University Of Liège

MATH0462 - Discrete optimization

Box search for data mining using heuristics and MIP

Authors:
BERNARD Aurélien, OZDEMIR Kenan

1 Introduction

This project focuses on box searches for data mining which is essentially a process retrieving relevant variables from a data set with regard to an output variable. The formal definition of such a problem is given in the project's statement. Note that we assume the data sets to be correctly encoded. Indeed, we noticed that the given data set "DataProjetExport.csv" contains an erroneous data point at line 591 which has been retrieved in all the performed tests.

2 Domain partitioning

To make this problem a discrete optimisation problem the continuous domain had to be first normalised and then discretised. Every point was then normalised with respect to these values in each dimension to be in the range $[0; 1]$ determined by the largest and smallest value of each dimension.

In order to discretise the domain our approach consisted of first sorting the values in each dimension. Then, the sorted values were used as the limits of each interval. In our implementation the interval ends above the value of the point. Meaning that for two sorted points p_i and p_{i+1}, the interval will be defined as $Interval_i =]p_i, p_{i+1}]$. As for the last interval, the distance between the last point and 1 is used to define it.The length of each interval was recorded in a matrix of size of $(D \times Nb\_elements + 1)$.

In this discretisation there are two main flaws. The first being that due to points having the same values in the same dimension, there are intervals that have a distance of 0. This adds useless elements to the total domain of possible solutions, adding some overhead. However, this does not compromise correctness of the algorithms. The second flaw is that with our discretisation of the domain we accept that there will be an error on the length of the box in the interval shown in equation 1.

$$\text{Total error} \in [0, \sum_{n=1}^{D} \max_{i \in nb\_intervals} (interval_{n,i})] \tag{1}$$

To reduce these error segments, we could have implemented our solution in a more complex manner. However, for questions 1 and 2 the error made can be detected and compensated after execution of the algorithms. Furthermore, for this application the general location of the box seemed more relevant than its exact size. For this reason in addition to a lack of time, it was decided to allow these errors and keep this segmentation.

3 Collision detection

In order to detect collisions and model the location of each point of the set, a matrix for each point was defined. These matrices p_n are of size $(D \times nb\_intervals)$. In these matrices the values are binary and represent whether the value of p_n in dimension i is within interval j or not.

The matrix to represent the box is represented in a similar manner. It is a binary matrix of same size where each element represents if the segment is used in a side of the box or not. The '1' values should always be contiguous within each dimension as the sides cannot be fragmented.

The collision of a point with the box are readily obtained when these two matrices are used. There will be a collision with a point if for every position where $p_{n,i,j} = 1$, $box_{i,j} = 1$ too.

4 Questions

4.1 Q1: MIP model to find largest box $B \cap X = \varnothing$.

To execute our solution, the following command should be entered:

```
$ julia question1.jl csv_filename threshold
```

Where *csv_filename* is the path to the data set and *threshold* dedicated to the output variable. Let S = D+1 be the number of intervals. The MIP formulation for this problem can be seen in the following equations:

$$\max \sum_{i=1}^{D} \quad \sum_{j=1}^{S} \big(Lower_{(i,j)} - Upper_{(i,j)}\big) * Interval\_length_{(i,j)} \tag{2}$$

Subject to :

$$Lower_{(i,j)} \geq Upper_{(i,j)}, \forall i, j \tag{3}$$

$$Lower_{(i,j)} \leq Lower_{(i,j+1)}, \forall i, j \tag{4}$$

$$Upper_{(i,j)} \leq Upper_{(i,j+1)}, \forall i, j \tag{5}$$

$$\forall x_p \in X, \sum_{i=1}^{D} \sum_{j=1}^{S} occupancy_{(p,i,j)} * (Lower_{(i,j)} - Upper_{(i,j)}) \leq (D-1) \tag{6}$$

The vectors of binary values *Lower* and *Upper* are used to define the box. The box is defined by Lower - Upper. Constraints 3 to 5 included are used to correctly define the box. No segmentation of the sides are allowed this way. Constraint 6 is to forbid an overlap of any point of X with the box.

4.2 Q2:Heuristic approach to solve problem 1

As MIP solvers can be slow in certain instances and do not always scale well, heuristic approaches are useful to get solutions that are readily computed and that hopefully approximate the optimal solution. Our heuristic work in two steps.

The first step of the heuristic consists of choosing an initial box of the largest size possible through a greedy approach. This is done by choosing the segment of largest size in each dimension to define the box. Although it is highly unlikely, it is possible for this first box to contain a sample of X. If this is the case, we randomly generate boxes of side 1 segment per dimension until an empty box is found.

Then, the second step of the heuristic is applied. In this step the box is expanded in each dimension until no expansion is possible without a collision. To do so, we iterate over each dimension, choosing to expand "far" from the origin or "near" the origin. Meaning that we expand the box by adding the adjacent segment that is on the closest to the origin or the furthest away from it. Then, we check for collisions by comparing each point to the box. If a point is covered by all dimensions of the box, then the point is included in the box and the expansion is reverted. This process is repeated as long as there are sides that can be expanded with no collisions.

4.2.1 Complexity

The complexity of the first step of the heuristic is O(nb_segments · D). For each dimension each segment is checked for its length. The complexity of the second step varies with the number of expansions. Each expansion has a worst case complexity of O(nb_elements · (D · nb_segments)). For each element, the collision between the box and the element has to be checked segment by segment. However, this worst case scenario will rarely be reached as a full collision can be discarded as soon as there is not a collision in one dimension.

4.3 MIP model that finds a box B to max $\#(B \cap Y)$.

To execute our solution, the following command should be entered:

```
$ julia question3.jl csv_filename threshold
```

Where *csv_filename* is the path to the data set and *threshold* dedicated to the output variable.

$$\max \sum_{i=1}^{nb\_elements} z_i \qquad , z_i \in \{0,1\} \tag{7}$$

Subject to :

$$Lower_{(i,j)} \geq Upper_{(i,j)} \forall i,j \tag{8}$$

$$Lower_{(i,j)} \leq Lower_{(i,j+1)} \forall i,j \tag{9}$$

$$Upper_{(i,j)} \leq Upper_{(i,j+1)} \forall i,j \tag{10}$$

$$\forall x_p \in X, \sum_{i=1}^{D}\sum_{j=1}^{S} occupancy_{(x_p,i,j)} * (Lower_{(i,j)} - Upper_{(i,j)}) \leq (D-1) \tag{11}$$

$$\forall y_p \in Y,\ z_i * D \leq \sum_{i=1}^{D}\sum_{j=1}^{S} occupancy_{(y_p,i,j)} * (Lower_{(i,j)} - Upper_{(i,j)}) \leq (D-1) \tag{12}$$

As for section 4.1 constraints 8 to 12 are used for the same reasons. The new vector of binary values z_i contains a 1 for every elements of Y that is in the box. We constraint this using constrain 12. With this last constrain z_i will only be allowed to take the value 1 if a point y_i is in the box. The number of elements in the box will then be maximised by the objective function in 7

4.4 Q4: Heuristic approach to solver problem 2

To execute our solution, the following command should be entered:

```
$ julia question4.jl csv_filename threshold iter_limit
```

Where *csv_filename* is the path to the data set, *threshold* dedicated to the output variable, and *iter_limit* the maximum number of iterations.

Our heuristic for this question picks randomly a point from the data set Y that will be used as an *origin_point*. We also define *box* as a list containing all points of Y that represents the defined box search. Essentially, the algorithm keeps track of all points of the set Y that have to be explored through a list *to_explore* and executes the following:

1) Among all points available in *to_explore*, find the closest point to *origin_point* that will be called *closest_neighbour*.

2) Retrieve *closest_neighbour* from *to_explore* and add it to *box*

3) Among the points of *box*, find the minimum and maximum values at each dimension and verify whether any points from the set X are in those intervals. If it is the case, *closest_neighbour* is retrieved from *box*.

4) Repeat from step 1 until *to_explore* is empty or if the number of iterations succeeds the given limit *iter_limit*.