Why solving the travelling salesman problem can be difficult: a hunt for hard instances

Jano van Hemert
<jvhemert@cwi.nl>

Centre of Emergent Computing, Napier University, Edinburgh
Motivation

① Combining studies on creating a TSP problem generator
Motivation

① Combining studies on creating a TSP problem generator

② and studies on evolving binary constraint satisfaction problems
Motivation

1. Combining studies on creating a TSP problem generator
2. and studies on evolving binary constraint satisfaction problems
3. into a search for interesting structural properties of TSP instances.
TSP in one page

Objective: in a graph, find the Hamiltonian cycle with minimal length.
TSP in one page

Objective: in a graph, find the Hamiltonian cycle with minimal length.
TSP in one page

Objective: in a graph, find the Hamiltonian cycle with minimal length.
An evolutionary TSP generator
TSP generator

✔ Idea in (van Hemert and Urquhart, 2004) is to create maps for vehicle routing problems
TSP generator

✔ Idea in (van Hemert and Urquhart, 2004) is to create maps for vehicle routing problems

✔ These maps are more life-like by containing clusters of nodes rather than a uniform distribution of the nodes (van Hemert and La Poutré, 2004)
TSP generator

✔ Idea in (van Hemert and Urquhart, 2004) is to create maps for vehicle routing problems

✔ These maps are more life-like by containing clusters of nodes rather than a uniform distribution of the nodes (van Hemert and La Poutré, 2004)

✔ Using an evolutionary algorithm maps are generated where certain properties are being satisfied
TSP generator

✔ Representation: list of 100 \((x, y)\) points on a \(500 \times 500\) map
TSP generator

✔ Representation: list of 100 \((x, y)\) points on a \(500 \times 500\) map

✔ Objective is to satisfy a minimum and maximum distance constraint between all the nodes
TSP generator

✔ Representation: list of 100 \((x, y)\) points on a \(500 \times 500\) map

✔ Objective is to satisfy a minimum and maximum distance constraint between all the nodes

✔ Fitness minimises the amount by which the current configuration is violating the set constraints
TSP instances

✔ Using GDBSCAN (Sander et al., 1998) we find that clusters are better defined using the evolutionary approach than when using a random uniform method of creating clusters.
TSP instances

- Using GDBSCAN (Sander et al., 1998) we find that clusters are better defined using the evolutionary approach than when using a random uniform method of creating clusters.

- Using the EA approach, for different number of clusters we generate 50 maps, and solve each one 50 times using Chained Lin-Kernighan (Applegate et al., 1999).
TSP instances

✔ Using GDBSCAN (Sander et al., 1998) we find that clusters are better defined using the evolutionary approach than when using a random uniform method of creating clusters.

✔ Using the EA approach, for different number of clusters we generate 50 maps, and solve each one 50 times using Chained Lin-Kernighan (Applegate et al., 1999).

✔ We repeat this experiment for maps with 50, 100, 150, and 200 nodes.
Difficulty in small TSPs

![Graph showing Lin-Kernighan steps vs. ratio of clusters (clusters/nodes) for different node counts: 200 nodes (red), 150 nodes (green), 100 nodes (blue), and 50 nodes (purple). The graph illustrates how the number of Lin-Kernighan steps increases with a decrease in the ratio of clusters.](image-url)
Difficulty in large TSPs

![Graph showing the relationship between Lin-Kernighan steps and the ratio of clusters (clusters/nodes) for different node counts (1500, 1250, and 1000 nodes).]
A short summary

✔ Some configurations of clusters seem to have a major impact on the search effort of Chained Lin-Kernighan
A short summary

✔ Some configurations of clusters seem to have a major impact on the search effort of Chained Lin-Kernighan

✔ For small instances (50–200 nodes), the peak of search effort required seems to align to clusters = 0.10 × nodes
A short summary

✔ Some configurations of clusters seem to have a major impact on the search effort of Chained Lin-Kernighan

✔ For small instances (50–200 nodes), the peak of search effort required seems to align to clusters = 0.10 × nodes

✔ This result does not hold for larger instances (> 1000 nodes)
A short summary

- Some configurations of clusters seem to have a major impact on the search effort of Chained Lin-Kernighan.
- For small instances (50–200 nodes), the peak of search effort required seems to align to clusters = 0.10 × nodes.
- This result does not hold for larger instances (> 1000 nodes).
- The difference in cluster sizes also influences the search effort.
Evolving hard to solve binary constraint satisfaction problem instances
Binary constraint satisfaction

✓ Binary constraint satisfaction is a clear and easy to define constraint satisfaction problem
Binary constraint satisfaction

- Binary constraint satisfaction is a clear and easy to define constraint satisfaction problem
- It was a popular problem for research in both the domains of constraint programming and artificial intelligence
Binary constraint satisfaction

✔ Binary constraint satisfaction is a clear and easy to define constraint satisfaction problem

✔ It was a popular problem for research in both the domains of constraint programming and artificial intelligence

✔ This has led to much knowledge about the location of difficult instances in relation to order parameters
Binary constraint satisfaction

✓ Binary constraint satisfaction is a clear and easy to define constraint satisfaction problem

✓ It was a popular problem for research in both the domains of constraint programming and artificial intelligence

✓ This has led to much knowledge about the location of difficult instances in relation to order parameters

✓ The concept of phase transitions became a popular research topic
Binary constraint satisfaction

- Binary constraint satisfaction is a clear and easy to define constraint satisfaction problem
- It was a popular problem for research in both the domains of constraint programming and artificial intelligence
- This has led to much knowledge about the location of difficult instances in relation to order parameters
- The concept of phase transitions became a popular research topic

In (van Hemert, 2003), an evolutionary algorithm is used to search the space of problem instances difficult to solve for a backtracking algorithm
The evolutionary algorithm

✔ Start with 30 randomly created problem instances that are easy to solve
✔ Run a generational EA with elitism for 200 generations
✔ Apply crossover and then mutation to create new problem instances
✔ Run a backtracking algorithm on the new problem instances
✎ Record search effort, solvability and tightness of every problem instance created using a complete solver
Converging on difficult instances

![Graph showing the number of conflict checks against tightness, with data points for solvable and unsolvable instances.]
A short summary

✔ In binary constraint satisfaction, using an evolutionary algorithm, we were able to successfully locate problem instances significantly more difficult than the standard spectrum scan,
A short summary

✔ In binary constraint satisfaction, using an evolutionary algorithm, we were able to successfully locate problem instances significantly more difficult than the standard spectrum scan,

✔ in a much faster way,
A short summary

✔ In binary constraint satisfaction, using an evolutionary algorithm, we were able to successfully locate problem instances significantly more difficult than the standard spectrum scan,

✔ in a much faster way,

✔ and without the need for knowing where these instances can be found.
Evolving hard to solve TSP instances
Main idea

Combine the two previous studies: evolve TSP problem instances against Chained Lin-Kernighan and study clustering as a property responsible for difficulty.
Main idea

✔ Combine the two previous studies: evolve TSP problem instances against Chained Lin-Kernighan and study clustering as a property responsible for difficulty

✔ Representation: a list of \((x, y)\) coordinates representing 100 cities on a map of \(400 \times 400\)
Main idea

✎ Combine the two previous studies: evolve TSP problem instances against Chained Lin-Kernighan and study clustering as a property responsible for difficulty

✔ Representation: a list of \((x, y)\) coordinates representing 100 cities on a map of \(400 \times 400\)

✔ Population size of 30, using a generational scheme with elitism
Describing the evolutionary process

- fitness = runtime of Chained Lin-Kernighan
- run Chained Lin-Kernighan on each instance
- create TSP instances uniform randomly
- using crossover & mutation create new TSP instances
- replace the population using elitism

maximum number of generations reached?
Increase in difficulty

![Box plot showing the distribution of Chained Lin-Kernighan search effort across different generations and TSP generator. The plot includes symbols indicating the mean, near outliers (<= 3.0 IQR), and far outliers (> 3.0 IQR).]
The evolutionary process in action

start animation
Clustering properties

✔ Use a clustering algorithm (GDBSCAN) to determine the amount of clusters in TSP instances

✔ Problem with clustering algorithms is that their result depend highly on input parameters

✔ Work-around is to sweep the parameter space and show the cumulative results
Clustering properties

① For every TSP problem instance in the set:
Clustering properties

① For every TSP problem instance in the set:
② Run GDBSCAN within a fixed and sensible range of its parameters
Clustering properties

① For every TSP problem instance in the set:
② Run GDBSCAN within a fixed and sensible range of its parameters
③ For every parameter, record how many clusters are found
Clustering properties

① For every TSP problem instance in the set:
② Run GDBSCAN within a fixed and sensible range of its parameters
③ For every parameter, record how many clusters are found
④ Count for every number of clusters the number of occurrences
Clustering properties

![Graph showing clustering properties with two lines: one for uniform random and one for evolutionary algorithm. The x-axis represents the number of clusters, and the y-axis represents the number of clusterings found. The graph shows a decreasing trend as the number of clusters increases.]
Distribution of segment lengths

① Given a TSP, create a tour using Chained Lin-Kernighan
Distribution of segment lengths

① Given a TSP, create a tour using Chained Lin-Kernighan

② Calculate the length of all the segments of that tour
Distribution of segment lengths

① Given a TSP, create a tour using Chained Lin-Kernighan
② Calculate the length of all the segments of that tour
③ Repeat this for every TSP instance in the set
Distribution of segment lengths

① Given a TSP, create a tour using Chained Lin-Kernighan
② Calculate the length of all the segments of that tour
③ Repeat this for every TSP instance in the set
④ Create histograms over the averaged results
Distribution of segment lengths

random

600 generations
Distribution of segment lengths

![Graph showing the distribution of segment lengths over different generations.](image)
Conclusions

✔ Using an evolutionary algorithm we can find TSP problems that are difficult to solve for Chained Lin-Kernighan
Conclusions

✔ Using an evolutionary algorithm we can find TSP problems that are difficult to solve for Chained Lin-Kernighan

✔ The layout of cities in a TSP highly influences the difficulty, in which clustering seems to play an integral role
Conclusions

✔ Using an evolutionary algorithm we can find TSP problems that are difficult to solve for Chained Lin-Kernighan

✔ The layout of cities in a TSP highly influences the difficulty, in which clustering seems to play an integral role

✔ My intuition: the trick is to get a distribution of nodes so that the solver has difficulty choosing whether to stay in a cluster (= short distance) or to go to another cluster (= long distance)
Future work

✔ Test more TSP solvers (Independence in BINCSP)
Future work

✔ Test more TSP solvers (Independence in BINCSP)
✔ Define more structural properties that may help explain difficulty of problem instances, specifically for TSP
Future work

✔ Test more TSP solvers (Independence in BINCSP)
✔ Define more structural properties that may help explain difficulty of problem instances, specifically for TSP
✔ Cooperation with Julia Handl, using her clustering algorithm to determine the amount of clustering in TSP instances
Future work

✔ Test more TSP solvers (Independence in BINCSP)
✔ Define more structural properties that may help explain difficulty of problem instances, specifically for TSP
✔ Cooperation with Julia Handl, using her clustering algorithm to determine the amount of clustering in TSP instances
✔ Evolve instances in other application areas: timetabling is first on the list with the help of Rhyd Lewis’ Grouping Genetic Algorithm
Thanks for your attention

- Slides and publications are available from
  http://google.com/search?q=jano+homepage
References


