Footprints in Instance Space: Steps Towards a Free Lunch

Kate Smith-Miles

School of Mathematical Sciences
Monash University

South Pacific Optimisation Meeting (SPOM), Newcastle Univ., Feb. 2013
This research is funded by ARC Discovery Project grant DP120103678 “Footprints in Instance Space: Visualising the Suitability of Optimisation Algorithms”

The Travelling Salesman results are based on work with Dr. Jano van Hemert (University of Edinburgh), and my former student Thomas T. Tan

The Graph Colouring results are based on work with Dr. Davaa Baatar (Monash University), Dr. Rhyd Lewis (University of Cardiff), Dr. Leo Lopes (SAS, USA), and former students Nur Insani and Brendan Wreford

Thanks to Prof. Gordon Royle (University of Western Australia) for informing us about the “energy” of a graph
Acknowledgements

- This research is funded by ARC Discovery Project grant DP120103678 “Footprints in Instance Space: Visualising the Suitability of Optimisation Algorithms”

- The Travelling Salesman results are based on work with Dr. Jano van Hemert (University of Edinburgh), and my former student Thomas T. Tan

- The Graph Colouring results are based on work with Dr. Davaa Baatar (Monash University), Dr. Rhyd Lewis (University of Cardiff), Dr. Leo Lopes (SAS, USA), and former students Nur Insani and Brendan Wreford

- Thanks to Prof. Gordon Royle (University of Western Australia) for informing us about the “energy” of a graph
Acknowledgements

- This research is funded by ARC Discovery Project grant DP120103678 “Footprints in Instance Space: Visualising the Suitability of Optimisation Algorithms”

- The Travelling Salesman results are based on work with Dr. Jano van Hemert (University of Edinburgh), and my former student Thomas T. Tan

- The Graph Colouring results are based on work with Dr. Davaa Baatar (Monash University), Dr. Rhyd Lewis (University of Cardiff), Dr. Leo Lopes (SAS, USA), and former students Nur Insani and Brendan Wreford

- Thanks to Prof. Gordon Royle (University of Western Australia) for informing us about the “energy” of a graph
Acknowledgements

- This research is funded by ARC Discovery Project grant DP120103678 “Footprints in Instance Space: Visualising the Suitability of Optimisation Algorithms”

- The Travelling Salesman results are based on work with Dr. Jano van Hemert (University of Edinburgh), and my former student Thomas T. Tan

- The Graph Colouring results are based on work with Dr. Davaa Baatar (Monash University), Dr. Rhyd Lewis (University of Cardiff), Dr. Leo Lopes (SAS, USA), and former students Nur Insani and Brendan Wreford

- Thanks to Prof. Gordon Royle (University of Western Australia) for informing us about the “energy” of a graph
No-Free-Lunch Theorem (Wolpert & Macready, 1997)

- Standard practice is to use benchmark instances of optimisation problems to report strengths (rarely weaknesses!) of algorithms.

- NFL Theorem warns us against expecting a single algorithm to perform well on all instances of a problem, regardless of their structure and characteristics.

- The properties (or measurable features) of an instance tell us a lot about how an algorithm is expected to perform across a range of instances.

Reference

No-Free-Lunch Theorem (Wolpert & Macready, 1997)

- Standard practice is to use benchmark instances of optimisation problems to report strengths (rarely weaknesses!) of algorithms.

- NFL Theorem warns us against expecting a single algorithm to perform well on all instances of a problem, regardless of their structure and characteristics.

- The properties (or measurable features) of an instance tell us a lot about how an algorithm is expected to perform across a range of instances.

Reference

No-Free-Lunch Theorem (Wolpert & Macready, 1997)

- Standard practice is to use benchmark instances of optimisation problems to report strengths (rarely weaknesses!) of algorithms.

- NFL Theorem warns us against expecting a single algorithm to perform well on all instances of a problem, regardless of their structure and characteristics.

- The properties (or measurable features) of an instance tell us a lot about how an algorithm is expected to perform across a range of instances.

Reference

No-Free-Lunch Theorem (Wolpert & Macready, 1997)

- Standard practice is to use benchmark instances of optimisation problems to report strengths (rarely weaknesses!) of algorithms.

- NFL Theorem warns us against expecting a single algorithm to perform well on all instances of a problem, regardless of their structure and characteristics.

- The properties (or measurable features) of an instance tell us a lot about how an algorithm is expected to perform across a range of instances.

Reference

Travelling Salesman Problem (TSP) Example

Initial tour length = 14265.2
Press Continue to find better tour

There are 8.717829e+010 tours of 15 cities

Easy

Hard
What makes the TSP easy or hard?

A TSP Formulation (not the only one)

- Let $X_{i,j} = 1$ if city $i$ is followed by city $j$ in the tour; 0 otherwise.
- minimise
  $$\sum_{i=1}^{N} \sum_{j=1}^{N} D_{i,j} X_{i,j}$$
- subject to
  $$\sum_{i} X_{i,j} = 1 \quad \forall j$$
  $$\sum_{j} X_{i,j} = 1 \quad \forall i$$
  $$\sum_{i \in S} \sum_{j \in S} X_{i,j} \leq |S| - 1 \quad \forall S \neq \{0\}, S \subset \{1, 2, \ldots, N\}$$

- TSP is NP-hard, but some instances are easy depending on properties of the inter-city distance matrix $D$. 
What makes the TSP easy or hard?

A TSP Formulation (not the only one)

- Let \( X_{i,j} = 1 \) if city \( i \) is followed by city \( j \) in the tour; 0 otherwise
- minimise
  \[
  \sum_{i=1}^{N} \sum_{j=1}^{N} D_{i,j} X_{i,j}
  \]
- subject to
  \[
  \sum_{i} X_{i,j} = 1 \quad \forall j
  \]
  \[
  \sum_{j} X_{i,j} = 1 \quad \forall i
  \]
  \[
  \sum_{i \in S} \sum_{j \in S} X_{i,j} \leq |S| - 1 \quad \forall S \neq \{0\}, S \subset \{1, 2, \ldots, N\}
  \]

- TSP is NP-hard, but some instances are easy depending on properties of the inter-city distance matrix \( D \)
Questions

- How do instance features help us understand the strengths and weaknesses of optimisation algorithms?
- How can we infer and visualise algorithm performance across a huge “instance space”?
- How can we measure objectively the relative performance of algorithms?
- How easy or hard are the benchmark instances in the literature?
Questions

- How do instance features help us understand the strengths and weaknesses of optimisation algorithms?
- How can we infer and visualise algorithm performance across a huge “instance space”?
- How can we measure objectively the relative performance of algorithms?
- How easy or hard are the benchmark instances in the literature?
Questions

- How do instance features help us understand the strengths and weaknesses of optimisation algorithms?
- How can we infer and visualise algorithm performance across a huge “instance space”?
- How can we measure objectively the relative performance of algorithms?
- How easy or hard are the benchmark instances in the literature?
Questions

- How do instance features help us understand the strengths and weaknesses of optimisation algorithms?
- How can we infer and visualise algorithm performance across a huge “instance space”?
- How can we measure objectively the relative performance of algorithms?
- How easy or hard are the benchmark instances in the literature?
Aims

- Develop a new methodology to
  - visualise “instance space” based on instance features
  - visualise algorithm performance across the instance space
  - define where algorithm performance is expected to be “good” (called the “algorithm footprint”)
  - measure the relative size of an algorithm’s footprint

- Enable objective assessment of the power of optimisation algorithms.

- Understand and report the boundary of good performance of an algorithm – essential for good research practice, and to avoid deployment disasters.
Aims

- Develop a new methodology to
  - visualise “instance space” based on instance features
  - visualise algorithm performance across the instance space
  - define where algorithm performance is expected to be “good” (called the “algorithm footprint”)
  - measure the relative size of an algorithm’s footprint

- Enable objective assessment of the power of optimisation algorithms.

- Understand and report the boundary of good performance of an algorithm – essential for good research practice, and to avoid deployment disasters.
Aims

- Develop a new methodology to
  - visualise “instance space” based on instance features
  - visualise algorithm performance across the instance space
  - define where algorithm performance is expected to be “good” (called the “algorithm footprint”)
  - measure the relative size of an algorithm’s footprint

- Enable objective assessment of the power of optimisation algorithms.

- Understand and report the boundary of good performance of an algorithm – essential for good research practice, and to avoid deployment disasters.
Algorithm Selection Problem, Rice (1976)

For a given problem instance \( x \in P \), with features \( f(x) \in F \), find the mapping \( S(f(x)) \) into algorithm space \( A \), such that the selected algorithm \( \alpha \in A \) maximises the performance mapping \( y(\alpha(x)) \in Y \).
Applications of Rice’s Framework: PDEs

- Rice and colleagues used this approach to predict the performance of the many methods (A) for numerical solution of elliptic partial differential equations (PDEs).

Reference


- PYTHIA matches the characteristics (F) of a given problem (x) with those of PDEs in an existing problem population (P).
- It then uses learned performance profiles (S) of the various solvers to select the appropriate method given user-specified error and solution time bounds (Y).
Applications of Rice’s Framework: PDEs

- Rice and colleagues used this approach to predict the performance of the many methods (A) for numerical solution of elliptic partial differential equations (PDEs).

Reference

- PYTHIA matches the characteristics (F) of a given problem (x) with those of PDEs in an existing problem population (P).

  It then uses learned performance profiles (S) of the various solvers to select the appropriate method given user-specified error and solution time bounds (Y).
Applications of Rice’s Framework: PDEs

- Rice and colleagues used this approach to predict the performance of the many methods ($A$) for numerical solution of elliptic partial differential equations (PDEs).

Reference


- PYTHIA matches the characteristics ($F$) of a given problem ($x$) with those of PDEs in an existing problem population ($P$).

- It then uses learned performance profiles ($S$) of the various solvers to select the appropriate method given user-specified error and solution time bounds ($Y$).
Applications of Rice’s Framework: Machine Learning

- For the last two decades, the field of meta-learning (learning about learning algorithms) has emerged.

- Here, measurable features of classification and prediction problems are used to predict the performance of machine learning algorithms.

- While Rice recommended regression models to model the relationship between features and algorithm performance, we can apply more powerful statistical or machine learning methods to this task.

Reference

Applications of Rice’s Framework: Machine Learning

- For the last two decades, the field of meta-learning (learning about learning algorithms) has emerged.

- Here, measurable features of classification and prediction problems are used to predict the performance of machine learning algorithms.

- While Rice recommended regression models to model the relationship between features and algorithm performance, we can apply more powerful statistical or machine learning methods to this task.

Reference
Applications of Rice’s Framework: Machine Learning

- For the last two decades, the field of meta-learning (learning about learning algorithms) has emerged.

- Here, measurable features of classification and prediction problems are used to predict the performance of machine learning algorithms.

- While Rice recommended regression models to model the relationship between features and algorithm performance, we can apply more powerful statistical or machine learning methods to this task.

Reference

Applications of Rice’s Framework: Machine Learning

- For the last two decades, the field of meta-learning (learning about learning algorithms) has emerged.

- Here, measurable features of classification and prediction problems are used to predict the performance of machine learning algorithms.

- While Rice recommended regression models to model the relationship between features and algorithm performance, we can apply more powerful statistical or machine learning methods to this task.

Reference

Extending Rice’s Framework

PHASE 3

1. Algorithms \{A\}
2. Problem instances \{P\}
3. Algorithm Performance Results \{Y\}
4. Dataset Features \{F\}

PHASE 2

5. Empirical Rules
6. Automated Algorithm Selection
7. Theoretical Support
8. Refinement of Algorithms

(Meta-)learning of meta-data

PHASE 1
Applications to Optimisation

- Represents a new direction for the OR community.

- Much needed, given
  - huge range of algorithms
  - frequent statements like “currently there is still a strong lack of . . . understanding of how exactly the relative performance of different meta-heuristics depends on instance characteristics.”

- Can also resolve longstanding debate about how instance choice affects evaluation of algorithm performance

Reference

Applications to Optimisation

- Represents a new direction for the OR community.
- Much needed, given
  - huge range of algorithms
  - frequent statements like “currently there is still a strong lack of... understanding of how exactly the relative performance of different meta-heuristics depends on instance characteristics.”
- Can also resolve longstanding debate about how instance choice affects evaluation of algorithm performance

Reference

Applications to Optimisation

- Represents a new direction for the OR community.

- Much needed, given
  - huge range of algorithms
  - frequent statements like “currently there is still a strong lack of . . . understanding of how exactly the relative performance of different meta-heuristics depends on instance characteristics.”

- Can also resolve longstanding debate about how instance choice affects evaluation of algorithm performance

Reference

Amongst the many algorithms, we consider two variants of the famous Lin-Kernighan heuristic

- Lin-Kernighan with Cluster Compensation (LKCC)
- chained Lin-Kernighan (CLK)

Lin-Kernighan is an edge-swapping heuristic

Which algorithm is better, and for which types of instances?
Amongst the many algorithms, we consider two variants of the famous Lin-Kernighan heuristic

- Lin-Kernighan with Cluster Compensation (LKCC)
- chained Lin-Kernighan (CLK)

Lin-Kernighan is an edge-swapping heuristic

Which algorithm is better, and for which types of instances?
Step 1: Summarising Instances as Feature Vectors

- There are many statistical properties of an instance that correlate with “hardness” of an instance
  - generic properties like number of variables, constraints
  - problem-specific properties like eigenvalues of adjacency matrix in graph colouring; slackness in rooms for timetabling; etc.

- Suppose an instance is summarised by $n$ (topology preserving) features:
  - each instance can be represented as a point in $\mathbb{R}^n$
  - two instances that are similar should be close in $\mathbb{R}^n$
  - two instances that are very dissimilar should be far apart in $\mathbb{R}^n$

- Similar instances elicit similar behaviour from algorithms, except where we observe a phase transition.
Step 1: Summarising Instances as Feature Vectors

- There are many statistical properties of an instance that correlate with “hardness” of an instance:
  - generic properties like number of variables, constraints
  - problem-specific properties like eigenvalues of adjacency matrix in graph colouring; slackness in rooms for timetabling; etc.
- Suppose an instance is summarised by \( n \) (topology preserving) features:
  - each instance can be represented as a point in \( \mathbb{R}^n \)
  - two instances that are similar should be close in \( \mathbb{R}^n \)
  - two instances that are very dissimilar should be far apart in \( \mathbb{R}^n \)
- Similar instances elicit similar behaviour from algorithms, except where we observe a \textit{phase transition}
Step 1: Summarising Instances as Feature Vectors

- There are many statistical properties of an instance that correlate with “hardness” of an instance
  - generic properties like number of variables, constraints
  - problem-specific properties like eigenvalues of adjacency matrix in graph colouring; slackness in rooms for timetabling; etc.

- Suppose an instance is summarised by \( n \) (topology preserving) features:
  - each instance can be represented as a point in \( \mathbb{R}^n \)
  - two instances that are similar should be close in \( \mathbb{R}^n \)
  - two instances that are very dissimilar should be far apart in \( \mathbb{R}^n \)

- Similar instances elicit similar behaviour from algorithms, except where we observe a phase transition
Step 1: TSP

- A comprehensive set of \( n = 40 \) features were used to summarize the properties of TSP instances:
  - Standard deviation of inter-city distances
  - Radius of TSP instance (mean distance from cities to centroid)
  - Fraction of distinct distances in the distance matrix
  - Rectangular area in which the cities lie
  - Normalized variance and coefficient of variation of the nNNd’s (normalized nearest neighbor distances)
  - Number of clusters found using GDBSCAN, \( \text{max}=10 \)
  - Cluster ratio (number of clusters to the number of cities)
  - Outlier ratio (number of outliers to number of cities)
  - Variance of the number of cities in each cluster
  - Ratio of nodes near the edges of the plane
  - Mean radius of the clusters
  - etc.
Step 2: Evolving Easy and Hard Instances

- To see the strengths and weaknesses of algorithms we need diverse instance, well-spread across feature space
  - also important for statistical generalisation
- Randomly generated or benchmark instances are frequently not diverse enough on the spectrum of difficulty
- We also generate instances that have been evolved from random instances to be intentionally easy or hard for a given algorithm under consideration
  - to create hard instances, the fitness function can be the run-time to find an optimal solution, or the solution gap to a known optimal solution after a fixed run-time
  - inverse for easy instances
  - take ratio of one algorithm’s performance to another to generate uniquely easy or hard instances
Step 2: Evolving Easy and Hard Instances

- To see the strengths and weaknesses of algorithms we need diverse instance, well-spread across feature space
  - also important for statistical generalisation
- Randomly generated or benchmark instances are frequently not diverse enough on the spectrum of difficulty
- We also generate instances that have been evolved from random instances to be intentionally easy or hard for a given algorithm under consideration
  - to create hard instances, the fitness function can be the run-time to find an optimal solution, or the solution gap to a known optimal solution after a fixed run-time
  - inverse for easy instances
  - take ratio of one algorithm’s performance to another to generate uniquely easy or hard instances
Step 2: Evolving Easy and Hard Instances

To see the strengths and weaknesses of algorithms we need diverse instance, well-spread across feature space
  ▶ also important for statistical generalisation

Randomly generated or benchmark instances are frequently not diverse enough on the spectrum of difficulty

We also generate instances that have been evolved from random instances to be intentionally easy or hard for a given algorithm under consideration
  ▶ to create hard instances, the fitness function can be the run-time to find an optimal solution, or the solution gap to a known optimal solution after a fixed run-time
  ▶ inverse for easy instances
  ▶ take ratio of one algorithm’s performance to another to generate uniquely easy or hard instances
Evolving Instances

fitness = % gap to Concorde solution after fixed run-time

run TSP-solver on each instance

create TSP instances uniform randomly

using crossover & mutation create new TSP instances

replace the population employing elitism

maximum number of generations reached?

stop

start

Reference

Step 2: TSP

- We consider only $N = 100$ city TSP instances, and have three types of instances:
  - randomly generated: 190 instances randomly placing 100 cities in a 400 x 400 plane
  - TSPLIB benchmark instances: 6 instances with exactly 100 cities (kroA100, kroB100, kroC100, kroD100, kroE100, and rd100)
  - evolved easy and hard for each algorithm: 190 instances of easy and hard for each of CLK and LKCC
  - evolved uniquely easy and hard for each algorithm: 190 instances that are easy for CLK but hard for LKCC, and 190 instances that are hard for CLK but easy for LKCC

- In total we have 1336 100-city TSP instances that are intentionally diverse
Step 2: TSP

- We consider only \( N = 100 \) city TSP instances, and have three types of instances:
  - randomly generated: 190 instances randomly placing 100 cities in a 400 x 400 plane
  - TSPLIB benchmark instances: 6 instances with exactly 100 cities (kroA100, kroB100, kroC100, kroD100, kroE100, and rd100)
  - evolved easy and hard for each algorithm: 190 instances of easy and hard for each of CLK and LKCC
  - evolved uniquely easy and hard for each algorithm: 190 instances that are easy for CLK but hard for LKCC, and 190 instances that are hard for CLK but easy for LKCC

- In total we have 1336 100-city TSP instances that are intentionally diverse
Step 3: Visualising Instances in Feature Space

- Suppose we have a set of $m$ instances and $n$ measurable features for each instance
  - we store the data in a matrix $X \in \mathbb{R}^{m \times n}$
- We use Principal Components Analysis (PCA) to project $X$ from $\mathbb{R}^{m \times n}$ to $\mathbb{R}^{m \times 2}$
  - each instance can now be visualised in $\mathbb{R}^2$ (axes are the top two eigenvectors of $X^T X$)
  - the noise in the data has been reduced
  - essential relationships defining similarities and differences between instances are preserved
  - we call this 2-d projected feature space the *instance space*
- If similar instances are not grouped together in instance space, we must question the discriminatory power of our feature set and revisit Step 1.
Step 3: Visualising Instances in Feature Space

- Suppose we have a set of $m$ instances and $n$ measurable features for each instance
  - we store the data in a matrix $X \in \mathbb{R}^{m \times n}$
- We use Principal Components Analysis (PCA) to project $X$ from $\mathbb{R}^{m \times n}$ to $\mathbb{R}^{m \times 2}$
  - each instance can now be visualised in $\mathbb{R}^2$ (axes are the top two eigenvectors of $X^T X$)
  - the noise in the data has been reduced
  - essential relationships defining similarities and differences between instances are preserved
  - we call this 2-d projected feature space the *instance space*
- If similar instances are not grouped together in instance space, we must question the discriminatory power of our feature set and revisit Step 1.
Step 3: Visualising Instances in Feature Space

- Suppose we have a set of \( m \) instances and \( n \) measurable features for each instance
  - we store the data in a matrix \( X \in \mathbb{R}^{m \times n} \)

- We use Principal Components Analysis (PCA) to project \( X \) from \( \mathbb{R}^{m \times n} \) to \( \mathbb{R}^{m \times 2} \)
  - each instance can now be visualised in \( \mathbb{R}^2 \) (axes are the top two eigenvectors of \( X^T X \))
  - the noise in the data has been reduced
  - essential relationships defining similarities and differences between instances are preserved
  - we call this 2-d projected feature space the *instance space*

- If similar instances are not grouped together in instance space, we must question the discriminatory power of our feature set and revisit Step 1.
Step 3: TSP Instances in Instance Space
Step 4: Visualising Algorithm Footprints in Instance Space

- We can now superimpose the performance of algorithms in the instance space by evaluating them on our $m$ instances.

- We first need to define “good” performance, where the algorithm solves an instance easily (e.g. 1% optimality gap).

- All instances are labelled 1 (“good”) or 0 (“bad”), and we create a binary matrix $Y \in \mathbb{R}^{m \times |A|}$, where $|A|$ is the number of algorithms we are considering.

- For a given algorithm, we consider points labelled as good, and:
  - remove outliers through clustering,
  - calculate the convex hull to define a generalised area of expected good performance,
  - remove the convex hull of contradicting points,
  - validate the accuracy of the remaining “footprint” through out-of-sample testing.
Step 4: Visualising Algorithm Footprints in Instance Space

- We can now superimpose the performance of algorithms in the instance space by evaluating them on our $m$ instances.

- We first need to define “good” performance, where the algorithm solves an instance easily (e.g. 1% optimality gap).

- All instances are labelled 1 (“good”) or 0 (“bad”), and we create a binary matrix $Y \in \mathbb{R}^{m \times |A|}$, where $|A|$ is the number of algorithms we are considering.

- For a given algorithm, we consider points labelled as good, and
  - remove outliers through clustering,
  - calculate the convex hull to define a generalised area of expected good performance,
  - remove the convex hull of contradicting points,
  - validate the accuracy of the remaining “footprint” through out-of-sample testing.
Step 4: Visualising Algorithm Footprints in Instance Space

- We can now superimpose the performance of algorithms in the instance space by evaluating them on our $m$ instances.

- We first need to define “good” performance, where the algorithm solves an instance easily (e.g. 1% optimality gap).

- All instances are labelled 1 ("good") or 0 ("bad"), and we create a binary matrix $Y \in \mathbb{R}^{m \times |A|}$, where $|A|$ is the number of algorithms we are considering.

- For a given algorithm, we consider points labelled as good, and
  - remove outliers through clustering,
  - calculate the convex hull to define a generalised area of expected good performance
  - remove the convex hull of contradicting points
  - validate the accuracy of the remaining “footprint” through out-of-sample testing
Step 4: Visualising Algorithm Footprints in Instance Space

- We can now superimpose the performance of algorithms in the instance space by evaluating them on our $m$ instances.

- We first need to define “good” performance, where the algorithm solves an instance easily (e.g. 1% optimality gap).

- All instances are labelled 1 (“good”) or 0 (“bad”), and we create a binary matrix $Y \in \mathbb{R}^{m \times |A|}$, where $|A|$ is the number of algorithms we are considering.

- For a given algorithm, we consider points labelled as good, and:
  - remove outliers through clustering,
  - calculate the convex hull to define a generalised area of expected good performance
  - remove the convex hull of contradicting points
  - validate the accuracy of the remaining “footprint” through out-of-sample testing.
Step 4: Algorithm Footprints in TSP Instance Space

- CLK performance labels
- LKCC performance labels
Case Study: Graph Colouring

Step 1: Instance Features
Step 2: Instance Generation
Step 3: Visualising Instance Space
Step 4: Visualising Algorithm Footprints
Step 5: Measuring Algorithm Footprints
Convex and Concave Hulls
Convex and Concave Hulls

watch out for contradictions
Step 5: Measuring the Size of Algorithm Footprints

- Now we need only to calculate the area defining the footprint
  - our metric of the power of an algorithm is the ratio of this area to the total area of the instance space

### Area of Algorithm Footprint

- Let \( \mathcal{H}(S) \) be the convex hull of a region defined by a set of points \( S = \{(x_i, y_i) \forall i = 1, \ldots \eta\} \)

\[
\text{Area}(\mathcal{H}(S)) = \frac{1}{2} \sum_{j=1}^{k} (x_jy_{j+1} - y_jx_{j+1}) + (x_ky_1 - y_kx_1)
\]

with the subset \( \{(x_j, y_j) \forall j = 1, \ldots k\} \) and \( k \leq \eta \) defining the extreme points of \( \mathcal{H}(S) \)
Step 5: Measuring the Size of Algorithm Footprints

- Now we need only to calculate the area defining the footprint
  - our metric of the power of an algorithm is the ratio of this area to the total area of the instance space

Area of Algorithm Footprint

Let \( H(S) \) be the convex hull of a region defined by a set of points \( S = \{(x_i, y_i) \forall i = 1, \ldots \eta\} \)

\[
\text{Area}(H(S)) = \frac{1}{2} \sum_{j=1}^{k} (x_jy_{j+1} - y_jx_{j+1}) + (x_ky_1 - y_kx_1)
\]

with the subset \( \{(x_j, y_j) \forall j = 1, \ldots k\} \) and \( k \leq \eta \) defining the extreme points of \( H(S) \)
Step 5: CLK and LKCC Footprint Size

<table>
<thead>
<tr>
<th>Footprints</th>
<th>Convex Hull</th>
<th>Non-Convex Hull</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Points</td>
<td>Area (%)</td>
</tr>
<tr>
<td>CLK</td>
<td>202</td>
<td>11.27 (21.22%)</td>
</tr>
<tr>
<td></td>
<td>17</td>
<td>3.12 (5.87%)</td>
</tr>
<tr>
<td>CLK unique</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LKCC</td>
<td>188</td>
<td>8.84 (16.64%)</td>
</tr>
<tr>
<td>LKCC unique</td>
<td>3</td>
<td>0.70 (1.32%)</td>
</tr>
<tr>
<td>All instances</td>
<td>1336</td>
<td>53.11 (100%)</td>
</tr>
</tbody>
</table>

Good performance defined as 1% gap to known optimal solution (via Concorde TSP solver)
Footprint size versus goodness definition

Top line: good
Middle line: uniquely good
Bottom line: uniquely good with no contradictions
Discussion

How do instance features help us understand the strengths and weaknesses of optimisation algorithms?

- Provided we have the right feature set, we can create a topology-preserving instance space
- The boundary between good and bad performance can be seen
- Feature selection methods may improve topology-preservation

How can we infer and visualise algorithm performance across a huge “instance space”?

- PCA has been used to visualise instances in 2-d (or 3-d)
- More than 90% of variation in data was preserved, but some important information (as well as noise) is naturally lost
- If the 4th largest eigenvalue is still large, then we loose too much detail, and other dimension reduction methods are needed
Discussion

- How do instance features help us understand the strengths and weaknesses of optimisation algorithms?
  - Provided we have the right feature set, we can create a topology-preserving instance space
  - The boundary between good and bad performance can be seen
  - Feature selection methods may improve topology-preservation

- How can we infer and visualise algorithm performance across a huge “instance space”?
  - PCA has been used to visualise instances in 2-d (or 3-d)
  - More than 90% of variation in data was preserved, but some important information (as well as noise) is naturally lost
  - If the 4th largest eigenvalue is still large, then we lose too much detail, and other dimension reduction methods are needed
How can we objectively measure algorithm performance?

- We have proposed a method to calculate the relative size of the area of algorithm footprints
- Convex or concave hulls can be used depending on generalisation comfort (out-of-sample testing can help)
- The area of the footprint depends on the definition of “good”
- LKCC has a larger footprint, for a broader definition of good, than CLK

How easy or hard are the benchmark instances?

- TSPLIB and random instances are all in the middle (average features), and lie within the footprint of both algorithms
- More discriminating instances can be generated intentionally using evolutionary algorithms
- The diversity of instances is critical to achieve a generalised instance space
Discussion, continued

- How can we objectively measure algorithm performance?
  - We have proposed a method to calculate the relative size of the area of algorithm footprints
  - Convex or concave hulls can be used depending on generalisation comfort (out-of-sample testing can help)
  - The area of the footprint depends on the definition of “good”
  - LKCC has a larger footprint, for a broader definition of good, than CLK

- How easy or hard are the benchmark instances?
  - TSPLIB and random instances are all in the middle (average features), and lie within the footprint of both algorithms
  - More discriminating instances can be generated intentionally using evolutionary algorithms
  - The diversity of instances is critical to achieve a generalised instance space
Graph Colouring

- Given an undirected graph $G(V, E)$ with $|V| = n$, colour the vertices such that no two vertices connected by an edge share the same colour.

- Try to find the minimum number of colours needed to colour the graph (chromatic number).

- NP-hard problem $\rightarrow$ numerous heuristics for large $n$.

- Many applications, such as timetabling where edges represent conflicts between events.
What makes graph colouring hard?

- In total we have 18 features that describe a graph instance $G(V,E)$
- 5 features relating to the nodes and edges
  - The number of nodes or vertices in a graph: $n = |V|$
  - The number of edges in a graph: $m = |E|$
  - The density of a graph: the ratio of the number of edges to the number of possible edges.
  - Mean node degree: the degree of a node is the number of connections a node has to other nodes.
  - SD of node degree: the average node degree and its standard deviation can give us an idea of how connected a graph is.
What makes graph colouring hard?

- In total we have 18 features that describe a graph instance $G(V, E)$

- 5 features relating to the nodes and edges
  - The number of nodes or vertices in a graph: $n = |V|$
  - The number of edges in a graph: $m = |E|$
  - The density of a graph: the ratio of the number of edges to the number of possible edges.
  - Mean node degree: the degree of a node is the number of connections a node has to other nodes.
  - SD of node degree: the average node degree and its standard deviation can give us an idea of how connected a graph is.
Graph features (continued)

- 8 features related to cycles and paths on the graph
  - The diameter of a graph: max shortest path distance between any two nodes.
  - Average path length: average length of shortest paths for all node pairs.
  - The girth of a graph: the length of the shortest cycle.
  - The clustering coefficient: a measure of node clustering.
  - Mean betweenness centrality: average fraction of all shortest paths connecting all pairs of nodes that pass through a given node.
  - SD of betweenness centrality: with the mean, the SD gives a measure of how central the nodes are in a graph.
  - Szeged index / revised Szeged index: generalisation of Wiener number to cyclic graphs (correlates with bipartivity)
  - Beta: proportion of even closed walks to all closed walks (correlates with bipartivity)
Graph features (continued)

- 5 features related to the Adjacency and Laplacian matrices
  - Mean eigenvector centrality: the Perron-Frobenius eigenvector of the adjacency matrix, averaged across all components.
  - SD of eigenvector centrality: together with the mean, the standard deviation of eigenvector centrality gives us a measure of the importance of a node inside a graph.
  - Mean spectrum: the mean of absolute values of eigenvalues of the adjacency matrix (a.k.a “energy” of the graph).
  - SD of the set of absolute values of eigenvalues of the adjacency matrix.
  - Algebraic connectivity: the 2nd smallest eigenvalue of the Laplacian matrix, reflecting how well connected a graph is. Cheeger’s constant, another important graph property, is bounded by half the algebraic connectivity.
Graph Colouring Instances

- We use a set of 6788 instances from a variety of well-studied sources, and others we have generated to explore bipartivity.

<table>
<thead>
<tr>
<th>DataSet</th>
<th># instances</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>1000</td>
<td>Bipartivity Controlled</td>
</tr>
<tr>
<td>C1</td>
<td>1000</td>
<td>Culberson: cycle-driven</td>
</tr>
<tr>
<td>C2</td>
<td>932</td>
<td>Culberson: geometric</td>
</tr>
<tr>
<td>C3</td>
<td>1000</td>
<td>Culberson: girth and degree inhibited</td>
</tr>
<tr>
<td>C4</td>
<td>1000</td>
<td>Culberson: IID edge probabilities</td>
</tr>
<tr>
<td>C5</td>
<td>1000</td>
<td>Culberson: weight-biased</td>
</tr>
<tr>
<td>D</td>
<td>743</td>
<td>DIMACS instances</td>
</tr>
<tr>
<td>E</td>
<td>20</td>
<td>Social Network graphs</td>
</tr>
<tr>
<td>F</td>
<td>80</td>
<td>Sports Scheduling</td>
</tr>
<tr>
<td>G</td>
<td>13</td>
<td>Exam Timetabling</td>
</tr>
</tbody>
</table>
Graph Colouring Algorithms

- We use the same 8 algorithms considered by Lewis et al.
  - DSATUR: Brelaz’s greedy algorithm (exact for bipartite graphs)
  - RandGr: Simple greedy first-fit colouring of random permutations of nodes
  - Bktr: a backtracking version of DSATUR (Culberson)
  - HillClimb: a hill-climbing improvement on initial DSATUR solution
  - HEA: Hybrid evolutionary algorithm
  - TabuCol: Tabu search algorithm
  - PartCol: Like TabuCol, but doesn’t restricts to feasible space
  - AntCol: Ant Colony meta-heuristic

Reference

We use the same 8 algorithms considered by Lewis et al.

- DSATUR: Brelaz’s greedy algorithm (exact for bipartite graphs)
- RandGr: Simple greedy first-fit colouring of random permutations of nodes
- Bktr: a backtracking version of DSATUR (Culberson)
- HillClimb: a hill-climbing improvement on initial DSATUR solution
- HEA: Hybrid evolutionary algorithm
- TabuCol: Tabu search algorithm
- PartCol: Like TabuCol, but doesn’t restricts to feasible space
- AntCol: Ant Colony meta-heuristic

Reference

Visualising the instance space

Data Sets in Instance Space. #feat = 18, obj = minAvrErr, dim = R2

- B.dat
- C1.dat
- C2.dat
- C3.dat
- C4.dat
- C5.dat
- D.dat
- E.dat
- F.dat
- G.dat

Footprints in Instance Space
Defining goodness of algorithm performance

- Identify best solution from all 8 algorithms
- Algorithm is good if gap to best solution is $\leq \alpha\%$
- Blue means good=easy, Red means bad=hard

Bktr footprint: $\alpha = 0.04$
Defining difficulty of instances

- If less than a given fraction $\beta$ of the 8 algorithms find an instance easy, then we label the instance as hard for the portfolio of algorithms
  - e.g. if $\beta = 0.5$ then an instance will be labelled hard if less than half (only 1, 2 or 3 of the total eight algorithms) find it easy

- It is important that we understand where good algorithm performance is uninteresting (if all algorithms find the instances easy) or interesting (if other algorithms struggle)
Defining difficulty of instances

- If less than a given fraction $\beta$ of the 8 algorithms find an instance easy, then we label the instance as hard for the portfolio of algorithms
  - e.g. if $\beta = 0.5$ then an instance will be labelled hard if less than half (only 1, 2 or 3 of the total eight algorithms) find it easy

- It is important that we understand where good algorithm performance is uninteresting (if all algorithms find the instances easy) or interesting (if other algorithms struggle)
How many algorithms find an instance hard? ($\alpha = 0$)

Footprints in Instance Space
Algorithm Footprints

Footprints feat = 18, alpha = 0.40, 0.50

Area of Footprints can give quantitative measure of power
Algorithm Footprints

Footprints feat = 18, alpha = 0.04, 500

RandomGreedy  DSATUR  Bktr  HillClimber

HEA  PartialCol  TabuCol  AntCol

Warning: Blue (easy) may be on top of Red (hard)
Learning to predict easy or hard instances for a given \( \alpha, \beta \)

Naive Bayes classifier in \( \mathbb{R}^2 \) is 85% accurate
Recommendng algorithms

Algorithm Selection:
#feat = 18, Err = 0.07875, obj = min Avr Err, dim = R2

But why?
On which instance classes is each algorithm best suited?
Characterising algorithm suitability based on features

- Enables us to see what properties (not instance class labels) explain algorithm performance.
- Representation of instance space (location of instances) depends on feature set.
- We have used a GA to select optimal feature subset to maximise separability (reduce contradictions) in footprints to enable cleaner calculation of area of footprints.
- Using all 18 features, some interesting feature distributions clearly show the properties of instances that create easy or hard instances for each algorithm.
Characterising algorithm suitability based on features

- Enables us to see what properties (not instance class labels) explain algorithm performance.
- Representation of instance space (location of instances) depends on feature set.
- We have used a GA to select optimal feature subset to maximise separability (reduce contradictions) in footprints to enable cleaner calculation of area of footprints.
- Using all 18 features, some interesting feature distributions clearly show the properties of instances that create easy or hard instances for each algorithm.
Characterising algorithm suitability based on features

- Enables us to see what properties (not instance class labels) explain algorithm performance.

- Representation of instance space (location of instances) depends on feature set.

- We have used a GA to select optimal feature subset to maximise separability (reduce contradictions) in footprints to enable cleaner calculation of area of footprints.

- Using all 18 features, some interesting feature distributions clearly show the properties of instances that create easy or hard instances for each algorithm.
Characterising algorithm suitability based on features

- Enables us to see what properties (not instance class labels) explain algorithm performance.
- Representation of instance space (location of instances) depends on feature set.
- We have used a GA to select optimal feature subset to maximise separability (reduce contradictions) in footprints to enable cleaner calculation of area of footprints.
- Using all 18 features, some interesting feature distributions clearly show the properties of instances that create easy or hard instances for each algorithm.
Feature Distributions in Instance Space

Feature Distribution:
AlgConnectivity

Footprints in Instance Space
Feature Distribution: Density

Footprints in Instance Space
Feature Distribution:
EigenVectorCentrMean

Footprints in Instance Space
Feature Distribution:
Nodes

Footprints in Instance Space
Reference

Reference

HEA is not best everywhere (NFL) ... why not?
The proposed methodology is a first step towards providing researchers with a tool to

- report the strengths and weaknesses of their algorithms
- show the relative power of an algorithm either
  - across the entire instance space, or
  - in a particular region of interest (e.g. real world problems).

We are currently developing the key components of the methodology (evolved instances, feature sets) for a number of broad classes of optimization problems.

The approach generalises to parameter selection within algorithms as well, and to choice of formulation.

We hope to be providing a free lunch for optimisation researchers soon!
Conclusions

- The proposed methodology is a first step towards providing researchers with a tool to
  - report the strengths and weaknesses of their algorithms
  - show the relative power of an algorithm either
    - across the entire instance space, or
    - in a particular region of interest (e.g. real world problems).

- We are currently developing the key components of the methodology (evolved instances, feature sets) for a number of broad classes of optimization problems.

- The approach generalises to parameter selection within algorithms as well, and to choice of formulation.

- We hope to be providing a free lunch for optimisation researchers soon!
The proposed methodology is a first step towards providing researchers with a tool to

- report the strengths and weaknesses of their algorithms
- show the relative power of an algorithm either
  - across the entire instance space, or
  - in a particular region of interest (e.g. real world problems).

We are currently developing the key components of the methodology (evolved instances, feature sets) for a number of broad classes of optimization problems.

The approach generalises to parameter selection within algorithms as well, and to choice of formulation.

We hope to be providing a free lunch for optimisation researchers soon!
Conclusions

- The proposed methodology is a first step towards providing researchers with a tool to
  - report the strengths and weaknesses of their algorithms
  - show the relative power of an algorithm either
    - across the entire instance space, or
    - in a particular region of interest (e.g. real world problems).
- We are currently developing the key components of the methodology (evolved instances, feature sets) for a number of broad classes of optimization problems.
- The approach generalises to parameter selection within algorithms as well, and to choice of formulation.
- We hope to be providing a free lunch for optimisation researchers soon!
Further Reading

- Travelling Salesman Problem
- Quadratic Assignment Problem
- Job Shop Scheduling
- Timetabling
- Graph Colouring