Improving a Constraint Programming Approach for Parameter Estimation

Bertrand Neveu, Martin de la Gorce, Gilles Trombettoni

LIGM Ecole des Ponts Paris Tech, France
LIRMM Université de Montpellier, France

SWIM , June 2016, Lyon

Plan

1. Parameter Estimation
2. Contributions
3. Experiments

Plan

1. Parameter Estimation
2. Contributions
3. Experiments

Parameter Estimation

- Consider a model defined by n parameters.
- Input: m observations
- Output : “all” the models, i.e. the parameter values that fit at least Q **observations**, within a given tolerance τ
- more precisely, one model fitting each maximal set of observations of cardinality $>= Q$

Application to computer vision: **shape detection**

- 2D: find the lines, the circles...
- 3D: find the planes, the spheres, the cylinders...

Didactic example: finding lines in an image

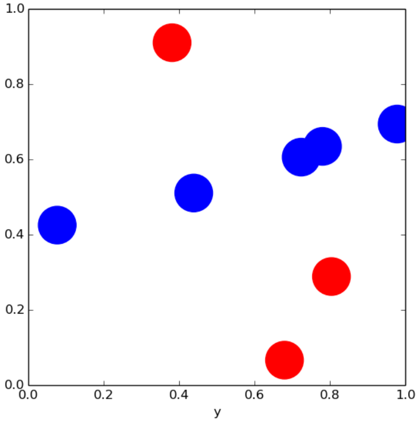
- m 2D points (x_i, y_i)
- Finding the lines defined by (**a**,**b**) with the equation $y = \mathbf{a}\, x + \mathbf{b}$
- such that each line contains at least Q points (x_i, y_i) (**inliers**), with a given tolerance τ :
  $|y_i - \mathbf{a}\, x_i - \mathbf{b}| < \tau$
- Only the lines with maximal sets of inliers (with at least Q points) are searched for.

Didactic example: a numeric CSP

The problem can be represented as a numeric CSP with continuous variables handled with intervals

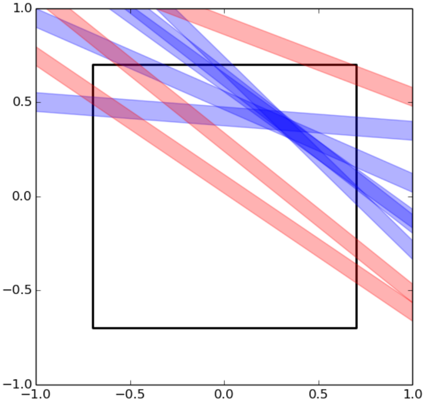
- **variables a**, **b**
- **domains** $[-1000, 1000], [-10, 10]$
- **constraints** : one constraint
  at least (Q, $|y_i - \mathbf{a}\, x_i - \mathbf{b}|$, i=1 .. m)
- **solutions** : small boxes

Didactic example: finding lines in an image



The data: the points in the image

Didactic example: finding lines in an image



The parameter space (a, b) for the lines

State of the art for parameter estimation : RANSAC

RANdom SAmple Consensus: **randomized** algorithm
Version for finding **all the models** (the lines)

1. Choose randomly 2 points, compute the corresponding line and compute a **consensus** for that line, i.e. checks that Q points belongs to the line (within the tolerance τ)

   ["Improve" the line with a better consensus (checking more points)]
2. If no line is found after some iterations: stop
3. Otherwise, store the solution.
4. Remove all the corresponding points

   Go to 1 to find another line

No guarantee to find **all** lines (all maximal sets of inliers).
Different runs give different sets of lines.

Plan

1. Parameter Estimation
2. Contributions
3. Experiments

Contributions

- Generic contributions
  - Algorithm based on a branch and prune scheme with **valid** and **possible** inliers
  - Q intersection in a **new direction**

- Specific contributions
  - A new efficient **parameterization** for lines and planes
  - A specific **bisection** (branching) heuristics

QInterEstim: Branch and Prune algorithm

A complete Branch and Prune algorithm in the continuous parameter space **QInterEstim**

- based on Luc Jaulin's interval parameter estimation tool
- using the Q-intersection
- with new features and improvements to make it efficient

QInterEstim algorithm

The continuous parameter space in bounded by a box in n dimensions.

Exhaustive tree search performed in that space.

The current box has :

- **possible inliers**: observations not discarded
- **valid inliers**: inliers guaranteeing a model

Pruning: Contraction and Q intersection reduce the set of possible inliers and discard the boxes with less than Q possible inliers

Stopping condition: (possible inliers $=$ valid inliers) or precision reached

Result: boxes containing a model with at least Q valid inliers, and small boxes possibly containing a model.

Postprocessing: boxes with maximal set of inliers.

QInterEstim Algorithm

Exhaustive parameter estimation algorithm based on Q-intersection :

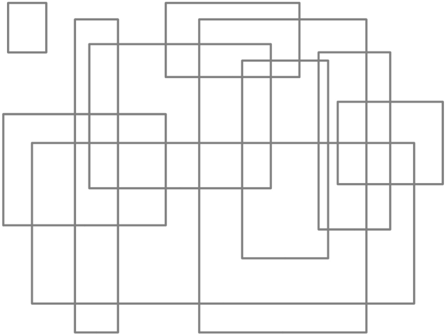
Algorithm `QinterEstim` *(box, observations, Q, ϵ_{sol}, τ)*

- *solutions* $\leftarrow \emptyset$; *node* $\leftarrow$ `new` *Node* ; *node.box* $\leftarrow$ *box*
- *node.possibleInliers* $\leftarrow$ *observations* ; *node.validInliers* $\leftarrow \emptyset$
- *nodeStack* $\leftarrow \{$*node*$\}$
- **while** *nodeStack* $\neq \emptyset$ **do**
  - *node* $\leftarrow$ `pop` (*nodeStack*) ; *box* $\leftarrow$ *node.box*
  - `contractAndQinter` (*box*, τ, *Q*, *node.possibleInliers*)
  - **if** *box* $\neq \emptyset$ **then**
    - `validateInliers` (*box*, τ, *node.possibleInliers*, *node.validInliers*)
    - **if** *width(box)* $< \epsilon_{sol}$ **or** *node.validInliers* $=$ *node.possibleInliers* **then**
      - *solutions* $\leftarrow$ *solutions* $\cup$ $\{$*node*$\}$
    - **else**
      - `bisect` (*box*, *box*$_1$, *box*$_2$) /* split the box */
      - `push` (*nodeStack*, "*box*$_1$") ; `push` (*nodeStack*, "*box*$_2$")
- **return** solutions

Q-intersection: principle

Definition

Let S be a set of boxes.
The Q-intersection of S is the box of **smallest perimeter** *that encloses the set of points of $\mathbb{R}^n$ belonging to* **at least Q boxes**.



Q-intersection: principle

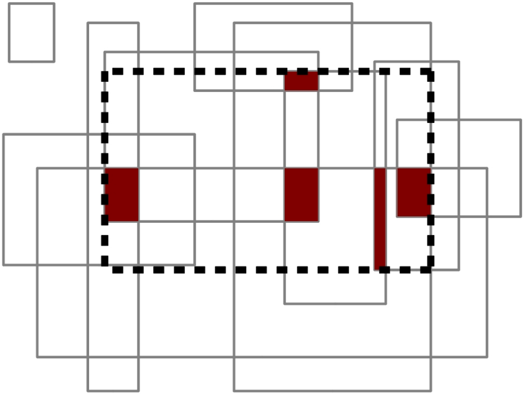
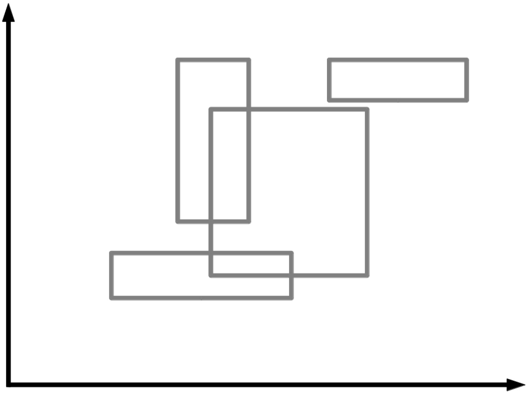


Illustration of Q-intersection for $Q = 4$, $n = 2$

Q-intersection: the Q-projection approximate algorithm

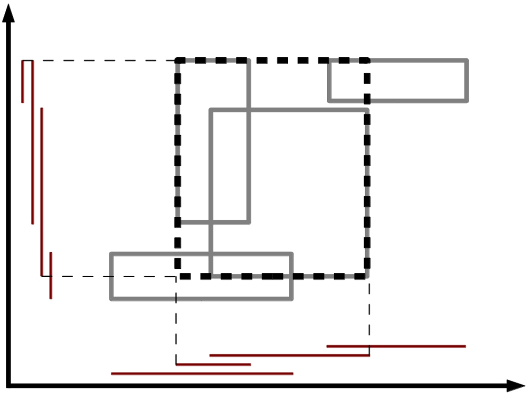


4 boxes in the current parameter space,
one for each possible observation.

Q-intersection: Algorithms

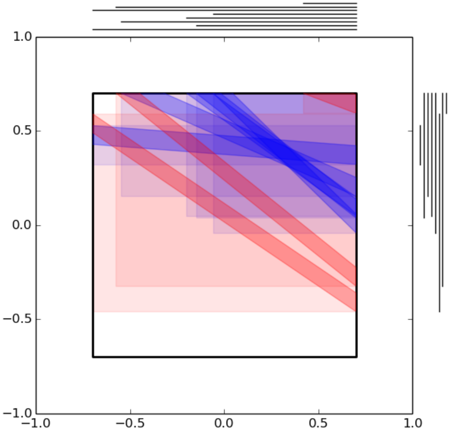
- an exact algorithm (Carbonnel et al. AAAI 2014) in $O(m^n)$
- an approximate algorithm (Jaulin et al) in $O(n \times m \times log(m))$ that projects the boxes on every dimension.

Q-intersection: the Q-projection algorithm



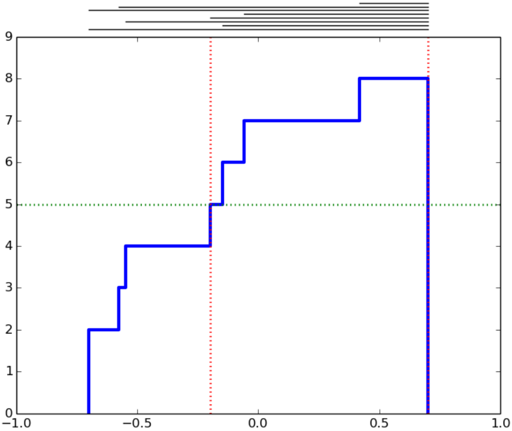
Projection on every parameter
Resulting box: 2-intersection of the 4 input boxes

Q-Intersection in the line example



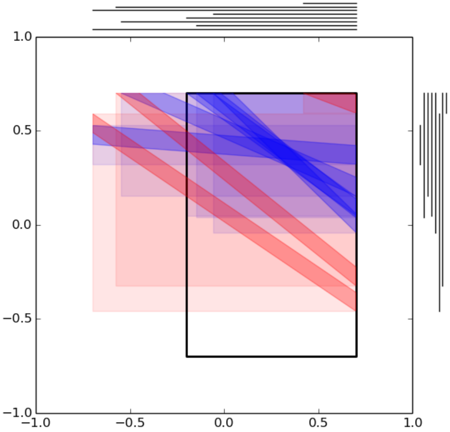
The boxes for each point in the parameter space
Current box $a = [-1, 1]$, $b = [-1, 1]$

Q-Intersection in the line example



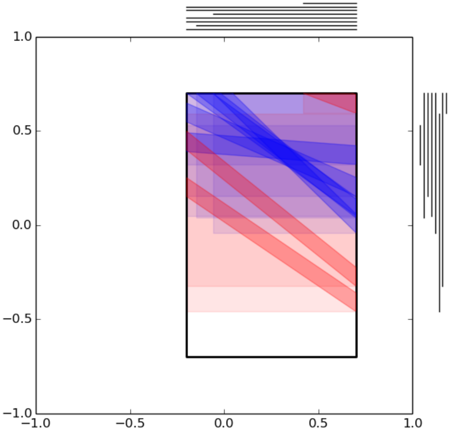
Q-projection on the *a* parameter with $Q = 5$

Q-Intersection in the line example



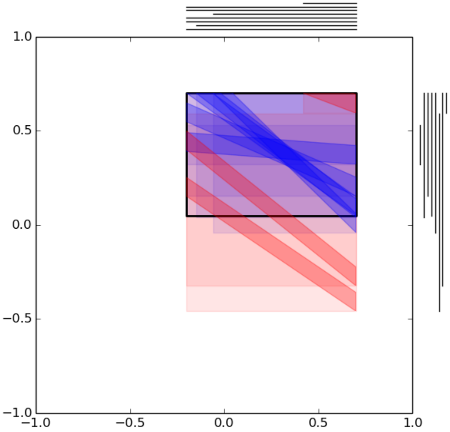
Contraction due to 5-projection on the *a* parameter

Q-Intersection in the line example



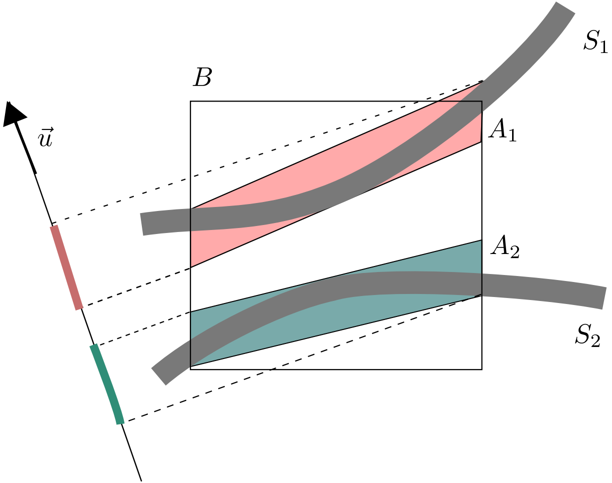
After the contraction due to 5-projection on a

Q- Intersection in the line example



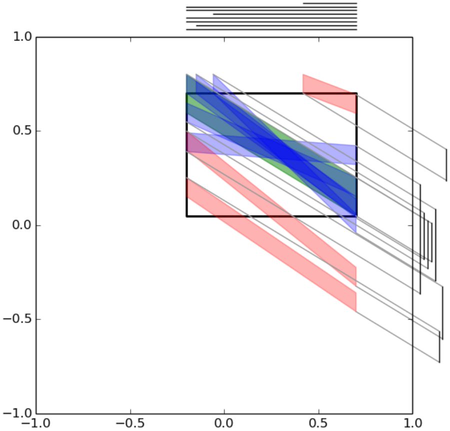
Contraction due to 5-projection on the *b* parameter

Q-Intersection in a new projection direction



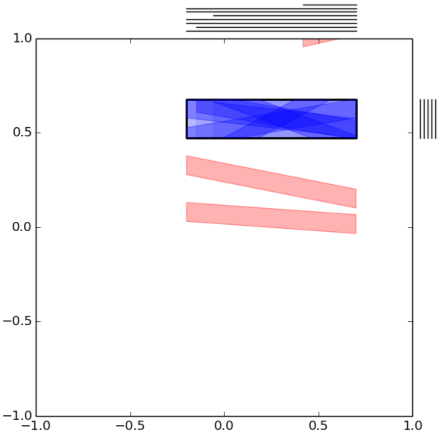
Projecting parallelograms A_i along the mean normal direction

The new projection direction in the line example



Q-projection on the mean direction

The new projection direction in the line example



Q-projection on the mean direction m in the (a, m) referential
The red points can be discarded

Specific improvements

Line and plane parameterization

- Classical model: $\mathbf{a}\,x + \mathbf{b}\,y + \mathbf{c}\,z + \mathbf{d} = 0$
  with $\mathbf{a}^2 + \mathbf{b}^2 + \mathbf{c}^2 = 1$
- Our linear model: $\mathbf{a}\,x + \mathbf{b}\,y + \mathbf{c}\,z + \mathbf{d} = 0$
  with $\mathbf{a} \pm \mathbf{b} \pm \mathbf{c} = 1$ (4 cases to study)

Branching heuristics

1. Round robin on **a**, **b**, **c**
2. When **[a]**, **[b]**, **[c]** are small, split **[d]**

Plan

Experiments : benchmark

- Plane detection: artificial test cases P_1 to P_9
- Plane detection in a 3D point cloud in a outdoor scene view
  points were labeled for a building : H_{40}
- Circle detection : a buoy in 2D images C_1 and C_2

Table : Characteristics of the artificial plane detection test cases

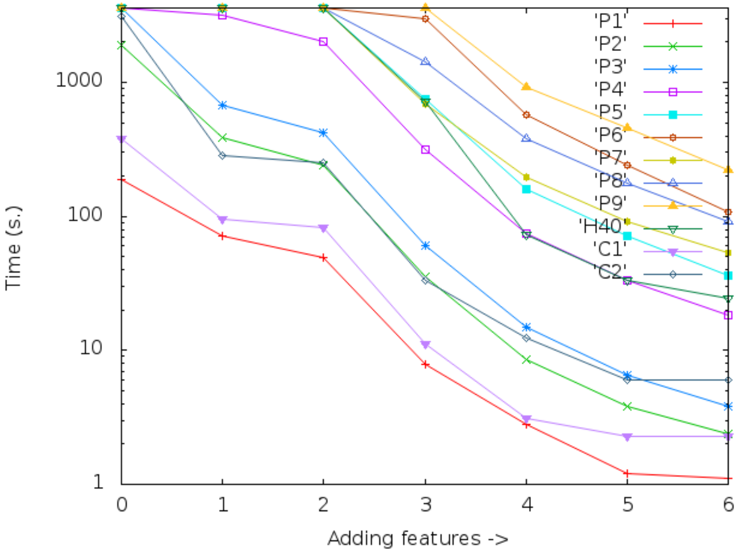
Test case	P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8	P_9
points	1000	1000	1000	1000	1000	1000	4000	4000	4000
planes	4	4	4	25	25	25	25	25	25
inlier rate	10%	5 %	4 %	2 %	1.5%	1%	2%	1.5%	1%

Experiments: results

Each improvement is added to the solving method

- 0: **initial algorithm**: basic implementation of Jaulin's Q-intersection based algorithm (without incremental maintain of possible inliers and without validations)
- 1: generic **QinterEstim** algorithm
- 2: **update** of possible observations after Q-projection
- 3: use of **dedicated** forward backward algorithm
- 4: Q-projection on the **new direction**
- 5: new bisection strategy
- 6: efficient plane parameterization

Experiments: results



Future work

- **Automatic selection** of the parameters Q and τ.
- **Optimization**: find the solution with the **maximum** number of inliers within a given tolerance.
- **Postprocessing** of the solutions: discriminate between the solutions with the maximal inliers sets.
- More experiments on **real scenes** for shape detection.
- Other problems in computer vision: computation of the **fundamental matrix**, **essential matrix** between two images.