Informing multi-objective optimisation benchmark construction through Instance Space Analysis

Estefania Yap, Mario Andrés Muñoz and Kate Smith-Miles *

1 Supplementary Materials

1.1 Algorithm Footprints

Table 1 shows the updated area and density for each algorithm's footprint upon the inclusion of newly generated instance. The updated footprints are shown in Figure 1a–1f.

Table 1: Area and density of each algorithm's footprint defining good performance on all instances.

Algorithm	Footprint Area	Footprint Density
GrEA	0.491	1.736
HypE	0.524	1.213
IBEA	0.735	1.298
MOEAD	0.451	0.822
NSGAII	0.388	1.161
SPEA2	0.403	1.098

1.2 Feature Predictive Power

Table 2 contains two comparison Random Forest in R – one for algorithm selection using the original 8 multi-objective features, and the other including all features – for the existing benchmark problems. 1298 total instances are used, where no tie-breakers were observed. The inclusion of feature is shown to reduce the out-of-bag (OOB) error rate, and thus provide evidence that the predictive power increases when including additional features.

*E. Yap, M.A. Muñoz and K. Smith-Miles are with the School of Mathematics and Statistics, The University of Melbourne, Parkville, VIC 3010, Australia.

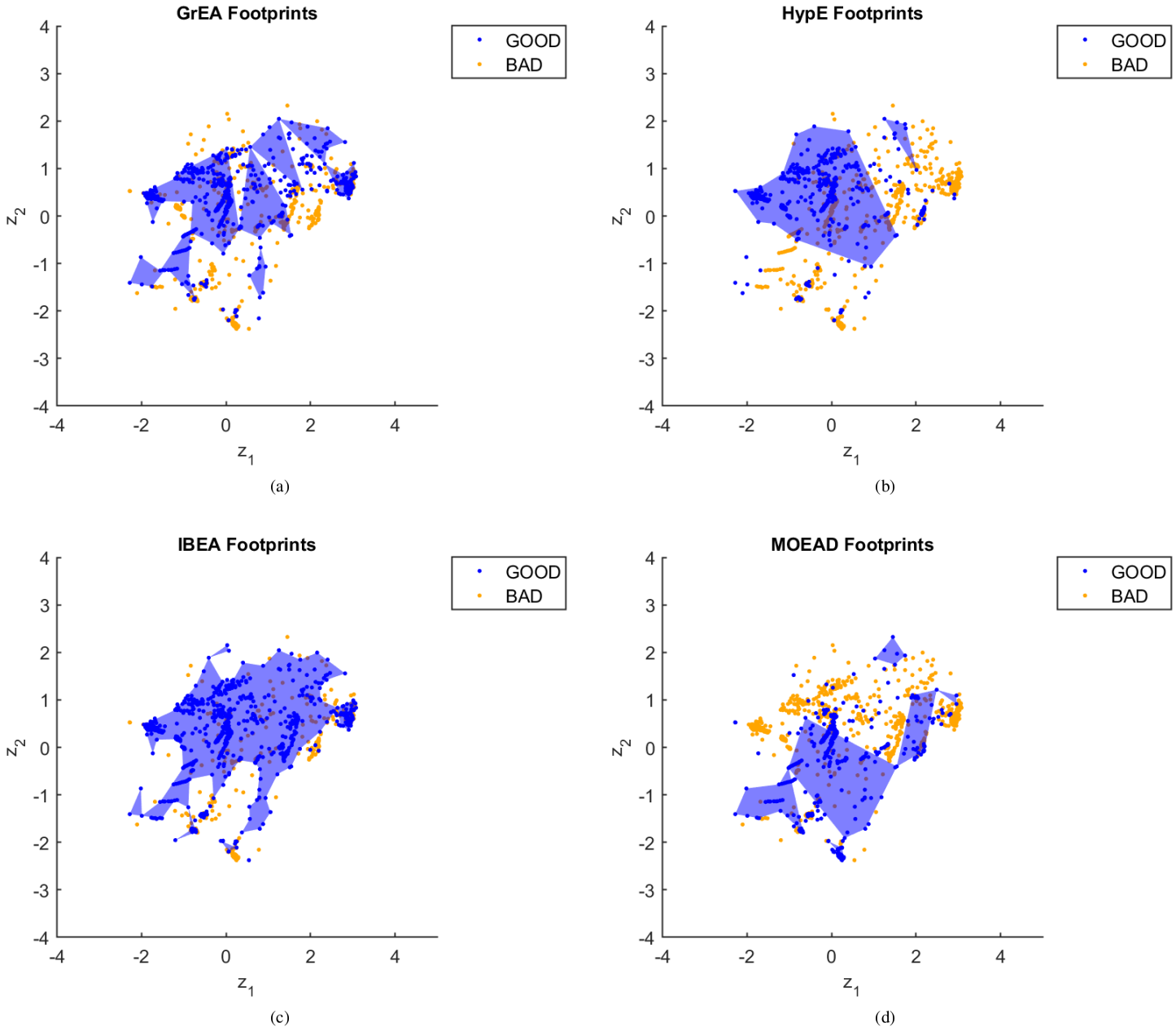

Figure 1: Algorithm Footprints obtained when using all benchmark problems and the performance obtained experimentally. An algorithm is good if it is within 1% percent of the best performing algorithm (cont.)

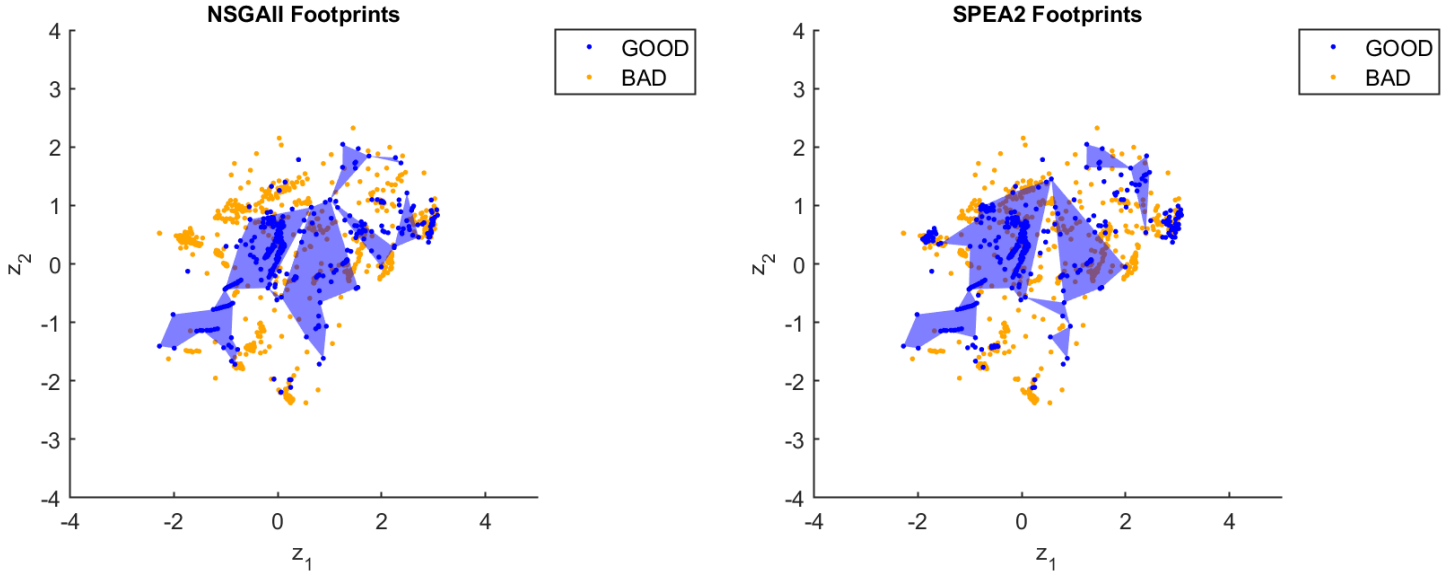

Figure 1: Algorithm Footprints obtained when using all benchmark problems and the performance obtained experimentally. An algorithm is good if it is within 1% percent of the best performing algorithm.

Table 2: Confusion matrix for Random Forest in R for all features versus only 8 existing MO features. OOB estimate of error rate for all features = 22.42%

Algorithm	GrEA	HypE	IBEA	MOEAD	NSGAII	SPEA2	Classification error
GrEA	184	30	26	5	7	3	0.278
HypE	11	246	39	5	0	1	0.185
IBEA	19	23	303	21	9	3	0.198
MOEAD	5	1	23	168	2	3	0.168
NSGAII	7	0	15	7	44	4	0.423
SPEA2	6	1	10	2	3	62	0.262
Confusion matrix for only 8 features. OOB estimate of error rate = 28.27%							
GrEA	166	36	26	13	9	5	0.349
HypE	13	240	40	8	1	0	0.205
IBEA	23	40	281	21	12	1	0.257
MOEAD	14	1	25	154	3	5	0.238
NSGAII	9	5	17	9	35	2	0.545
SPEA2	10	1	12	5	1	55	0.345

1.3 Instance Space Analysis Methodology

Following the collection of metadata, the instance space can be constructed using four algorithms: PRELIM, SIFTED, PILOT, and CLOISTER. Firstly, the PRELIM algorithm pre-processes the data: defining "good" algorithm performance from a user defined specification, removing outliers and normalising features.

In the following step, the SIFTED algorithm selects a subset of features which best summarises the key features of the instances that affect algorithm performance. This is achieved by calculating Pearson correlations between features f_x and algorithm performance $y_{\alpha,x}$. Features which share the highest correlations with algorithm performance are retained, and k-means clustering is used to group similar features together. To identify the best combination of features, one feature from within each cluster is taken, forming a feature subset, and this is repeated for all possible combinations. The optimal feature vector is selected which demonstrates the lowest predictive error when classifying whether an algorithm is good, based on a user-defined metric, upon projection onto $2D$ using Principal Component Analysis (PCA).

Once a predictive subset of features has been selected, the optimal projection into a $2D$ instance space can be generated from the feature space – using the PILOT algorithm – with the aim of simultaneously preserving visual trends in algorithm performance and feature distributions. This is achievable by ensuring that upon projection, instances are located in an orderly, linear fashion, with low values of features or algorithm performance metrics towards one edge, and high values towards the opposite edge. The output of this projection is a linear transformation matrix which maps each instance from its high-dimensional feature vector $\mathbf{f}$ into (Z_1, Z_2) coordinates in the $2D$ instance space. The algorithm CLOISTER then generates a theoretical boundary for the instance space, representing the area where, based on empirical evidence obtained from feature ranges in the meta-data, the superset of all possible instances is likely to be contained. This boundary is used to evaluate whether there are gaps in the existing test instance benchmarks, and the kinds of characteristics such unstudied instances

possess as revealed by their feature values.

At this stage, there is sufficient information to guide automated algorithm selection. An algorithm named PYTHIA is employed for this purpose. Since the instance space has been optimised for linear trends, it can then be partitioned into regions where each algorithm demonstrates strong performance using three algorithms: TRACE, Tracebuild and TRACEcomp. This information can be used to infer performance on untested instances using their location, as long as their feature vector f_x is known. A Support Vector Machine (SVM) is trained to recommend algorithms for each region with the highest accuracy.

Algorithms 1 – 4 relate to the construction of the instance space. Algorithms 5 – 8 relate to algorithm performance prediction and construction of footprints.

Input: A matrix $\in^{m \times n}$ of instance features, a matrix $\in^{a \times n}$ of performance measures, a performance threshold, ε, a binary flag *max* defining minimization or maximization, a binary flag *abs* defining absolute or relative performance, and the binary flags $\{bnd, nrm\}$ defining whether bounding and normalization are performed.

Output: A matrix $\in^{m \times n}$ of instance features, a matrix $\in^{a \times n}$ of performance measures, a matrix $\in^{a \times n}$ of binary performance measures, a vector $\in^{n}$ of best performance values, a vector $\in \{1, \ldots, a\}^{n}$ of best performing algorithms, and a set of output parameters, Π.

Function PRELIM $(,,\varepsilon, max, abs, bnd, nrm)$ **is**
    // Calculation of the binary measure of ``good'' performance
    if $max =$ **then**
        $[=] \leftarrow -\infty$;
        $\{,\} \leftarrow$ FindMaxByCol(); // Returns the minimum and its index for each column
        if $abs =$ **then** $\leftarrow \geq \varepsilon$;
        else
            $\leftarrow 1 - \left(\oslash \text{RowVectorAsMatrix}(,a)\right)$; // Apply an element-wise division
            $\leftarrow \leq \varepsilon$;
        end
    else
        $[=] \leftarrow \infty$;
        $\{,\} \leftarrow$ FindMinByCol(); // Returns the maximum and its index for each column
        if $abs =$ **then**
            $\leftarrow \left(\oslash \text{RowVectorAsMatrix}(,a)\right) - 1$; // Apply an element-wise division
        end
        $\leftarrow \leq \varepsilon$;
    end
    // Pre-processing function for bounding and scaling of the meta-data
    if $bnd =$ **then**
        $\Pi_{\cdot \text{med}} \leftarrow$ FindMedianByRow();
        $\Pi_{\cdot} \leftarrow$ FindIQRByRow();
        $_{\text{max}} \leftarrow$ ColVectorAsMatrix$(\Pi_{\cdot \text{med}} + 5\Pi_{\cdot}, n)$;
        $_{\text{min}} \leftarrow$ ColVectorAsMatrix$(\Pi_{\cdot \text{med}} - 5\Pi_{\cdot}, n)$;
        $\leftarrow \circ \left(\geq_{\text{min}} \parallel \leq_{\text{max}}\right) +_{\text{min}} \circ (\leq_{\text{min}}) +_{\text{max}} \circ (\geq_{\text{max}})$;
    end
    if $nrm =$ **then**
        $\Pi_{\cdot \text{min}} \leftarrow$ FindMinByRow();
        $\leftarrow -$ColVectorAsMatrix$(\Pi_{\cdot \text{min}} + 1)$;
        $\{,\Pi_{\cdot}\} \leftarrow$ BoxCoxByRow();
        $\{,\Pi_{\cdot},\Pi_{\cdot}\} \leftarrow$ ZScoreByRow();
        $[= 0] \leftarrow \varepsilon$;
        $\{,\Pi_{\cdot}\} \leftarrow$ BoxCoxByRow();
        $\{,\Pi_{\cdot},\Pi_{\cdot}\} \leftarrow$ ZScoreByRow();
    end
    return $\{,,,,\Pi\}$;
end

Algorithm 1: Preparation for Learning of Instance Meta-data (PRELIM) method

Input: A matrix $\in^{m \times n}$ of instance features, a matrix $\in^{a \times n}$ of performance measures, a matrix $\in^{a \times n}$ of binary performance measures, and a desired number of features, K.
Output: A matrix $\in^{q \times n}$ of instance features, and a set of output parameters, Π.

Function SIFTED $(,,K)$ **is**
  $\{\Pi_{.,}, \Pi_{.,}\} \leftarrow$ PearsonCorrByRow$(,)$;
  $\leftarrow \Pi_{.,}$;
  $[= \parallel \Pi_{.,} > 0.05] \leftarrow 0$;
  $\text{idx}_1 \leftarrow$ FindIdxMaxByRow$(||)$; // Index of the maximum absolute value for each row
  $\text{idx}_2 \leftarrow$ FindIdxByRow$(|| \geq 0.3)$; // Index of the elements with absolute value greater than 0.3
  $\Pi.\text{idx}_{\text{cor}} \leftarrow$ FindUniqueValues$(\{\text{idx}_1, \text{idx}_2\})$;
  $\leftarrow [\Pi.\text{idx}_{\text{cor}}, :]$;
  $\Pi_{.,} \leftarrow$ PearsonCorrByRow$(,)$;
  $\Pi. \leftarrow$ KMeans$\left(1 - |\Pi_{.,}|, K\right)$;
  $\Pi.\text{idx}_{\text{clu}} \leftarrow$ FindOptimalCombination$(\Pi_{.,},)$;
  $\leftarrow [\Pi.\text{idx}_{\text{clu}}, :]$;
  return $\{, \Pi\}$;
end

Algorithm 2: Selection of Instance Features to Explain Difficulty (SIFTED)

Input: A matrix $\in^{q \times n}$ of instance features, a matrix $\in^{a \times n}$ of performance measures, and a number of random restarts N_{try}, and a binary flag *num* that determines whether the analytical or the numerical solution is calculated.

Output: A matrix $\in^{n \times 2}$ of coordinates in the $2D$ instance space, and a set of projection matrices $\{_r, _r, _r\}$.

Function PILOT$\left(, , N_{try}, num\right)$ **is**
  if $num =$ **then**
    $\leftarrow [;];$
    $\leftarrow$ FindTopEigenVectors$\left({}^{\top}, 2\right);$
    $_r \leftarrow [1:n, :];$
    $_r \leftarrow [n+1:m, :];$
    $_r \leftarrow {}^{\top}\left({}^{\top}\right)^{-1};$
    $_r \leftarrow {}_r^{\top};$
    $\leftarrow _i;$
  else
    $_H \leftarrow$ EuclideanDist(); // Distance between instances in the feature space
    $_H \leftarrow$ MatrixAsRowVector$(_H)$; // Reshape as a column vector
    $\rho_{\text{best}} \leftarrow -\infty;$
    for $i = 1$ **to** N_{try} **do** // Repeat N_{try} times
      // Initialise the projection matrices randomly between $[-1, 1]$
      $_0 \leftarrow$ UniformRandMatrix$\left(2, m, [-1, 1]\right);$
      $_0 \leftarrow$ UniformRandMatrix$\left(m, 2, [-1, 1]\right);$
      $_0 \leftarrow$ UniformRandMatrix$\left(a, 2, [-1, 1]\right);$
      $\{_i, _i, _i\} \leftarrow$ BFGS$(\mathcal{D}, _0, _0, _0);$ // Use BFGS to find a solution to $\mathcal{D}$
      $_i \leftarrow _i;$
      $_L \leftarrow$ EuclideanDist$(\mathbf{Z_i})$;
      $_L \leftarrow$ MatrixAsRowVector$(_L)$;
      $\rho_i \leftarrow$ PearsonCorrByRow$(_H, _L)$;
      if $\rho_{best} < \rho_i$ **then** $\{_r, _r, _r, \rho_{\text{best}}\} \leftarrow \{_i, _i, _i, _i, \rho_i\};$ // Best solution so far
    end
  end
  return $\{, _r, _r, _r\};$
end

Algorithm 3: Projecting Instances with Linearly Observable Trends (PILOT)

Input: A matrix $\in^{m \times n}$ of instance features, a minimum correlation value for which a par of features would be considered co-linear, ε, and a p-value for which a pair of features would be considered uncorrelated, p.
Output: A matrix $_{\text{edge}} \in^{n \times 2}$ of boundary coordinates in the $2D$ instance space.

Function `CLOISTER`$(,\varepsilon,p)$ **is**
  $\{\rho_{,},p_{,}\} \leftarrow$ `PearsonCorrByRow`$(,)$;
  $\rho_{,}[p_{,} \geq p] \leftarrow 0$;
  $\mathbf{f}_L \leftarrow$ `FindMinByRow`$()$;
  $\mathbf{f}_U \leftarrow$ `FindMaxByRow`$()$;
  $\leftarrow$ `FindAllCombinations`$(\mathbf{f}_L,\mathbf{f}_U)$; `// Generate a hyper-cube enclosing all instances`
  $c \leftarrow$ `NumberOfColumns`$()$;
  for $i = 1$ **to** c **do**
    for $j = 1$ **to** m **do**
      for $i = j+1$ **to** m **do**
        if $\rho_{,}[j,k] > \varepsilon$ && `Sign`$([j,i]) \neq$ `Sign`$([k,i])$ **then** $\mathbf{r}_i =$;
        else if $\rho_{,}[j,k] < -\varepsilon$ && `Sign`$([j,i]) =$ `Sign`$([k,i])$ **then** $\mathbf{r}_i =$;
      end
    end
  end
  $Z_c \leftarrow_i [\neg\mathbf{r},:]$;
  $_{\text{edge}} \leftarrow$ `FindConvexHull`$(\mathbf{Z}_c)$;
  return $\{_{\text{edge}}\}$;
end

Algorithm 4: Correlated Limits of the Instance Space's Theoretical or Experimental Regions (CLOISTER)

Input: A matrix $\in^{n \times 2}$ of coordinates in the $2D$ instance space, a matrix $\in^{a \times n}$ of performance measures, a matrix of binary performance measures, $\in^{a \times n}$, a vector of best performance values, $\in^{n}$, and a vector of best performing algorithms, $\in \{1,\ldots,a\}^n$.
Output: A set of SVM models , a matrix of cross-validated estimated binary performance measures, $_{\text{cv,bin}} \in^{a \times n}$, a vector of cross-validated estimated best performing algorithms, $_{\text{cv}} \in \{1,\ldots,a\}^n$, and its equivalents using the full dataset as training data, $_{\text{bin}}$ and , a tensor $\mathbf{C} \in^{k,n,a}$ that identifies the cross-validation sets.

Function `PYTHIA`$(,,,,)$ **is**
  $\{_{\text{norm}},\Pi.,\Pi.\} \leftarrow$ `ZScoreByRow`(Z)
  for $i = 1$ **to** a **do**
    $\mathbf{C}[:,:,i] \leftarrow$ `CreateCVPartition`$([i,:],k)$;
    $\{[i]_{,\text{cv,bin}},_{,\text{bin}},\Pi.C[i],\Pi.\gamma[i]\} \leftarrow$ `CrossValidatedSVMTrain`$(_{\text{norm}},[i,:],\mathbf{C}[:,:,i])$;
    $\{\Pi.P[i],\Pi.R[i],\Pi.A[i]\} \leftarrow$ `CalculateSVMPerformance`$(_{,\text{cv,bin}})$;
  end
  $_{\text{cv}} \leftarrow$ `FindMaxIndexByCol`$(_{\text{cv,bin}} \circ$ `ColVectorAsMatrix`$(\Pi.P,n))$;
  $\leftarrow$ `FindMaxIndexByCol`$(_{\text{bin}} \circ$ `ColVectorAsMatrix`$(\Pi.P,n))$;
  return $\{_{,\text{cv,bin}},_{,\text{bin}},_{,\text{cv}},,\Pi\}$
end

Algorithm 5: Performance prediction and automated algorithm selection (PYTHIA)

Input: A matrix $\in^{n\times 2}$ of coordinates in the $2D$ instance space, a matrix $\in^{n\times a}$ of binary performance measures, and a vector $\in \{1,\ldots,a\}^{n\times 1}$ of indexes indicating the best performing algorithm for an instance.

Output: Two footprint sets, $\left\{\boldsymbol{\Phi}_{\text{good}}, \boldsymbol{\Phi}_{\text{best}}\right\}$, which correspond to good and best performance.

Function `FootprintAnalysis(,,)` **is**

  $S \leftarrow$ `TRACEbuild(,`$\mathbf{1}$`);` `// Calculate the area, density and purity of the space`

  for $i = 1$ **to** a **do** `// Build two footprints for all algorithms`

    $\boldsymbol{\Phi}_{\text{good},i} \leftarrow$ `TRACEbuild(,` $_i$`);`

    $\boldsymbol{\Phi}_{\text{best},i} \leftarrow$ `TRACEbuild(,`$= i$`);`

  end

  for $i = 1$ **to** a **do**

    for $j = i+1$ **to** a **do**

      `// Compare the best performance footprints for two different algorithms and retain the areas with the highest confidence`

      $\left\{\boldsymbol{\Phi}_{\text{best},i}, \boldsymbol{\Phi}_{\text{best},j}\right\} \leftarrow$ `TRACEcomp` $\left(\boldsymbol{\Phi}_{\text{best},i}, \boldsymbol{\Phi}_{\text{best},j}, , = i, = j\right)$;

    end

  end

end

Algorithm 6: Good and Best Footprints via the Triangulation with Removal of Areas with Contradicting Evidence (TRACE) method

Input: A matrix $\in^{n\times 2}$ of coordinates in the $2D$ instance space and a vector $\in^{n\times 1}$ of binary performance measures.

Output: A footprint $\boldsymbol{\Phi}$.

Function `TRACEbuild(,)` **is**

  $_u \leftarrow$ `UniquePoints` $\left(\left\{_i | y_i = \right\}\right)$; `// Find the unique points` $_u$ `that have` $y_i =$

  $\{r,c\} \leftarrow$ `MatrixSize` $(_u)$; `// Find the number of rows and columns of the matrix`

  $k \leftarrow \max\left(\min\left(\lceil r/20 \rceil, 50\right), 3\right)$;

  $\varepsilon \leftarrow \left(k\Gamma(2)/\sqrt{r\pi}\right)\left(\text{range}(_1) \times \text{range}(_2)\right)$;

  $\mathbf{c} \leftarrow$ `DBSCAN` $(_u, k, \varepsilon)$; `// Use DBSCAN to identify outliers and clusters of dense data`

  $\boldsymbol{\Phi}$.polygon $\leftarrow \emptyset$

  for $i = 1$ **to** $\max(\mathbf{c})$ **do**

    `// For every detected cluster, build an` α`-shape`

    $\boldsymbol{\Phi}$.polygon $\leftarrow$ `JoinPolygons` $\left(\boldsymbol{\Phi}\text{.polygon}, \text{BuildAlphaShape}\left(\left\{_i | c_i = i\right\}\right)\right)$;

  end

  $\boldsymbol{\Phi}$.area $\leftarrow$ `FindPolygonArea` $(\boldsymbol{\Phi}\text{.polygon})$;

  $\boldsymbol{\Phi}$.density $\leftarrow$ `CountElements` $(\boldsymbol{\Phi}\text{.polygon},)/\boldsymbol{\Phi}$.area;

  $\boldsymbol{\Phi}$.purity $\leftarrow$ `CountElements` $\left(\boldsymbol{\Phi}\text{.polygon}, \left\{_i | y_i = \right\}\right)$ `/CountElements` $(\boldsymbol{\Phi}\text{.polygon},)$;

end

Algorithm 7: TRACEbuild: Footprint construction algorithm

Input: A base and test footprints $\{\Phi_B, \Phi_T\}$, a matrix $\in^{n\times 2}$ of coordinates in the $2D$ instance space and two vectors $_{B,T} \in^{n\times 1}$ of binary performance measures.

Output: A base and test footprints $\{\Phi_B, \Phi_T\}$ with removed contradictions.

Function `TRACEcomp` $(\Phi_B, \Phi_T, _{,B}, _{,T})$ **is**

$C \leftarrow$ `PolygonIntersection` $(\Phi_B.\text{polygon}, \Phi_T.\text{polygon})$;

$N_{\text{try}} \leftarrow 0$;

$N_{\max} \leftarrow 3$;

while `FindPolygonArea` $(C) > 0 \wedge N_{try} < N_{\max}$ **do**

$\pi_B \leftarrow$ `CountElements` $\left(C, \{_i | y_{B,i} =\}\right)$ / `CountElements` $(C,)$;

$\pi_T \leftarrow$ `CountElements` $\left(C, \{_i | y_{T,i} =\}\right)$ / `CountElements` $(C,)$;

if $\pi_B > \pi_T$ **then**

$\Phi_T.\text{polygon} \leftarrow$ `RemovePolygon` $(\Phi_T.\text{polygon}, C)$;

else if $\pi_B < \pi_T$ **then**

$\Phi_B.\text{polygon} \leftarrow$ `RemovePolygon` $(\Phi_B.\text{polygon}, C)$;

else

break;

end

$C \leftarrow$ `PolygonIntersection` $(\Phi_B.\text{polygon}, \Phi_T.\text{polygon})$;

$N_{\text{try}} \leftarrow N_{\text{try}} + 1$;

end

$\Phi_B.\text{area} \leftarrow$ `FindPolygonArea` $(\Phi_B.\text{polygon})$;

$\Phi_B.\text{density} \leftarrow$ `CountElements` $(\Phi_B.\text{polygon},)\,/\,\Phi_B.\text{area}$;

$\Phi_B.\text{purity} \leftarrow$ `CountElements` $\left(\Phi_B.\text{polygon}, \{_i | y_i =\}\right)$ / `CountElements` $(\Phi_B.\text{polygon},)$;

$\Phi_T.\text{area} \leftarrow$ `FindPolygonArea` $(\Phi_T.\text{polygon})$;

$\Phi_T.\text{density} \leftarrow$ `CountElements` $(\Phi_T.\text{polygon},)\,/\,\Phi_T.\text{area}$;

$\Phi_T.\text{purity} \leftarrow$ `CountElements` $\left(\Phi_T.\text{polygon}, \{_i | y_i =\}\right)$ / `CountElements` $(\Phi_T.\text{polygon},)$;

end

Algorithm 8: TRACEcomp: Footprint comparison algorithm