_0^2 + 0.134x_0x_1 + 0.719x_1^2 + 0.816x_0 - 0.967x_1 + 0.898 \\
\text{constraint } 1: \quad & -.572 \leq .125x_0 + .151x_1 + .486 \leq .306 \\
\text{constraint } 2: \quad & -.918 \leq .621x_0 - .333x_1 + .162 \leq .774 \\
\text{constraint } 3: \quad & -.016 \leq -.19x_0 + .007x_1 - .201 \leq .215 \\
\text{constraint } 4: \quad & -.157 \leq .504x_0 - .742x_1 + .351 \leq .377
\end{aligned}
$$

According to the definition in lines 3, Fish-SAT will verify that the minimum value of the objective function under the constraints is greater than -.313 and

```
1  x0,x1
2  2,2
3  -0.313,0.608,0.134,0.719,0.816,-0.967,0.898,0.069
4  -0.051,-0.851,0.617,0.079,0.439,-0.842,-0.541,0.833
5  -0.572,0.125,0.151,0.486,0.306
6  -0.918,-0.621,-0.033,0.162,0.774
7  -0.016,-0.19,0.007,-0.201,0.215
8  -0.157,0.504,-0.742,0.351,0.377
```

Figure 4: An example input for the FISH-SAT program

the maximum value is less than .069. Once it verifies this, it well then do the same for the definition in line 4.

For example, in figure four, we are given a problem with 2 variables, 4 objective functions and the following file specifies 4 polynomials in lines 3-6. In lines 7 and 8 I specify the linear constraints.

For the last 4 lines (lines 5-8 in figure 4) I specify the constraints, the format for the constraints is:

1. Column 1: the minimum for this constraint
2. Column 2: the coefficient of the x_1 variable for this constraint
3. Column 3: the coefficient of the x_2 variable for this constraint
4. Column n: the coefficient of the x_n variable
5. Column n+1: the coefficient of the constant term in the constraint
6. Column n+2 (last number): the maximum for this constraint

3.3 Program Algorithm

The program reads the files, then finds the minimum (or maximum) for each polynomial under the given constraints. In order to transform the polynomials and constraints into Boolean expressions I use PySMT with the Z3 library. The algorithm proceeds as follows:

3.4 Example Results

To generate sample data I wrote a "tester" program that generates config files using random numbers in the unit interval for coefficients and constraints. By setting the offsets to the random number generation carefully, I was able to come up with random polynomials that had a pretty even distribution of "sat" and "unsat".

The algorithm is able to correctly identify solutions to quadratic programming problems with any number of expressions, variables and constraints. As the number of variables increases, the time complexity increases plots are provided to show the relationship. From the presence of the outliers, one can

Algorithm 1 QP to SAT

procedure QP2SAT($file$)
  $g^i \leftarrow g^i(x_1, ..., x_n)$ quadratic objective functions from file
  $c_j \leftarrow And([And(GE(Real(Z), Plus(Times(Real(A1), Y1), Times(Real(A2), Y2)),$
    $LT(Real(X), Plus(Times(Real(B1), Y1), Times(Real(B2), Y2))))$
    for X, Y, Z in zip(mins, variables,maxes)])
  for $i = 1..num\_functions$ **do**
    for $j = 1...num\_terms$ **do**
      $eqns[i] \leftarrow Plus(eqns[i], Times(Real($ coefficient of jth term) , jth term))
    print minimum for $eqns[i]$
  if minimum over all $g^i > grand\_min$ **then**
    return true
  return false

infer that the time complexity is more than just a function of the number of variables, constraints, and expressions. Substantial experimentation in multiple different toolboxes has yet to reveal why some polynomials are hard to optimize. I have provided some "hard" polynomials "longinput1.csv" and "longinput2.csv" on the github for this project. Further investigation would be needed to debug and isolate the issue. I did investigate whether or not there was some problem with the Hessian, but I do not see any issue with the Hessian. I am randomly generating the polynomials and the constraints, so it is possible I have found a bug in the z3 sat software suite. I have filed a bug here: `https://github.com/pysmt/pysmt/issues/648`. The problem becomes intractible write around the point that I start having problems with 4 variables and 4 constraints. I found that there was nothing particularly unusual about the Hessian except that it consistently is such that half the eigenvalues are negative and the other half are positive, but I can find cases where pySMT solves the problem quickly and this is true.

Performance results for several different configurations and fish models are found in results.csv on the github. Inputs were tested for 1, 2, 3, 4 variabes; 1,2,3,4,5,6,7,8,9,10 constraints, and 1,2,3,4,5,6,7,8,9,10 objective functions. The results give examples of yes and no instances of the problem. Notice that as the number of objective functions increases, the liklihood that we "randomly" select a polynomial whose hessian gives an np-hard problem increases. Hence, for large number of expressions and large numbers of variables, the true performance of the sat solver comes to light. The results file shows which configurations (csv files) were satisfiable, and which were not, it also provides the amount of time it took to find the satisfiability of the given configuration.

Excluding cases where the program froze (assuming these are because of a bug), if I were to estimate, I would say that the time complexity as a function of the number of variables is exponential for the general satisifiability problem (where not constraints are placed on the eigenvalues of the hessian matrix). As seen in table for some 3-variable problems, pysmt takes a considerable amount

of time to solve. Furthermore, we also see that the number of constraints affects the amount of time it takes to process the input. As the number of constraints increases with 3 variables, the amount of time seems to increase at least quadratically, if not exponentially. When plotting the time complexity as a function of the number of constraints, we see there are cases where 3 variables and 10 constraints takes longer than 4 variables and 10 constraints, so some polynomials are definitely harder to solve than others, and so are some constraints. When plotting the results as a function of the number of expressions, I find that as the number of expressions increases, the likelihood that there is a substantial increase in the amount of time to solve the quadratic programming problem increases. This is because it becomes more and more likely that a polynomial is optimized that has several negative and positive eigenvalues. In every case that the program hangs, I have found that at least half the eigenvalues of the problematic polynomial are negative.

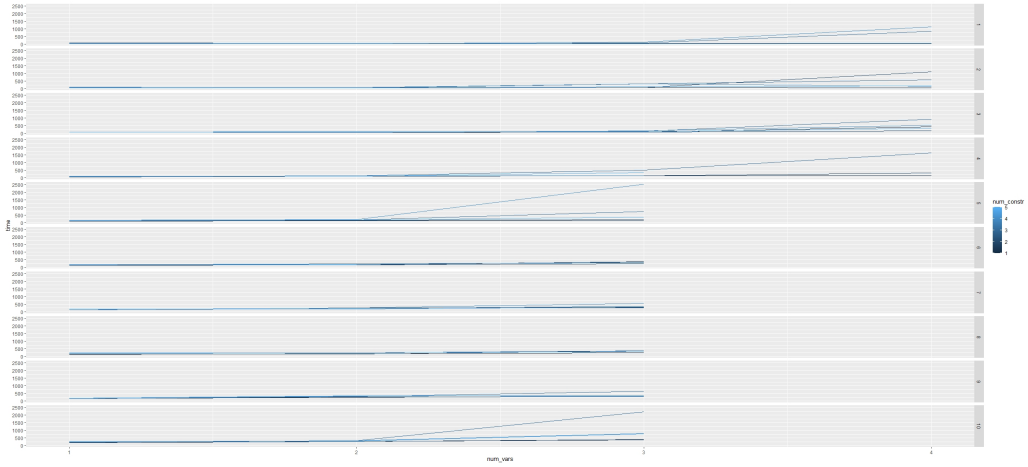

Figure 5: A plot of time as a function of the number of variables. Each row is a different number of expressions (from 1 to 10), the shade of blue gives the number of constraints. Notice that for a large number of constraints (light blue) and a large number variables (x-axis), the amount of time to solve is large for some expressions.

3.5 Conclusion

The Fish-SAT program can take any number of polynomials of degree 2, any number of variables, and any number of linear constraints and determine if

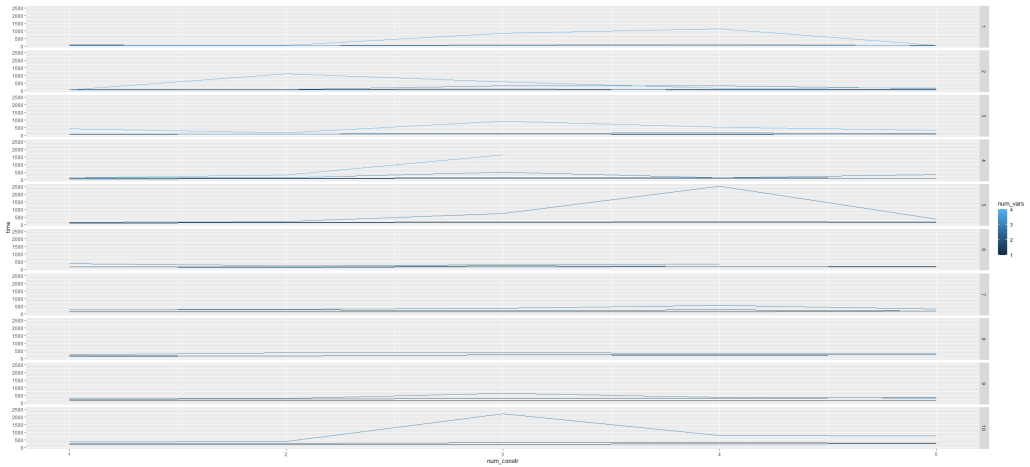

Figure 6: A plot of time as a function of the number of constraints. Each row is a different number of expressions (from 1 to 10), the shade of blue gives the number of variables. Notice that for 3 or more constraints (x-axis) and for 3 or more variables (light blue) we have that the amount of time to solve the QP is large. But it is only large when the randomly generated polynomial has a Hessian with negative eigenvalues (represented by the peak points).

there is a solution. I was able to find many configurations which pySMT failed to solve in a reasonable amount of time. I file a bug on this on the github because it seemed odd (see: `https://github.com/pysmt/pysmt/issues/648`). Several plots are provided, showing that the time complexity is a function of the number of variables and the number of constraints as well as the "hardness" of the hessian of the objective function. For objective functions with lots of negative eigenvalues, lots of variables and a large number of constraints, the pySMT program takes a over 10,000 ms to solve.

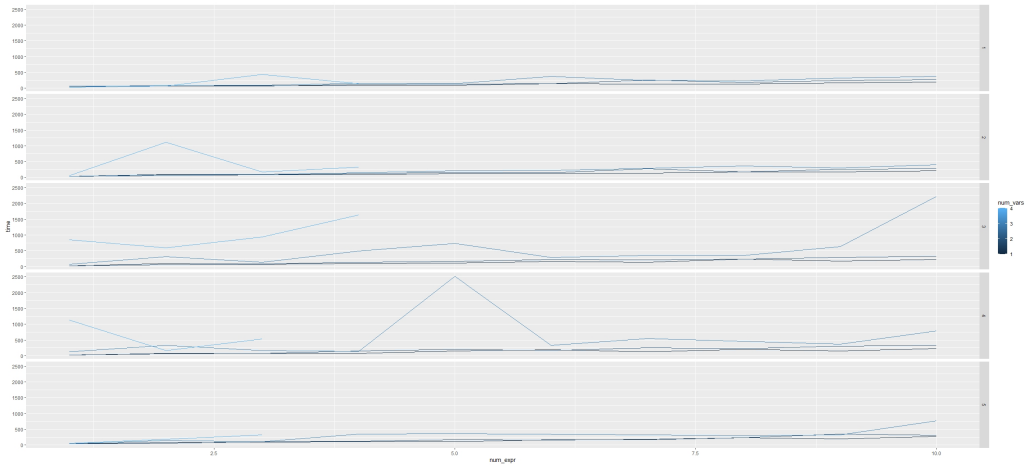

Figure 7: A plot of time as a function of the number of expressions. Each row is a different number of constraints (from 1 to 5), the shade of blue gives the number of variables. Notice that for 3 constraints and 4 variables, we have that the amount of time to solve the QP is large (light blue, 3rd row). Also of note here is that for 2 constrains (second row) and 4 variables, there are cases where the QP problem takes a long time to solve.