An Algorithm for the Classification of the Facets of Constraint Polytopes

Automated Facet Discovery Tool

Adam DeJans Jr.

Advisor: Dr. Serge Kruk

April 11, 2018

Oakland University - Department of Mathematics & Statistics

Table of contents

1. Introduction
2. Projection
3. Clustering
4. Coefficient Function Identification
5. Results & Implementation

Introduction

Introduction

In computer science, *constraint programming* is a programming paradigm wherein relations between variables are stated in the form of constraints. Constraint modeling languages feature declarative specification of the problem, separating model formulation, from data and search strategy.

Introduction

Constraint programming is a rapidly maturing technology, with several commercial tools now available. The techniques of constraint programming are applicable to a wide range of optimization problems including scheduling and resource allocation problems, and there are already a number of successfully implemented applications.

Motivation for Work

- Constraint programming is expressive
- Integer programming is fast

A Constraint Programming Model of the Sudoku Puzzle

```
matrix = Matrix(9,9)

sudoku =
      Model(
          [AllDiff(row) for row in matrix.row], #row constraint

          [AllDiff(col) for col in matrix.col], #column constraint

          [AllDiff(matrix[x:x+3, y:y+3].flat) #3x3 block constraint
              for x in range(0,9,3)
              for y in range(0,9,3)]
          )
```

Note: *.flat* is a 1-D iterator over the array.

An Integer Programming Model of the Sudoku Puzzle

$$\sum_{i=1}^{9} x_{ijk} = 1 \text{ for } j,k = 1..9 \quad \text{(columns)}$$

$$\sum_{j=1}^{9} x_{ijk} = 1 \text{ for } i,k = 1..9 \quad \text{(rows)}$$

$$\sum_{k=1}^{n} x_{ijk} = 1 \text{ for } i,j = 1..9 \quad \text{(all filled)}$$

$$\sum_{j=3p-2}^{3p} \sum_{i=3q-2}^{3q} x_{ijk} = 1 \text{ for } k = 1..9 \text{ and } p,q = 1..3 \quad \text{(3x3 grids)}$$

$$x_{ijk} = 1 \, \forall (i,j,k) \in G \;\cong\; \text{all the known cells}$$

Assignment in Constraint Programming

```
set of int: People = 1..n;
set of int: Tasks 1..m;
array[People, Tasks] of int: cost;
array[People] of var Tasks: task;
constraint alldifferent(task);
solve minimize sum(w in People)(cost[w,task[w]]);
```

Assignment in Integer Programming

```
set People;
set Tasks;
param c {People, Tasks} > 0;
var X {People, Tasks} binary;
minimize Z: sum {p in People} sum {q in Tasks}
    c[p,q] * X[p,q];
subject to P1 {p in People}:
    sum {q in Tasks} X[p,q] = 1;
subject to Q1 {q in Tasks}:
    sum {p in People} X[p,q] <= 1;
```

Contrast

- CP shorter (orders of magnitude)
- CP “natural” (300++ global constraints)
- IP requires “tricks” (and tricks vary with solvers)

Integer Programming Usually Solves Faster

Consider a library from Google with both CP and IP;[1]

- CP solver time: 475 miliseconds
- IP solver time: 20 miliseconds

[1]Previous Assignment problem

Our Goal

Given a constraint problem, find the mathematical representation of the the facets of the convex-hull representing the constraints.[2]

That is, automatically translate CP models into efficient IP models.

[2] Facet-defining inequalities

Why?

The major strength of Constraint Programs is how easily expressive they are, while the advantage of Integer Programs is obtaining bounds via linear programming relaxations.

General Overview of Steps

1. Write a generator of instances of combinatorial problems
2. Have a pre-processor generate a set of linear inequalities
    - Generates some valid IP formulation with additional variables
3. Cluster structurally identical constraints
4. Generate functions representing facets based on original parameters. That is, identify coefficient parameterized from initial data.

Current Status[3]

- IP Formulation is entirely done by hand
- Generation and projection is done via Fourier-Motzkin (Medcoff)
- Classification (DeJans)
  - Currently being tested with original clustering algorithm and DBScan.
  - Python prototype and beggining of LISP implementation
- Identification done via non-linear least squares (DeJans/Sibu)
  - Python prototype

[3]*This is a multi-year project, with multiple students.

Current Focus: Classification

- For each problem instance i (dependent on parameters $p_1, p_2, \dots, p_k$), say there is a set of S_i inequalities.
- In each of these S_i, the inequalities can be partitioned into structurally identical subsets (data mining part).
- Taking the whole universe of sets S_i, we wish to infer the dependence of each partition of the inequalities to the parameters $(p_1, \dots, p_k)$ (AI part).

Example: *all-different*

$$all - different(x_1, \ldots, x_n) \iff x_i \neq x_j$$

Valid IP Formulation

$x_i \in [0, l]$	original
$y_{ij} \in \{0, 1\}$	additional
$x_i = \sum_j j y_{ij}$	assign value
$\sum_j y_{ij} = 1$	exactly one value
$\sum_i y_{ij} \leq 1$	no two on same value

Example: *at-least*

$$at - least_m(x_1, \ldots, x_n) = k,\ x_i \in [0, l]$$

At least m variables out of n take on value k.

Valid IP Formulation

$x_i \in [0, l]$	original
$y_{ij} \in \{0, 1\}$	additional
$x_i = \sum_j j y_{ij}$	assign value
$\sum_j y_{ij} = 1$	exactly one value
$\sum_i y_{ik} \geq m$	at least m have value k

Projection

Generate Instances

$$at - least_m(x_1, \ldots, x_n) = k,\ x_i \in [0, l]$$

- n original variables
- nl additional variables
- $n + l$ constraints

Projection

- n variables
- ? constraints

Part of what makes our task so difficult is the symmetric nature of constraint problems and the *usual* exponential explosion of constraints.

Note on Symmetric Nature

We can see both graphically and algebraically that the polytopes from constraint programming are very symmetric. For the *At-Least Constraint* this happens since the subsets of equal cardinality are structurally equivalent.

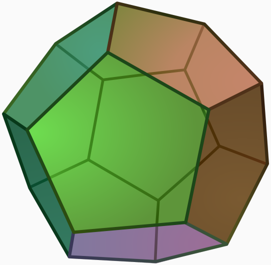


Figure 1: A *convex polytope* is a special case of a polytope, having the additional property that it is also a convex set of points in the n-dimensional space R^n

Fourier-Motzkin Elimination

Also known as the FME method, Fourier-Motzkin elimination is a mathematical algorithm for eliminating variables from a system of linear inequalities.

This method is currently used with Cernikov's rules, but we are moving towards a parameterized LP due to the exponential explosion of inequalities created using FME.

How to project efficiently?

Current contenders:

- Fourier-Motzkins with herustics to discard redundancies (90%)
- Double description method

Our situation is particular

- We project onto a very small subset of variables (from nl down to n).
- The elminated variables are all binaries.

Small Example

Consider

$$x_1 + x_2 + 2x_3 + 5 \geq 0$$

$$2x_1 - x_2 - x_3 - 4 \geq 0$$

Say we want to project onto (x_1, x_2)

Inspired by Farkas lemma

Consider $\lambda_i \geq 0$ and

$$\lambda_1(x_1 + x_2 + 2x_3 + 5) + \lambda_2(2x_1 - x_2 - x_3 - 4) \geq 0$$

is a valid inequality.

Rearrange the coefficients

To isolate x_3

$$x_1(\lambda_1 + 2\lambda_2) + x_2(\lambda_1 - \lambda_2) + x_3(2\lambda_1 - \lambda_2) \geq -5\lambda_1 + \lambda_2$$

Choose λ_i such that $(2\lambda_1 - \lambda_2) = 0$.

More generally

From a set of constraint

$$< a_i, x > -b_i \geq 0, \quad i \in I$$

and a set of J variables to eliminate, we consider

$$\min(x, \lambda) := \lambda_0 + \sum_i \lambda_i (< a_i, x > -b_i)$$

subject to coefficients of variables to eliminate to be zero.

A parameterized LP.

Conjecture (to be proven)

Every solution to the parameterized LP corresponds to a facet of the projected polytope.

Contrast to Fourier-Motzkin

- Eliminate all variables at one go
- No exponential blow-up (except in some cases)
- Need to solve a parameterized LP

Clustering

What to Cluster?

We want to cluster inequality constraints that are structurally identical, in order to examine the data representing them (the coefficients of the variables), so that we can analyze the data and find a function that will give us a facet-defining inequality.

Note here that every cluster should lead to a, possibly non-unique, facet.

Process of Clustering

Each cluster of constraints may lead to a new facet of the desired polytopes. While any pair of clusters may be data representative from the same facet, we must be sure to <u>never</u> mis-classify data from two different facets.

The first attempt at clustering was to use traditional clustering algorithms, however all but one seemed to work with any given mapping of the data to the same space.

Example

$$at - least_2(x_1, \ldots, x_5) = 3;\ x_i \in [0, \ldots, 6]$$

This is an *at-least* constraint problem with input values parameters $(m, n, k, l) = (2, 5, 3, 6)$.

First Class of Constraints

$$x1 + x2 + x3 + x4 + x5 <= 24$$

$$-x1 - x2 - x3 - x4 - x5 <= -6$$

Second Class of Constraints

$$+x1 - x2 - x3 - x4 - x5 <= 0$$

$$-x1 + x2 - x3 - x4 - x5 <= 0$$

$$-x1 - x2 + x3 - x4 - x5 <= 0$$

$$-x1 - x2 - x3 + x4 - x5 <= 0$$

$$-x1 - x2 - x3 - x4 + x5 <= 0$$

Third Class of Constraints

$$x1 + x2 + x3 + x4 - x5 <= 18$$

$$-x1 + x2 + x3 + x4 + x5 <= 18$$

$$x1 - x2 + x3 + x4 + x5 <= 18$$

$$x1 + x2 - x3 + x4 + x5 <= 18$$

$$x1 + x2 + x3 - x4 + x5 <= 18$$

Fourth Class of Constraints

$$x1 + x2 - x3 - x4 - x5 <= 6$$

$$x1 - x2 + x3 - x4 - x5 <= 6$$

$$x1 - x2 - x3 + x4 - x5 <= 6$$

$$x1 - x2 - x3 - x4 + x5 <= 6$$

$$-x1 + x2 + x3 - x4 - x5 <= 6$$

$$-x1 + x2 - x3 + x4 - x5 <= 6$$

$$-x1 + x2 - x3 - x4 + x5 <= 6$$

$$-x1 - x2 + x3 + x4 - x5 <= 6$$

$$-x1 - x2 + x3 - x4 + x5 <= 6$$

$$-x1 - x2 - x3 + x4 + x5 <= 6$$

Fifth Class of Constraints

$$x1 + x2 + x3 - x4 - x5 <= 12$$

$$x1 + x2 - x3 + x4 - x5 <= 12$$

$$x1 - x2 + x3 + x4 - x5 <= 12$$

$$-x1 + x2 + x3 + x4 - x5 <= 12$$

$$x1 + x2 - x3 - x4 + x5 <= 12$$

$$x1 - x2 + x3 - x4 + x5 <= 12$$

$$-x1 + x2 + x3 - x4 + x5 <= 12$$

$$x1 - x2 - x3 + x4 + x5 <= 12$$

$$-x1 + x2 - x3 + x4 + x5 <= 12$$

$$-x1 - x2 + x3 + x4 + x5 <= 12$$

Sixth Class of Constraints

$$-x_1 <= 0$$

$$-x_2 <= 0$$

$$-x_3 <= 0$$

$$-x_4 <= 0$$

$$-x_5 <= 0$$

Seventh Class of Constraints

$$x_1 <= 6$$

$$x_2 <= 6$$

$$x_3 <= 6$$

$$x_4 <= 6$$

$$x_5 <= 6$$

Representative from each class

$$x_1 + x_2 + x_3 + x_4 + x_5 <= 24$$

$$x_1 + x_2 + x_3 + x_4 - x_5 <= 18$$

$$x_1 + x_2 + x_3 - x_4 - x_5 <= 12$$

$$x_1 + x_2 - x_3 - x_4 - x_5 <= 6$$

$$x_1 - x_2 - x_3 - x_4 - x_5 <= 0$$

$$-x_1 - x_2 - x_3 - x_4 - x_5 <= -6$$

$$-x_1 <= 0$$

$$x_1 <= 6$$

Another Example: At-Least (3,4,5,6) - Inequalities

$$5x_1 + 5x_2 + 5x_3 - x_4 <= 75$$

$$5x_1 + 5x_2 - x_3 - x_4 <= 45$$

$$x_1 + x_2 + x_3 + x_4 <= 21$$

$$x_1 - x_2 - x_3 + x_4 <= 15$$

$$x_1 - x_2 - x_3 - x_4 <= -15$$

Gathering Data

We gather large samples of data by collecting *many* inequalities using *many* combinations of parameters (many different instances of the *at-least* problem).

After we have our generated data, we need to cluster inequalities that are structurally similar so that we can generalize the facets for the constraint problem.

Keeping the final goal in mind

For an *All-different* constraint of dimension n and taking on values 0 to k we obtain two facets:

$$x_1 + x_2 + \cdots + x_n \geq \frac{n(n+1)}{2}$$

and

$$x_1 + x_2 + \cdots + x_n \leq \frac{n(2k-n+1)}{2}$$

which are the lower and upper bound, respectively.

Keeping the final goal in mind

By using many instances of different *at-least* problems we will have enough data to find the generalized forms for the facets of the *at-least* polytope. One such class of facets is for the generalized $at - least_m(x_1, \ldots, x_n) = k; x_i \in [0, \ldots, l]$ is:

$$k \sum_{i \in J} x_i + (k - l) \sum_{i \in I \setminus J} x_i \leq mk^2 + (n - m - |J|)lk$$

J: subset of n of every size

Next step to reaching our goal

After collecting a *large* amount of data, our next step is to encode the data for clustering; recall that at this point our data are a bunch of inequalities.

Encoding data for clustering

Since inequalities may all be from different dimensions we must encode the inequalities in such a way that they are in the same dimension so we can have a fair chance at clustering them.

First Attempt at Encoding Data

Create a tuple such that for every unique coefficient of the inequality we add *the coefficient* and *how many times it appears*.[4] In the case where there is an inequality that has fewer unique coefficients than other inequalities we append $0, 0$.

For example,

$$x_1 + x_2 + x_3 + x_4 <= 21 \implies (0, 0, 1, 4, 21, 1)$$

$$x_1 - x_2 - x_3 + x_4 <= 15 \implies (1, 2, -1, 2, 15, 1)$$

$$x_1 - x_2 - x_3 - x_4 <= -15 \implies (1, 1, -1, 3, -15, 1)$$

[4] This is a reasonable approach due to the symmetric nature of these polytopes.

For a given set of points in some space to be clustered DBScan will classify points as *core points*, *(density-)reachable points*, and *outliers*. Each set of core and reachable points from core points are considered to be in the same cluster.

- A point p is a *core* point if at least a specified number of points are within an ϵ distance of it. We call those points within an ϵ distance *directly reachable* from p.
- All points not reachable from any other point are *outliers*.

DBScan Visualization

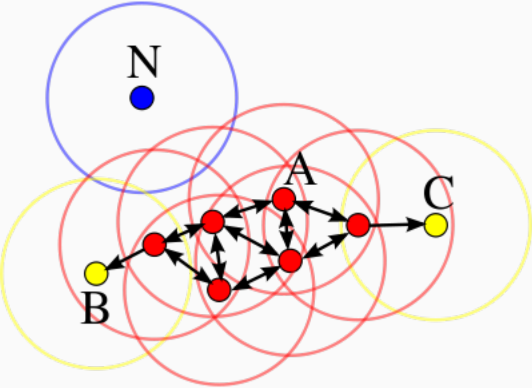


Figure 2: Minimum Points is 4. Only one cluster is present.[5]

[5] Wikipedia image

DBScan Unsuccessful?

This technique properly clustered data for the *All-Different*, and *At-Least* problems.

When tested with the *Multiple All-Different Predicates of Size* 2 *Arranged on a Cycle*, DBScan did not fail, in the sense of providing a *mis-cluster*, however, every point was deemed to be an outlier (or with different set parameters all the same cluster). Due to this experiment, it was decided that removing the spatial aspect would be the best approach.

Visualizing the constraints

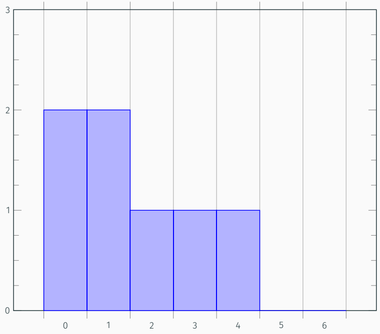
While clustering data by hand seems doable simply by looking at the constraints, it appears to be more challenging to automate the process.

This led to the idea of somehow encoding a visualization of the data. We arrived at the idea of visualizing the data as bar-like charts and encoding what we see.

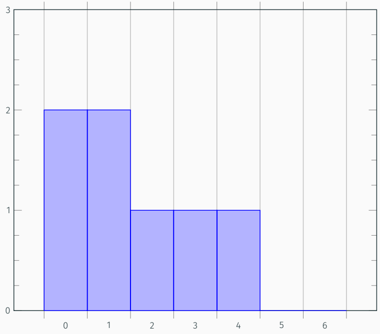
Pattern in Histograms

Histograms are able to take care of symmetric cases by rearranging inequalities in order from greatest to least coefficient values.[6]

$2x_1 + x_2 + x_3 + x_4 + 2x_5 + 0x_6 + 0x_7$



$x_1 + x_2 + 2x_3 + 0x_4 + 0x_5 + 2x_6 + x_7$



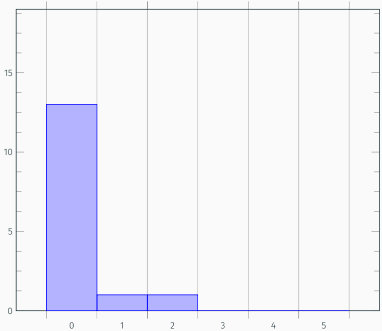
[6]Note: we only focus on the left-hand-side of the constraints first. After this is taken care of we will focus on the RHS separately.

Detecting Clusters with Histograms

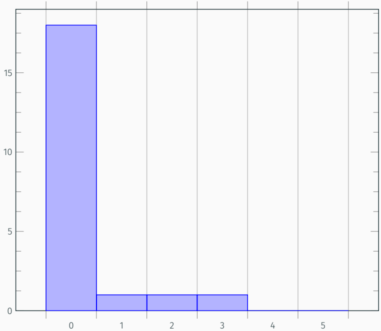
All-Different example with dimension 5 and $n = 7$.

Cluster 1:

$x_1 + x_2 + 0x_3 + 0x_4 + 0x_5 \leq 13$



$x_1 + x_2 + x_3 + 0x_4 + 0x_5 \leq 18$

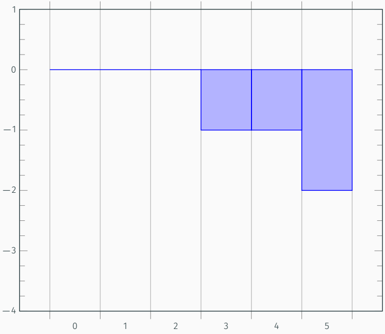


Detecting Clusters with Histograms

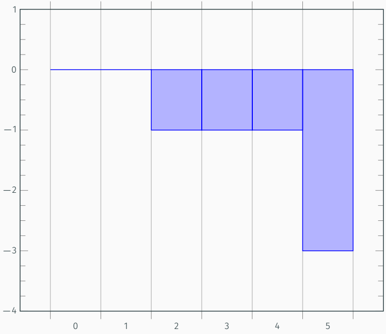
All-Different example with dimension 5 and $n = 7$.

Cluster 2:

$x_1 + x_2 + 0x_3 + 0x_4 + 0x_5 \geq 2$



$x_1 + x_2 + x_3 + 0x_4 + 0x_5 \geq 3$



Encoding for Clusters

From the examples, we can see that visually the graphs can be clustered together. After having many failures with spatial-based clustering we desire to encode or cluster in such a way that doesn't rely on points in space, but purely on the visuals.

We can remove the spatial aspect, and preserve the *shape/structure* of the bar-graphs, by encoding each inequality as a 3-*tuple*: (A,B,C), where

- A is the number of positive coefficient changes
- B is 0 or 1 representing whether or not a coefficient of 0 is present
- C is the number of negative coefficient changes

Process of Clustering

Given a cluster of constraints (inequalities), we may have many different non-zero coefficients to solve for. By the chosen encoding, we are guaranteed that there must be the same number of different coefficients in any constraint (inequality) in any given cluster.

For example, the two constraints

$$2x_1 + x_2 + 2x_3 + x_4 \leq 10 \mapsto (3, 0, 0)$$

and

$$4x_1 + 4x_2 + 3x_3 + 4x_4 + 3x_5 + 3x_6 + 4x_7 \leq 29 \mapsto (3, 0, 0)$$

will fall into the same cluster. In which case we see that the facet represented by the constraints in this cluster take the form

$$a \sum x_i + b \sum x_j \leq c,$$

where a, b, and c are functions of the input parameters $p_1, p_2, \ldots, p_k$.

A Subtle Difficulty

When deciding which data to use to solve for the function a in $a \sum x_i + b \sum x_j \leq c$, we *may* have to try several combinations of the coefficient data given.

For example, if we consider the previous two constraints

$$2x_1 + x_2 + 2x_3 + x_4 \leq 10$$

and

$$4x_1 + 4x_2 + 3x_3 + 4x_4 + 3x_5 + 3x_6 + 4x_7 \leq 29,$$

it is unclear which combination of coefficients we should be using:

$$1 \text{ and } 3 \qquad \textit{or} \qquad 1 \text{ and } 4 \qquad \textit{or} \qquad 2 \text{ and } 3 \qquad \textit{or} \qquad 2 \text{ and } 4.$$

It's easy to see that as our number of different coefficients within a constraint increases, we have to try substantially more combinations per cluster.

Thus far, in most instances, we have had success using the most natural combinations; that is, we match negative coeffiecients with negative coefficients, positives with positives, and when there is more than one positive or negative then we match the coeffeficients in numerical order.

For example, in the last example considering

$$2x_1 + x_2 + 2x_3 + x_4 \leq 10$$

and

$$4x_1 + 4x_2 + 3x_3 + 4x_4 + 3x_5 + 3x_6 + 4x_7 \leq 29,$$

we begin with trying (1 and 3) and (2 and 4).

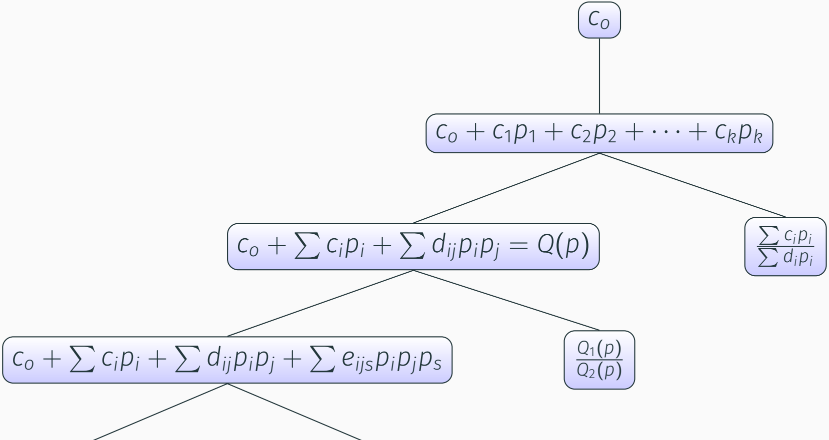
Defining Sucess

We define a successful function for the coefficients to be one that passes the following test: we generate more data and look for new data that falls into the same cluster as that which we just solved for; we test to see that the generated coefficient works for the newly clustered piece of data; if the function doesn't match the data then we know to try a new combination.

Thus far, in most instances, we have had success using the most natural combinations.

Coefficient Function Identification

For each coefficient desired, begin with the assumption that the function is a constant c_o. If this fails then assume that the function is a polynomial of one degree higher. If this fails we branch into the case where we assume a polynomial of the next degree, and the case where the function may be a rational of two polynomials of the current degree.



Function Identifier

Assuming that each facet is a polynomial or a rational of, we can use each cluster to formulate a facet[7]. Using data from the clusters we perform a least-squares optimization problem to solve for our desired functions.

[7] Clusters may not be unique facets

Non-linear Least-Squares Optimization Insight

Identification of coefficients is done via non-linear least-squares.

From numerous samples we have gained the insight that when generating data we must not use parameters that are close in value with each other as this may not produce enough necessary (useful) information needed to solve the least-squares problem.

At-least Deficiency of Data

For example, when generating data for the *at-least* problem, one cluster had over 129 constraints, but the rank of matrix A in minimizing $Ax = b$ was only 20 under the assumption that the facet was a polynomial of degree 2. In the *at-least* problem there are four given inputs (m, n, k, l), essentially we took all data generated from $(1, 1, 1, 1)$ to $(5, 5, 5, 5)$ and lacked sufficiency.

This tells us that when generating data, we need to use *very different* parameters.

The setup

Input:

- $(p_1, p_2, \ldots, p_k)$ (input parameters)
- a (input coefficient)

We want $f(p_1, p_2, \ldots, p_k) = a$.

Assume $f(p_1, p_2, \ldots, p_k) = c_o + \sum_i c_i p_i + \sum_{ij} c_{ij} p_{ij}$.

We want to minimize $\|Pc - a\|$.

The setup

Assuming a degree 2 polynomial,

$$\begin{bmatrix} 1 & p_1^1 & p_2^1 & \dots & p_k^1 & p_1^1 p_1^1 & \dots & p_k^1 p_k^1 \\ 1 & \dots & & & & & & \\ 1 & & \ddots & & & & & \\ \vdots & & & & & & & \\ 1 & \dots & & & & & & p_k^l p_k^l \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \\ \vdots \\ c_k \\ c_{11} \\ c_{12} \\ \vdots \\ c_{kk} \end{bmatrix} = \begin{bmatrix} a^1 \\ a^2 \\ \vdots \\ \vdots \\ \vdots \\ a^l \end{bmatrix}$$

$$Pc = a$$

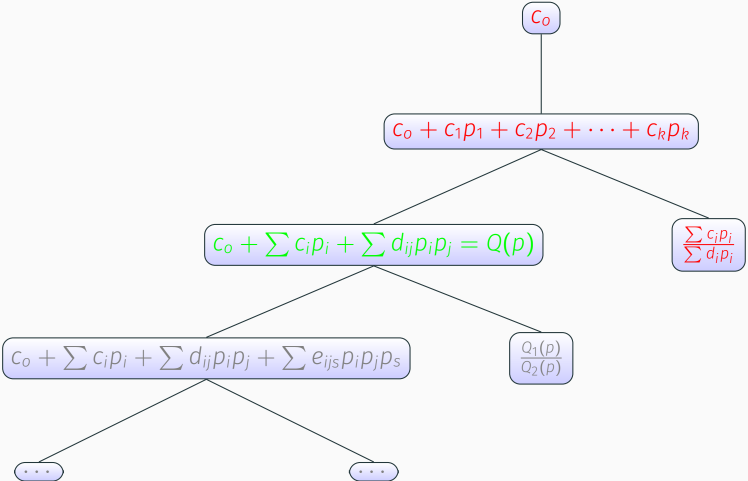
Input: $(p^1 \dots p^l)$, a, and d (the order of polynomial)

Output: c

After potential function found

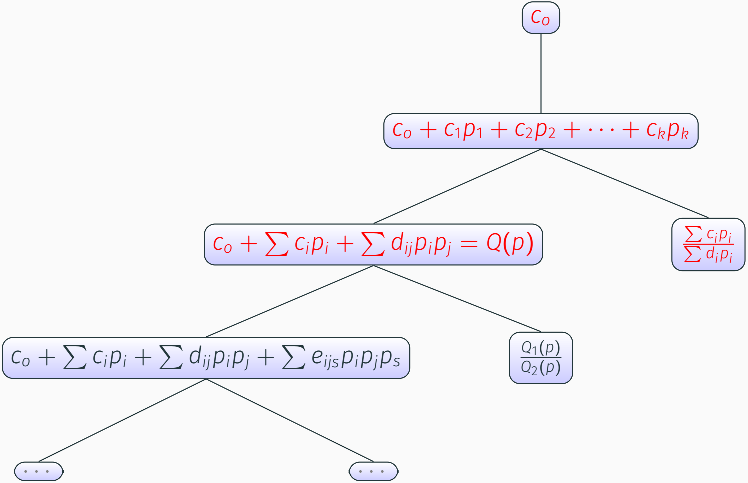
After obtaining a result (which always happens), we test the function with a new piece of data that falls within the same cluster.

If the discovered function satisfies the data then we stop, and accept the function.



After potential function found

If the function contradicts a new piece of data within the same cluster, then we branch off and repeat by assuming that the function is a rational of the same degree polynomials, and also by increasing the degree of the polynomial by 1.



Results & Implementation

Results to Date

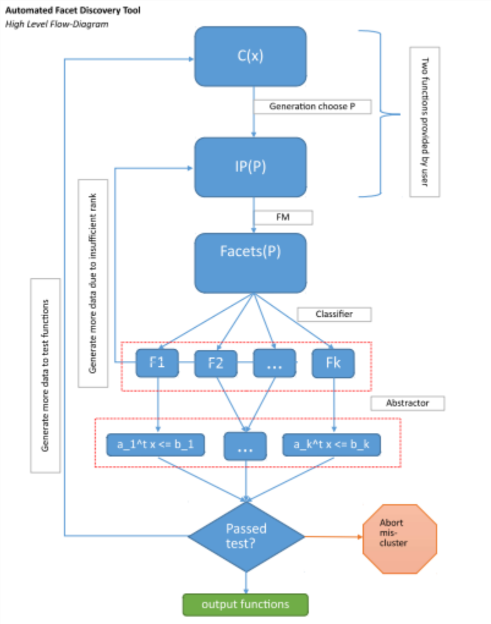
This method of finding the mathematical formulation of constraint polytopes has been successful for two known constraint problems: the *All-Different Predicate* and the *Multiple All-Different Predicates of Size* 2 *Arranged on a Cycle*.

This method has also worked well to obtain facets of the currently unpublished *At-Least Predicate*.

Properties of Library

- Written in Lisp
- Symbolic manipulation as much as possible
- Numeric computations over rationals
- On Github soon
- Includes:
    - Fourier-Motzkin
    - Simplex
    - Least-Squares
    - Classifier

Appended: Flow Diagram



The End

End of presentation.

References

- Study of the Polytope of the At-Least Predicate (*Kaso, Kruk*)
- Facets of Multiple All-Different Predicates of Size 2 Arranged in a Cycle (*Hayman, Kruk, Liptak*)