Heuristic Approaches in Computational Geometry

A walk through years of the CG challenge

Jack Spalding-Jamieson

Independent

Attribution: About 2/3rds of this presentation are based on one given by Da Wei (David) Zheng.

NP-Hard Geometry Optimization Problems

Many geometry optimization problems are NP-hard (e.g. Euclidean TSP).

NP-Hard Geometry Optimization Problems

Many geometry optimization problems are NP-hard (e.g. Euclidean TSP).

NP-Hard $\neq$ impossible:

NP-Hard Geometry Optimization Problems

Many geometry optimization problems are NP-hard (e.g. Euclidean TSP).

NP-Hard $\neq$ impossible:

- Exponential time algorithms
    - SAT/SMT/ILP solvers

NP-Hard Geometry Optimization Problems

Many geometry optimization problems are NP-hard (e.g. Euclidean TSP).

NP-Hard $\neq$ impossible:

- Exponential time algorithms
  - SAT/SMT/ILP solvers
- Approximation algorithms
  - Rounding methods, greedy solutions, etc.

NP-Hard Geometry Optimization Problems

Many geometry optimization problems are NP-hard (e.g. Euclidean TSP).

NP-Hard $\neq$ impossible:

- Exponential time algorithms
  - SAT/SMT/ILP solvers
- Approximation algorithms
  - Rounding methods, greedy solutions, etc.
- **Heuristic algorithms**
  - Local search techniques

About the CG:SHOP challenge

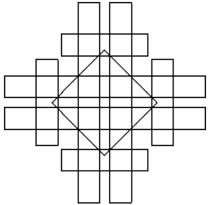
Computational Geometry: Solving Hard Optimization Problems (CG:SHOP) is an annual competition that is part of CG Week.

About the CG:SHOP challenge

Computational Geometry: Solving Hard Optimization Problems (CG:SHOP) is an annual competition that is part of CG Week.

- 2019-2022: Fairly "combinatorial" problems.
- 2023-2025: Solutions use rational coordinates.
- 2026: More "combinatorial" again.

This talk: 2020-2022.



Overview

1. Basic Framework: Local Search
2. CG:SHOP 2020
3. CG:SHOP 2021
4. CG:SHOP 2022

Overview

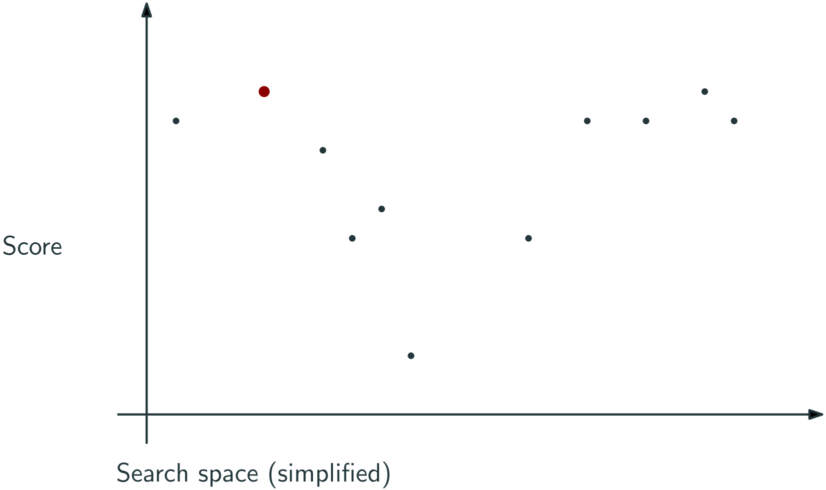
1. **Basic Framework: Local Search**
2. CG:SHOP 2020
3. CG:SHOP 2021
4. CG:SHOP 2022

Basic Framework: Local Search

1. Start at a decent feasible solution.
2. Repeatedly go to good "nearby" feasible solutions.

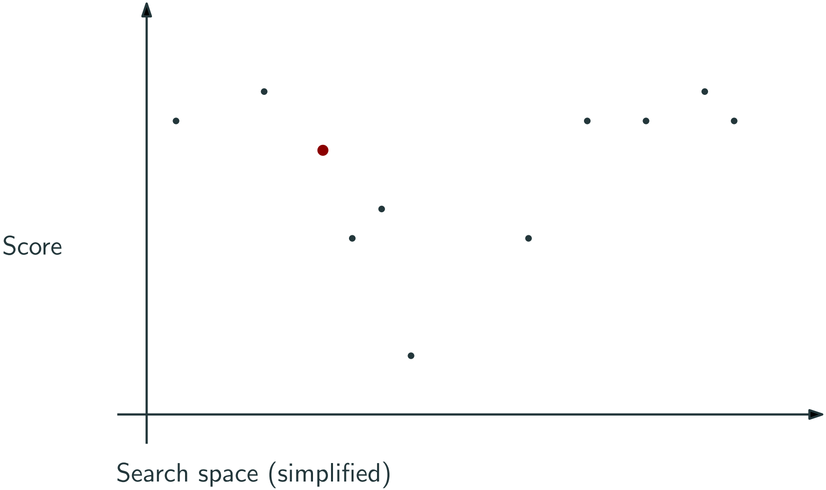
Basic Framework: Local Search

1. Start at a decent feasible solution.
2. Repeatedly go to good "nearby" feasible solutions.



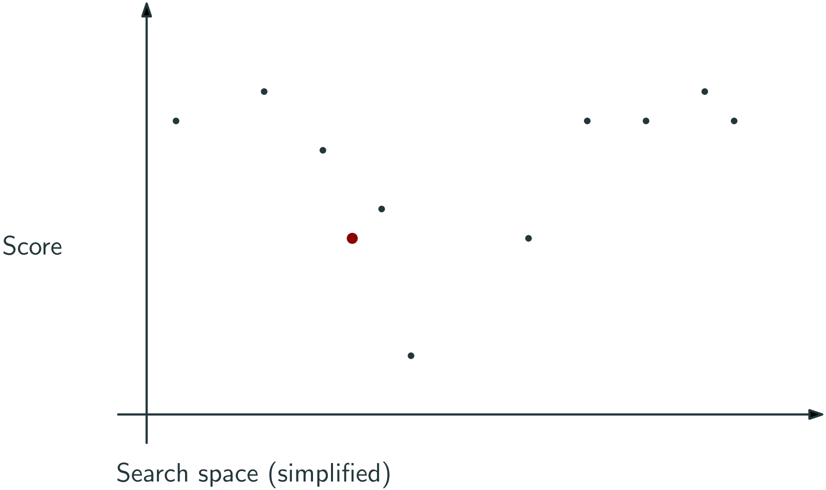
Basic Framework: Local Search

1. Start at a decent feasible solution.
2. Repeatedly go to good "nearby" feasible solutions.



Basic Framework: Local Search

1. Start at a decent feasible solution.
2. Repeatedly go to good "nearby" feasible solutions.



Elements needed for local search

- Need initial solution(s)

Elements needed for local search

- Need initial solution(s)
- Need to define “nearby” solutions (reconfiguration graph)

Elements needed for local search

- Need initial solution(s)
- Need to define "nearby" solutions (reconfiguration graph)
- How to decide which nearby solution to choose?

Elements needed for local search

- Need initial solution(s)
- Need to define “nearby” solutions (reconfiguration graph)
- How to decide which nearby solution to choose?
    - Greedy choice: Look at all, move if it’s better

Variations of Local Search

1. Generating different initialization to start from

Variations of Local Search

1. Generating different initialization to start from
    - Problem-specific ideas

Variations of Local Search

1. Generating different initialization to start from
    - Problem-specific ideas
    - Genetic algorithms that "merge" feasible solutions

Variations of Local Search

1. Generating different initialization to start from
    - Problem-specific ideas
    - Genetic algorithms that “merge” feasible solutions
2. Improving on greedy local search

Variations of Local Search

1. Generating different initialization to start from
   - Problem-specific ideas
   - Genetic algorithms that “merge” feasible solutions
2. Improving on greedy local search
   - Randomness + Restarts (repeated trials give different results)

Variations of Local Search

1. Generating different initialization to start from
   - Problem-specific ideas
   - Genetic algorithms that “merge” feasible solutions
2. Improving on greedy local search
   - Randomness + Restarts (repeated trials give different results)
   - Simulated Annealing (sometimes allow bad moves)

Variations of Local Search

1. Generating different initialization to start from
    - Problem-specific ideas
    - Genetic algorithms that "merge" feasible solutions
2. Improving on greedy local search
    - Randomness + Restarts (repeated trials give different results)
    - Simulated Annealing (sometimes allow bad moves)
    - k-opt (make k changes at once)

Variations of Local Search

1. Generating different initialization to start from
   - Problem-specific ideas
   - Genetic algorithms that "merge" feasible solutions
2. Improving on greedy local search
   - Randomness + Restarts (repeated trials give different results)
   - Simulated Annealing (sometimes allow bad moves)
   - k-opt (make k changes at once)
   - Conflict optimization (go outside the feasible region)

Overview

1. Basic Framework: Local Search
2. **CG:SHOP 2020**
3. CG:SHOP 2021
4. CG:SHOP 2022

CG:SHOP2020 - Team UBC

Team members: Jack Spalding-Jamieson, Brandon Zhang, and Da Wei (David) Zheng.

CG:SHOP2020 - Preview of Results

Winners and Accepted Papers — CG Challenge

1 **Team UBC, Canada:** Da Wei Zheng, Jack Spalding-Jamieson, Brandon Zhang

Total score	Best solutions (from 346 instances)	Unique best solutions
175.172880	209	11

All members of this team were students, so they also won the Junior Category.

2 **Team Haute-Alsace, France:** Laurent Moalic, Dominique Schmitt, Julien Lepagnot, Julien Kritter

Total score	Best solutions (from 346 instances)	Unique best solutions
175.130597	297	126

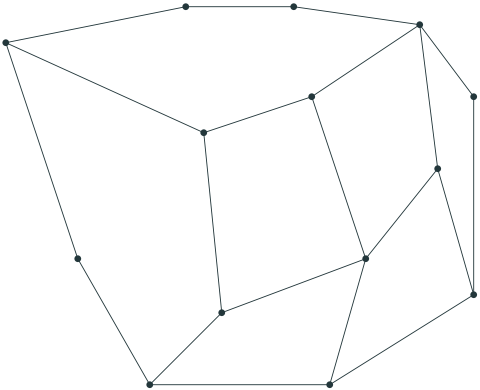
3 **Team Salzburg, Austria:** Günther Eder, Martin Held, Stefan de Lorenzo, Peter Palfrader

Total score	Best solutions (from 346 instances)	Unique best solutions
175.040207	187	0

CG:SHOP2020 - Problem Statement

Input: n points in the plane (S).

Output: A partitition of their convex hull into convex faces whose vertex set is S.



CG:SHOP2020 - Approach - Overview

- Small instances (< 100 vertices): solved exactly with MAXSAT formulation.

CG:SHOP2020 - Approach - Overview

- Small instances (< 100 vertices): solved exactly with MAXSAT formulation.
- Large instances: Local search method

CG:SHOP2020 - Approach - Overview

- Small instances (< 100 vertices): solved exactly with MAXSAT formulation.
- Large instances: Local search method
    1. **Initialization**: Delauney triangulation

CG:SHOP2020 - Approach - Overview

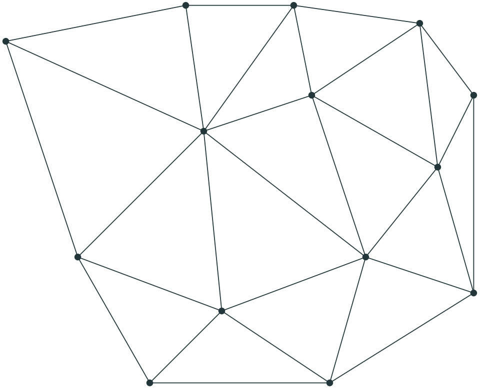
- Small instances (< 100 vertices): solved exactly with MAXSAT formulation.
- Large instances: Local search method
  1. **Initialization**: Delauney triangulation
  2. **Local search move type #1**: Remove edges
  3. **Local search move type #2**: Rotate edges

CG:SHOP2020 - Approach - Overview

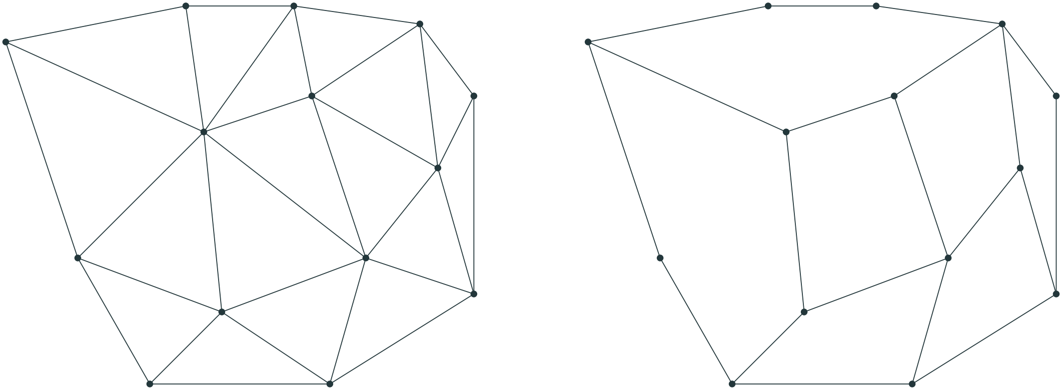
- Small instances (< 100 vertices): solved exactly with MAXSAT formulation.
- Large instances: Local search method
    1. **Initialization**: Delauney triangulation
    2. **Local search move type #1**: Remove edges
    3. **Local search move type #2**: Rotate edges

CG:SHOP2020 - Initialization

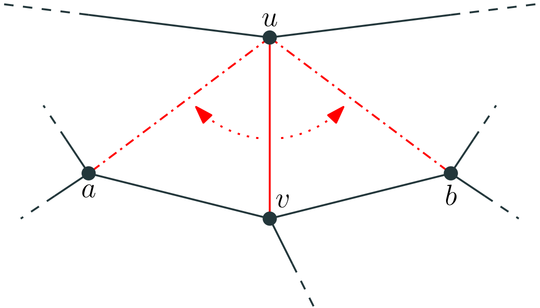
- The Delauney triangulation was used as the starting point.



- Some edges can be removed while keeping faces convex.
- If there are many edges, remove edges in a random order.

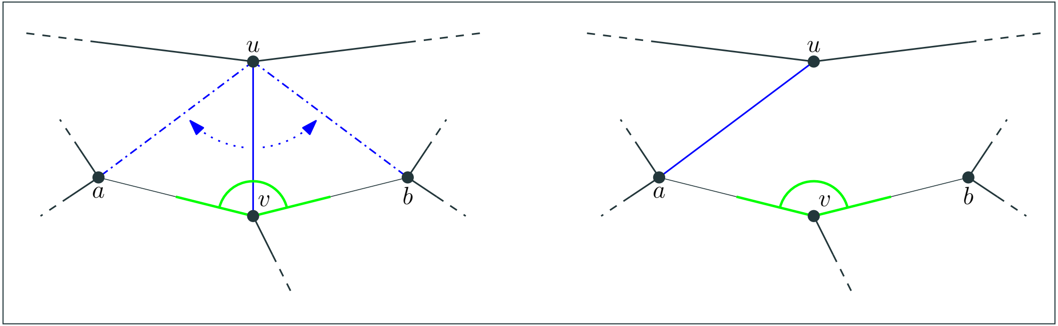


- Half-edges can often be rotated, so long as both incident faces remain convex.



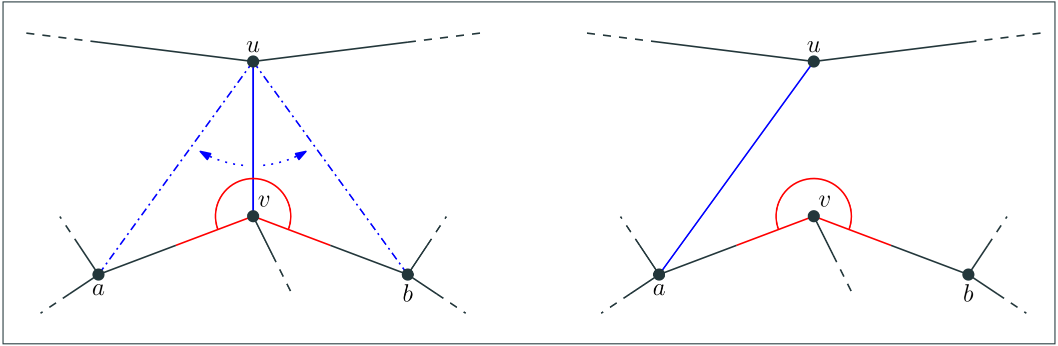
CG:SHOP2020 - Rotation Moves (2)

- Half-edges can be rotated so long as no angles become reflex.
- In this example, convexity is preserved:



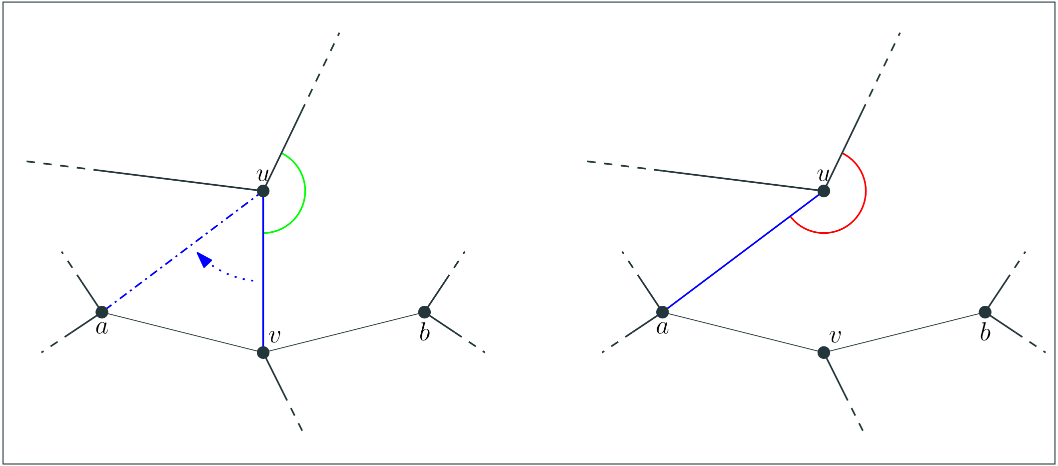
CG:SHOP2020 - Rotation Moves (3)

- In this example, the bottom angle becomes reflexive:



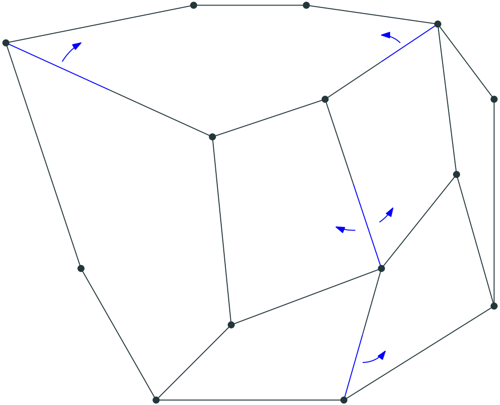
CG:SHOP2020 - Rotation Moves (4)

- In this example, the top angle becomes reflexive:



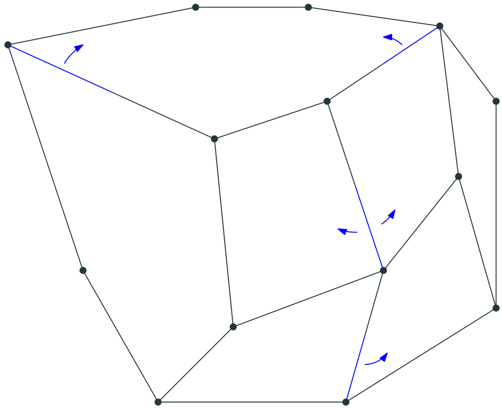
CG:SHOP2020 - Rotation Moves (5)

- There may be many edges that can be rotated.



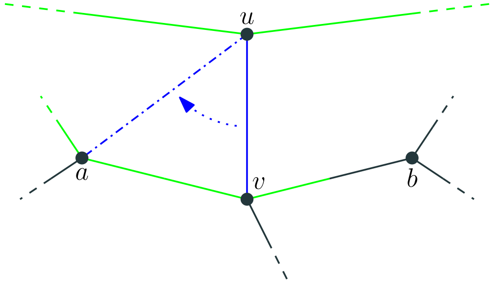
CG:SHOP2020 - Rotation Moves (5)

- There may be many edges that can be rotated.
- Choose a random one and do a random walk through the reconfiguration space of convex partitions.



CG:SHOP2020 - Approach Step 3 - Keeping Track of Half-Edges

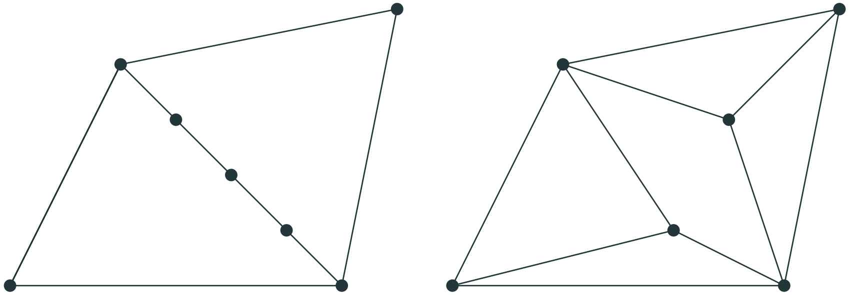
- We keep an up-to-date list of half-edges that can be rotated.
- There is a constant number of different half-edges that need to be updated.



In green: the half-edges that need to be updated.

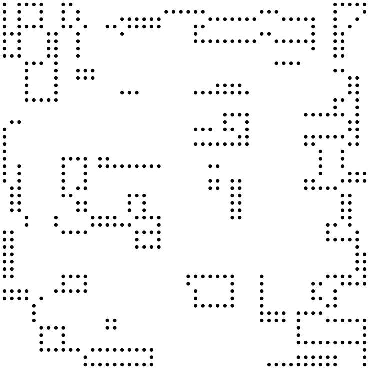
For a general position point set, all (interior) vertices have degree ≥ 3.

Can do better with collinear points.



CG:SHOP2020 - Extra instances

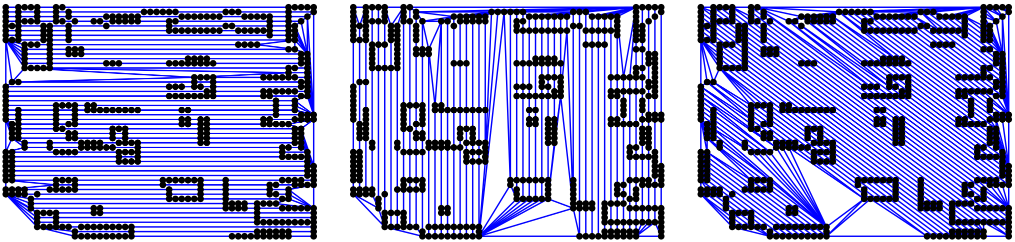
The organizers realized this as well and added many instances called `rop` and `ortho-rect` that looked like this:



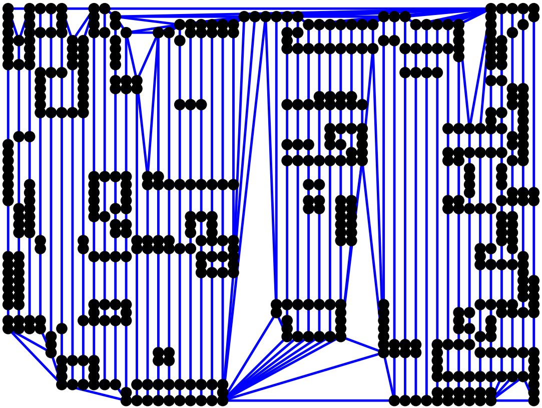
Starting with Delauney triangulation did ok, but we can do better.

CG:SHOP2020 - Initialization (v2)

- Joining collinear points together created degree 2 vertices. This is good.
- For these instances we joined points sharing the same x, y, or the same slope, then joined the end points together in a way that creates a convex partition.

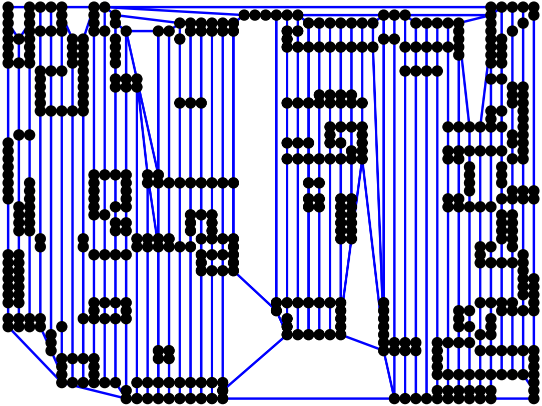


- Afterwards, the local search would remove extraneous edges.



Before running local search.

- Afterwards, the local search would remove extraneous edges.



After running local search.

CG:SHOP2020 - Results - Hardware

Ran on some UBC servers.

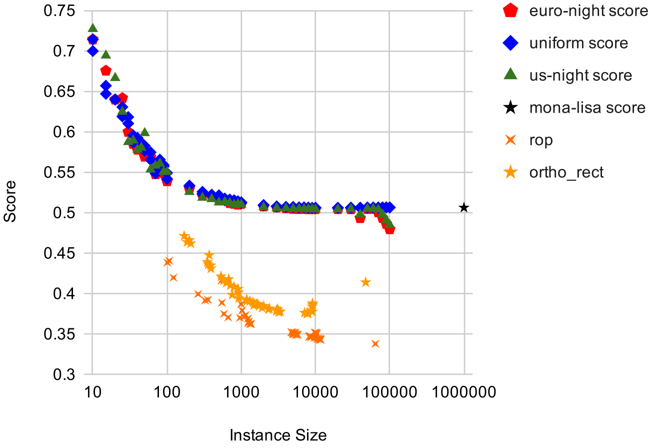
- Ran local search continuously for about 16 days.
- **Consumed approximately 1.5 years of CPU time.**

Instance `london-0000040` with 40 points, 469 iterations, and 64 final edges.

Animation: london-0000040 local search demo (469 iterations)

CG:SHOP2020 - Results - Visualization

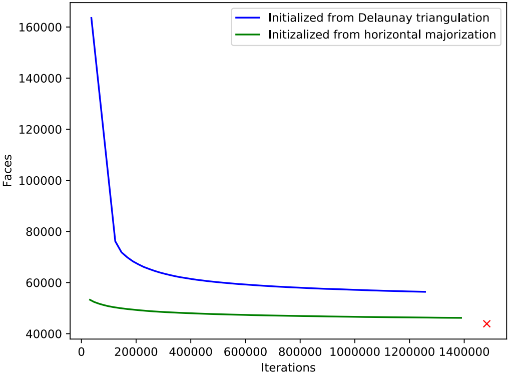
- Most of the instance groups and our respective scores are plotted on the table below:



CG:SHOP2020 - Results - Large `euro-night` instance

- For the 100 000 point `euro-night` instance, y-coordinates were in $[0, 57\,598] \implies$ many shared y-values.

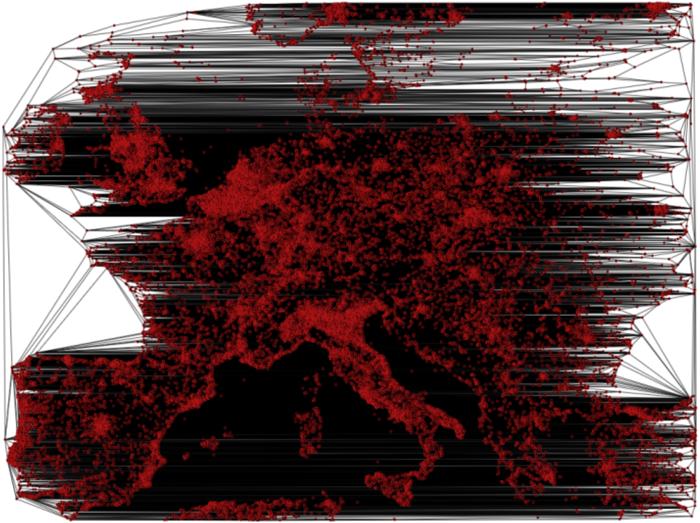
- For the 100 000 point `euro-night` instance, y-coordinates were in $[0, 57\,598] \implies$ many shared y-values.



(40,000 point euro-night instance)

CG:SHOP2020 - Results - Large `euro-night` instance

Optimized large `euro-night` instance with 40 000 points from horizontally-joined initialization.



CG:SHOP2020 - Results

Total: 346 instances

Winners and Accepted Papers — CG Challenge

1 **Team UBC, Canada:** Da Wei Zheng, Jack Spalding-Jamieson, Brandon Zhang

Total score	Best solutions (from 346 instances)	Unique best solutions
175.172880	209	11

All members of this team were students, so they also won the Junior Category.

2 **Team Haute-Alsace, France:** Laurent Moalic, Dominique Schmitt, Julien Lepagnot, Julien Kritter

Total score	Best solutions (from 346 instances)	Unique best solutions
175.130597	297	126

3 **Team Salzburg, Austria:** Günther Eder, Martin Held, Stefan de Lorenzo, Peter Palfrader

Total score	Best solutions (from 346 instances)	Unique best solutions
175.040207	187	0

CG:SHOP2020 - Techniques by the Other Teams

- Team Haute-Alsace
  - Used a memetic approach to take "good polygons" from two good solutions, then triangulated rest.
- Team Salzburg
  - Tried implementing known 3-Approximations
  - Used recursion from high degree vertices instead of doing globally random flipping

Overview

1. Basic Framework: Local Search
2. CG:SHOP 2020
3. **CG:SHOP 2021**
4. CG:SHOP 2022

CG:SHOP2021 - Team `gitastrophe`

All of us have left UBC. Paul (also UBC alumnus) joined our team, making us quite diverse.

Team members:

- Paul Liu (Stanford)
- Jack Spalding-Jamieson (Waterloo)
- Brandon Zhang (Working)
- Da Wei (David) Zheng (UIUC)

CG:SHOP2021 - Preview of Results

MAX

Rank	Team	Junior team	Score MAX	Score SUM	# Best solutions (MAX)	# Best solutions (SUM)
1	Shadoks		202.9375	180.4952611231	202	0
2	UNIST		174.0180514765	191.7893810645	14	120
3	gitastrophe	✓	159.5472362028	198.494347968	24	57

SUM

Rank	Team	Junior team	Score MAX	Score SUM	# Best solutions (MAX)	# Best solutions (SUM)
1	gitastrophe	✓	159.5472362028	198.494347968	24	57
2	UNIST		174.0180514765	191.7893810645	14	120
3	Shadoks		202.9375	180.4952611231	202	0

CG:SHOP2021 - Problem Statement

Given a set R of n robots, find a collision-free set of parallel motions for unit-square robots in the square grid $\mathbb{Z}^2$ that minimizes total distance travelled or minimizes makespan (different problem categories).

Animation: Robot motion planning demo

Animation: Roomba robot motion demo

Creative Commons attributions: "Doomba" model by PolyDucky, "Cardboard Box" model by Agustín Hönnun.

CG:SHOP2021 - Approach - Overview

Another local search approach. Two main components:

CG:SHOP2021 - Approach - Overview

Another local search approach. Two main components:

1. Initialization

CG:SHOP2021 - Approach - Overview

Another local search approach. Two main components:

1. Initialization
    - Basic initializations
    - Improved initializations

CG:SHOP2021 - Approach - Overview

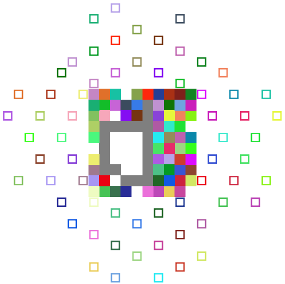
Another local search approach. Two main components:

1. Initialization
    - Basic initializations
    - Improved initializations
2. Optimization

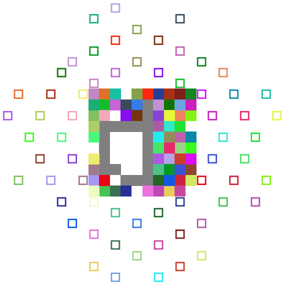
CG:SHOP2021 - Approach - Overview

Another local search approach. Two main components:

1. Initialization
    - Basic initializations
    - Improved initializations
2. Optimization
    - Basic greedy local search (1-opt)
    - k-opt
    - Algorithm engineering



Instances were boundaryless: Robots can move far away.

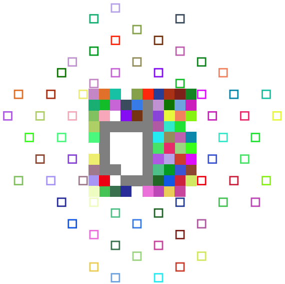


Instances were boundaryless: Robots can move far away.

We used the following algorithm:

1. Compute a set of far away intermediate locations

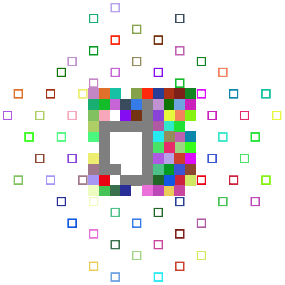
CG:SHOP2021 - Initialization



Instances were boundaryless: Robots can move far away.

We used the following algorithm:

1. Compute a set of far away intermediate locations
2. Compute min-cost matching of robots start and end locations to intermediate locations

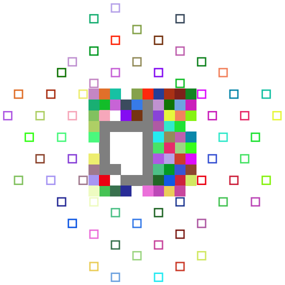


Instances were boundaryless: Robots can move far away.

We used the following algorithm:

1. Compute a set of far away intermediate locations
2. Compute min-cost matching of robots start and end locations to intermediate locations
3. Route robots from start locations to intermediate positions by order of start location depth

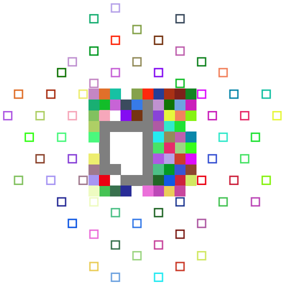
CG:SHOP2021 - Initialization



Instances were boundaryless: Robots can move far away.

We used the following algorithm:

1. Compute a set of far away intermediate locations
2. Compute min-cost matching of robots start and end locations to intermediate locations
3. Route robots from start locations to intermediate positions by order of start location depth
4. Route robots from intermediate locations to target locations in order of end location depth (guaranteed by previous step)



Instances were boundaryless: Robots can move far away.

We used the following algorithm:

1. Compute a set of far away intermediate locations
2. Compute min-cost matching of robots start and end locations to intermediate locations
3. Route robots from start locations to intermediate positions by order of start location depth
4. Route robots from **start locations** to target locations in order of end location depth (guaranteed by previous step)

CG:SHOP2021 - Initialization - Video

Videos of robot movement as robots are routed from start locations to target locations

Animation: Robot initialization routing

CG:SHOP2021 - Optimization - Basic Greedy Local Optimization

Given a feasible solution S:

- Pick a robot r and remove its path from S

CG:SHOP2021 - Optimization - Basic Greedy Local Optimization

Given a feasible solution S:

- Pick a robot r and remove its path from S
- Compute a new shortest path for r in the grid-time graph, respecting the other robots

CG:SHOP2021 - Optimization - Basic Greedy Local Optimization

Given a feasible solution S:

- Pick a robot r and remove its path from S
- Compute a new shortest path for r in the grid-time graph, respecting the other robots
- Repeat until no robot can shorten its path

CG:SHOP2021 - Optimization - 1-opt

Animation: 1-opt optimization demo

The 1-opt approach gets stuck in local minima easily.

The 1-opt approach gets stuck in local minima easily.
Traditional *k*-opt (optimally solving for *k* robots at once) would be better, but is slow.

The 1-opt approach gets stuck in local minima easily.
Traditional k-opt (optimally solving for k robots at once) would be better, but is slow.
Instead, use heuristic k-opt:

- Pick k robots $\{r_1, ..., r_k\} \in R$ and remove their paths from S
- Compute a new shortest path for r_i in the grid-time graph, respecting the other robots
- Repeat many times

Animation: k-opt optimization demo

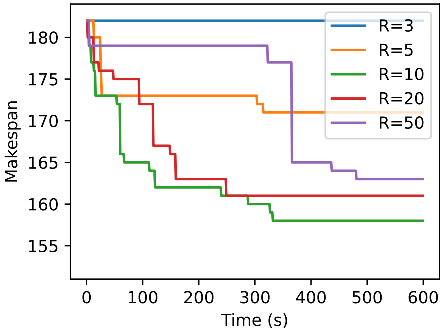
How do we make our optimization iterations run quickly and efficiently?

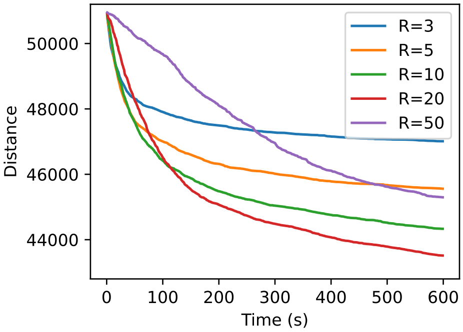
- To find paths, use A* with Manhattan distance as heuristic.
- Limit path-finding algorithm to explore locally around original path for some radius R.
- Choose k in the k-opt to balance runtime vs improvement.

CG:SHOP2021 - Optimization - Adjusting R

Makespan and distance plots as R varied. k was kept fixed at 7.

Final challenge makespan score: 126. Final distance score: 43 437.

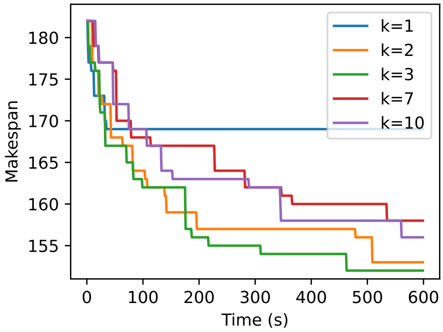


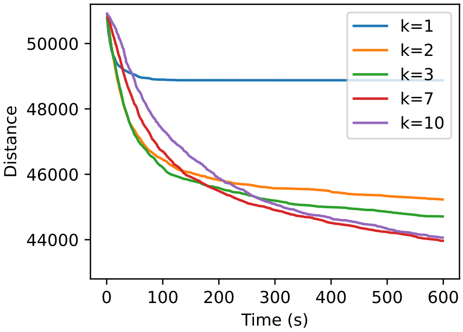


CG:SHOP2021 - Optimization - Adjusting k

Makespan and distance plots as k varied. R was kept fixed at 20.

Final challenge makespan score: 126. Final distance score: 43 437.





CG:SHOP2021 - Results

Total: 203 instances

MAX

Rank	Team	Junior team	Score MAX	Score SUM	# Best solutions (MAX)	# Best solutions (SUM)
1	Shadoks		202.9375	180.4952611231	202	0
2	UNIST		174.0180514765	191.7893810645	14	120
3	gitastrophe	✓	159.5472362028	198.494347968	24	57

SUM

Rank	Team	Junior team	Score MAX	Score SUM	# Best solutions (MAX)	# Best solutions (SUM)
1	gitastrophe	✓	159.5472362028	198.494347968	24	57
2	UNIST		174.0180514765	191.7893810645	14	120
3	Shadoks		202.9375	180.4952611231	202	0

CG:SHOP2021 - Results

MAX

Rank	Team	Junior team	Score MAX	Score SUM	# Best solutions (MAX)	# Best solutions (SUM)
1	Shadoks		202.9375	180.4952611231	202	0
2	UNIST		174.0180514765	191.7893810645	14	120
3	gitastrophe	✓	159.5472362028	198.494347968	24	57

SUM

Rank	Team	Junior team	Score MAX	Score SUM	# Best solutions (MAX)	# Best solutions (SUM)
1	gitastrophe	✓	159.5472362028	198.494347968	24	57
2	UNIST		174.0180514765	191.7893810645	14	120
3	Shadoks		202.9375	180.4952611231	202	0

- Team Shadoks' approach was by far the best: They did not even try to optimize for SUM.
- Two reasons: Smarter initialization + a new local search technique named conflict optimization...

Overview

1. Basic Framework: Local Search
2. CG:SHOP 2020
3. CG:SHOP 2021
4. **CG:SHOP 2022**

CG:SHOP2022 - Team gitastrophe

Team members:

- Jack Spalding-Jamieson (Waterloo)
- Brandon Zhang (Working)
- Da Wei (David) Zheng (UIUC)

CG:SHOP2022 - Standings Preview

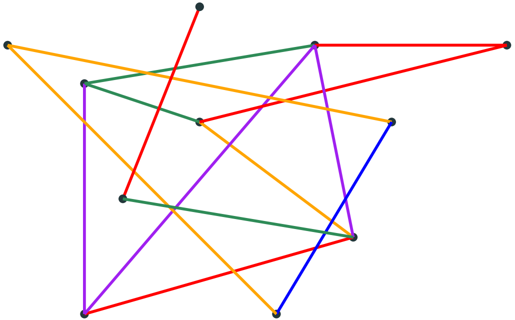
Rank	Team	Junior team	Score
1	Shadoks		225.0
2	gitastrophe	✔	217.4857474577237
3	LASAOFOOFUBESTINNRRALLDECA		211.80303248033107
4	TU Wien	✔	195.9666148217582

Problem Statement

Input: A straight-line drawing of a graph $G = (V, E)$.

Output: A partition of G into plane subgraphs (COLOURS).

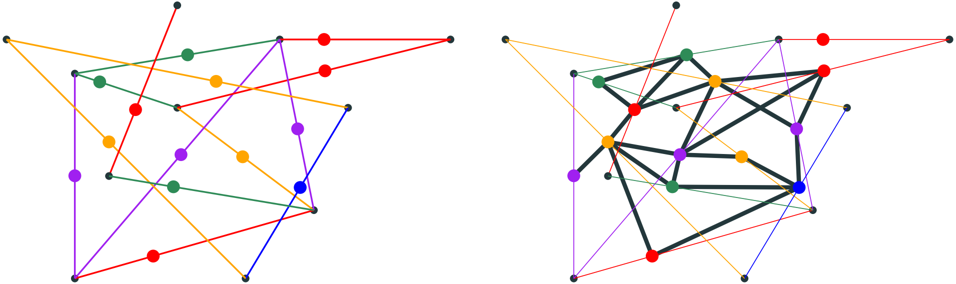
Goal: Minimize the number of subgraphs.



CG:SHOP2022 - Reduction to Vertex Colouring

Construct a conflict graph G':

- $V(G') := E(G)$
- $E(G') :=$ the pairwise intersections of the straight-line edges.



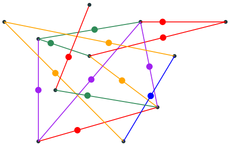
Approach - Overview

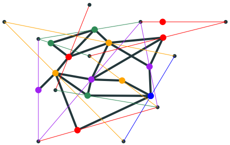
Two main components:

1. (Very basic) Initialization
2. Local search optimization
    - Conflict Optimization
    - Alternative heuristics

Initialization (Very Simple)

- Start with all the edges uncoloured.
- Loop through the straight-line edges in some order, colour them greedily.





Good orderings:

- Sorted by slope.
- Sorted by decreasing order of degree in the conflict graph (Welsh and Powell. 1967).

Optimization - Conflict Optimization

Conflict-Based Local Search/Conflict Optimization

- Initially used by was used by team Shadoks in CG:SHOP 2021 (Crombez et al. 2021).
- Very broad idea, can be applied this year as well.

Optimization - Conflict Optimization

Conflict-Based Local Search/Conflict Optimization

- Initially used by was used by team Shadoks in CG:SHOP 2021 (Crombez et al. 2021).
- Very broad idea, can be applied this year as well.

Main idea:

- Eliminate an entire colour class **without** giving the edges a new colour.

Optimization - Conflict Optimization

Conflict-Based Local Search/Conflict Optimization

- Initially used by was used by team Shadoks in CG:SHOP 2021 (Crombez et al. 2021).
- Very broad idea, can be applied this year as well.

Main idea:

- Eliminate an entire colour class **without** giving the edges a new colour.
- Try to colour each uncoloured edge while minimizing a **conflict score** (a heuristic).

Optimization - Conflict Optimization

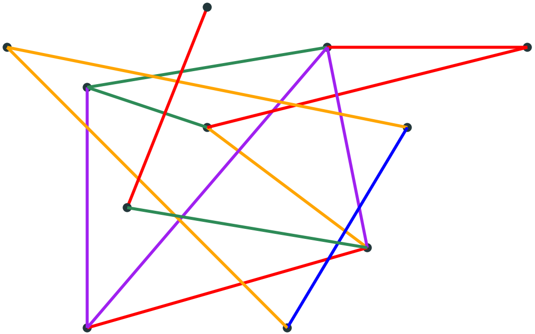
Conflict-Based Local Search/Conflict Optimization

- Initially used by was used by team Shadoks in CG:SHOP 2021 (Crombez et al. 2021).
- Very broad idea, can be applied this year as well.

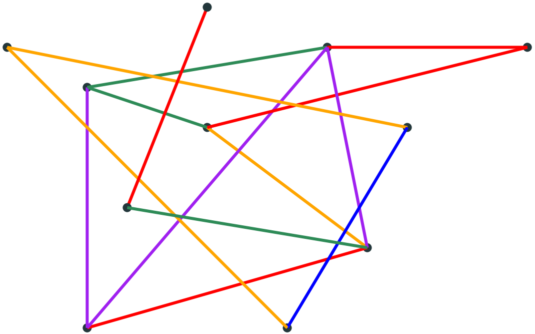
Main idea:

- Eliminate an entire colour class **without** giving the edges a new colour.
- Try to colour each uncoloured edge while minimizing a **conflict score** (a heuristic).
- Uncolour the conflicting edges when colouring an edge.

Optimization Example (1)

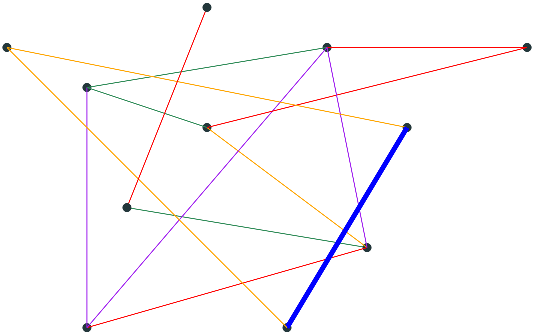


Optimization Example (4)



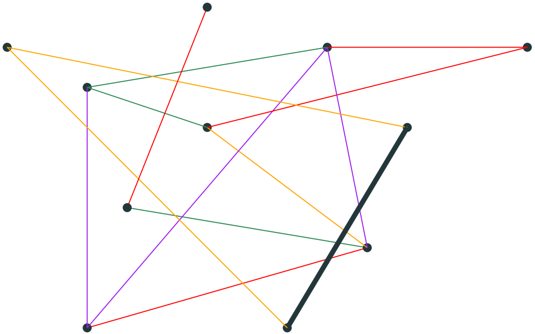
Step 1: Eliminate a Colour

Optimization Example (5)



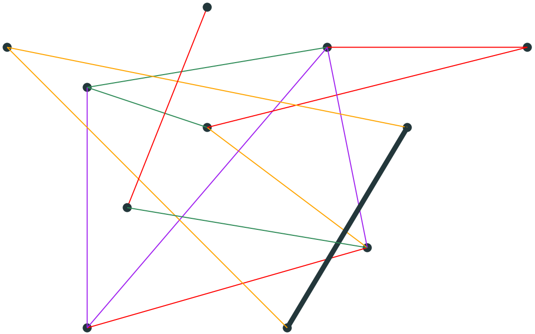
Choose blue to eliminate

Optimization Example (6)



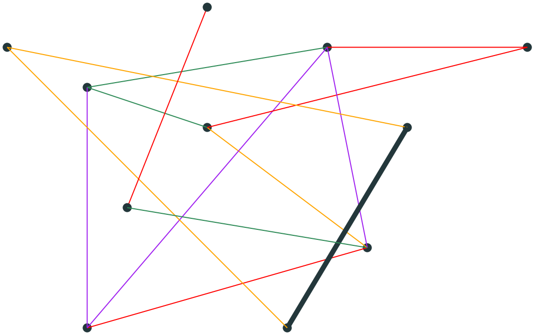
Uncolour all blue edges

Optimization Example (7)



Look at an uncoloured edge

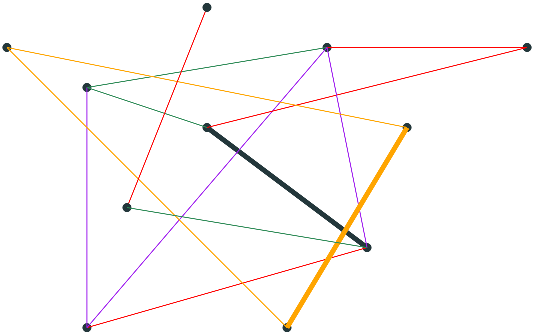
Optimization Example (8)



Pick a new colour according to a “conflict score” heuristic

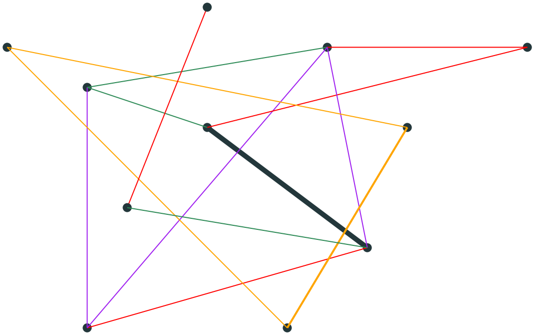
Choose orange

Optimization Example (9)



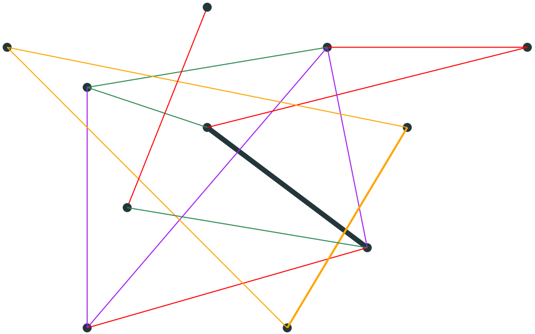
Colour the edge and uncolour all conflicting edges

Optimization Example (10)



If there is one: Look at an uncoloured edge

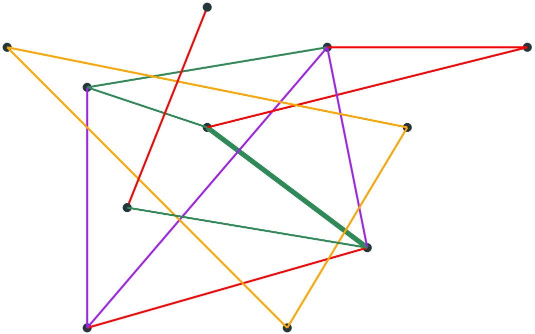
Optimization Example (11)



Pick a new colour according to a "conflict score" heuristic

Choose green

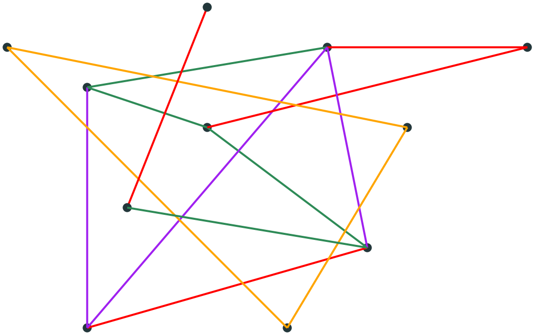
Optimization Example (12)



Pick a new colour according to a “conflict score” heuristic

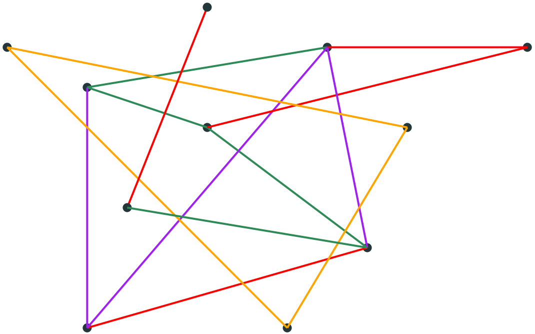
Choose green

Optimization Example (13)



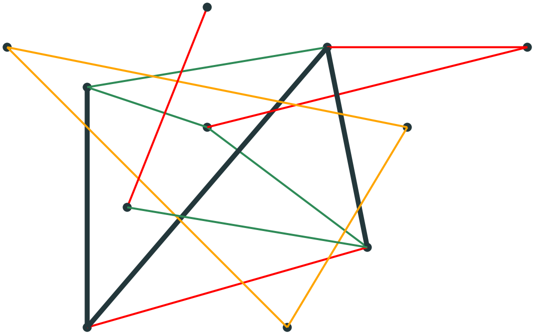
One colour down!

Optimization Example (14)



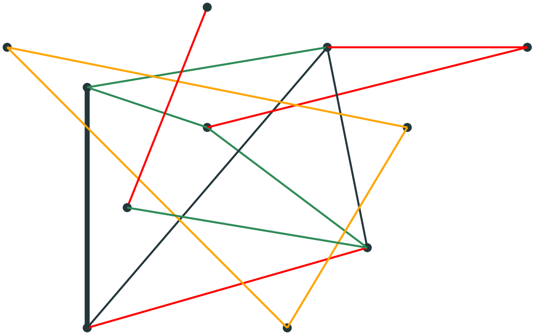
Let's try to eliminate another one: Purple

Optimization Example (15)



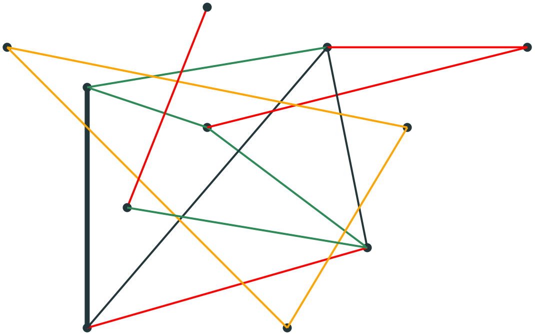
Uncolour all the purple edges

Optimization Example (16)



Look at an uncoloured edge

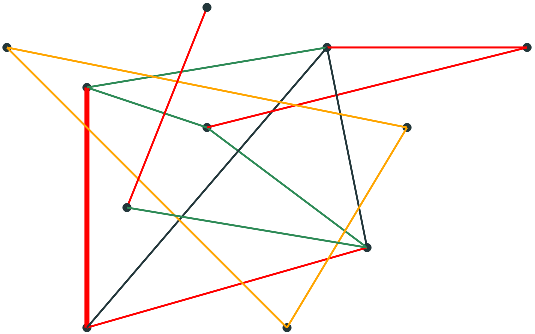
Optimization Example (17)



Choose a colour based on a "conflict score"

Choose red

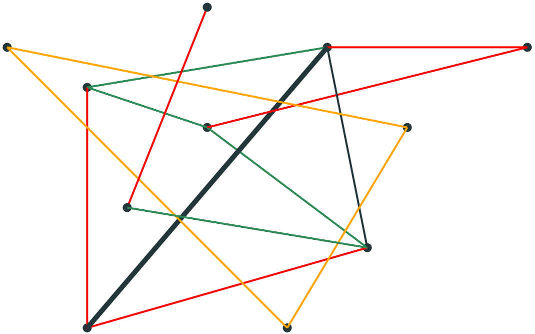
Optimization Example (18)



Colour the edge red and uncolour any conflicting edges

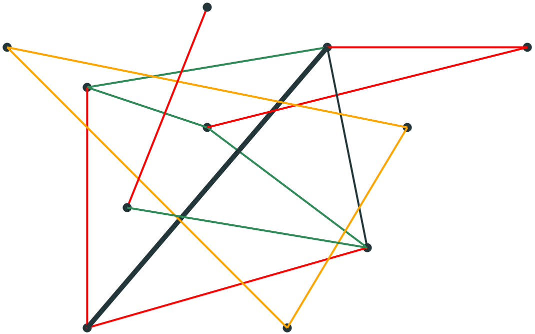
(none in this case)

Optimization Example (19)



Look at another uncoloured edge

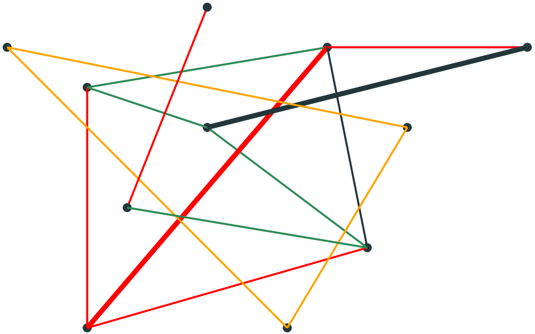
Optimization Example (20)



Choose a colour for it based on conflict score

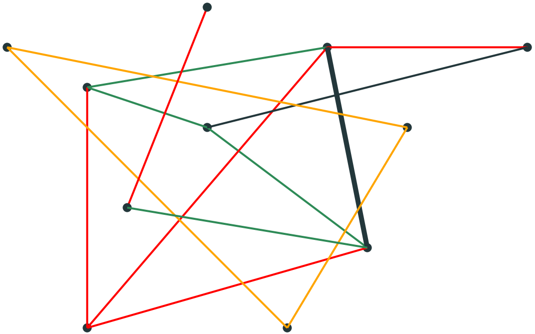
Choose red

Optimization Example (21)



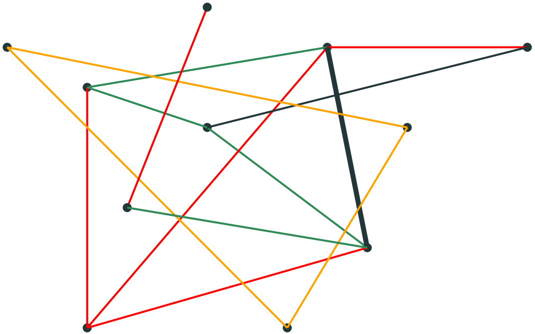
Colour the edge red and uncolour any conflicting edges

Optimization Example (22)



Look at an uncoloured edge

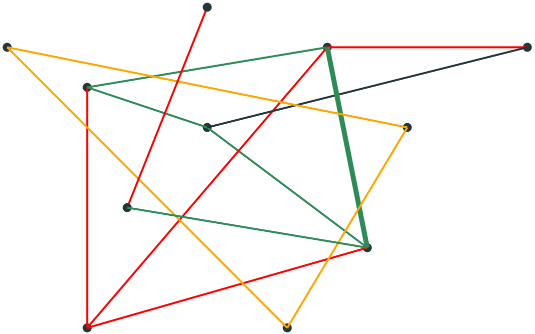
Optimization Example (23)



Choose a colour based on a "conflict score"

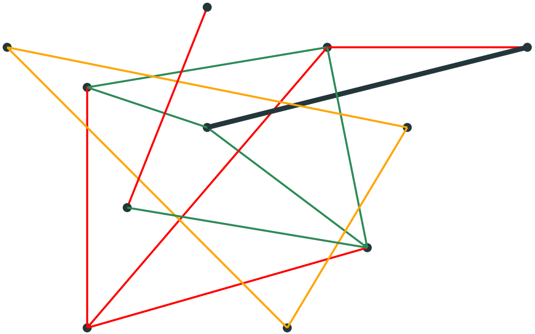
Choose green

Optimization Example (24)



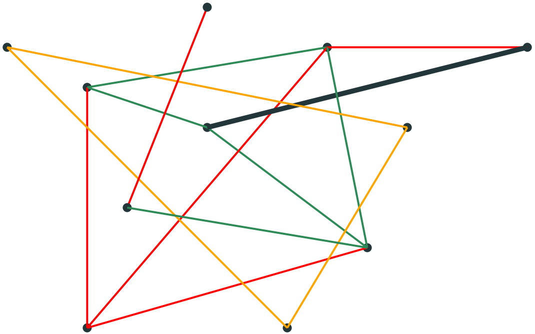
Colour the edge green and uncolour any conflicting edges

Optimization Example (25)



Look at an uncoloured edge

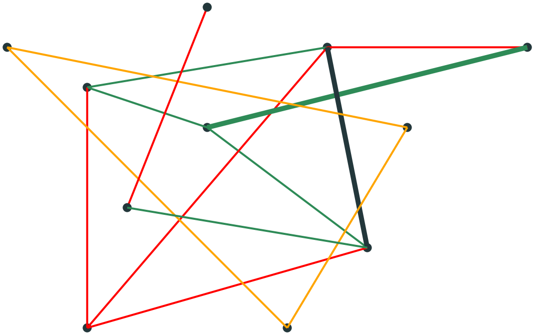
Optimization Example (26)



Choose a colour based on a “conflict score”

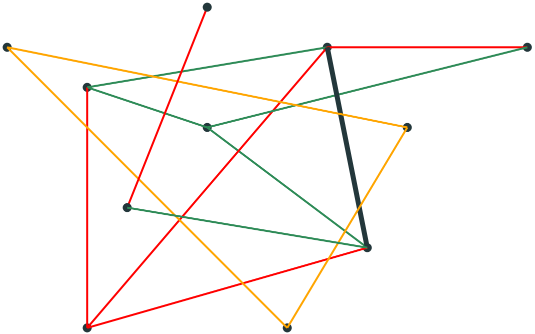
Choose green

Optimization Example (27)



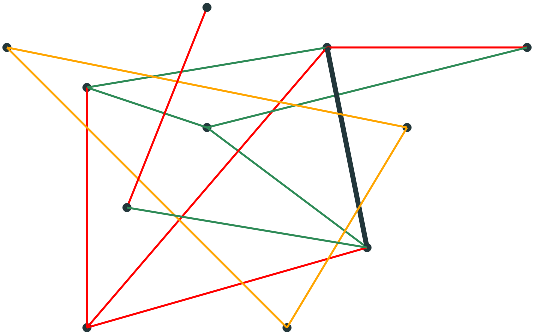
Colour the edge green and uncolour any conflicting edges

Optimization Example (28)



Look at an uncoloured edge

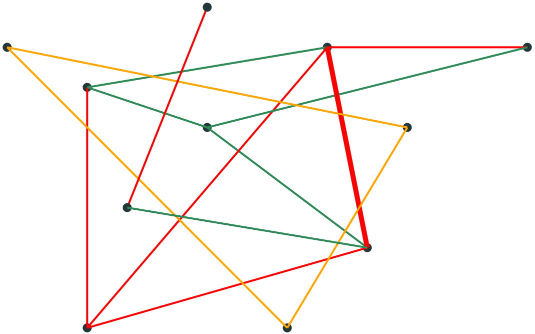
Optimization Example (29)



Choose a colour based on a "conflict score"

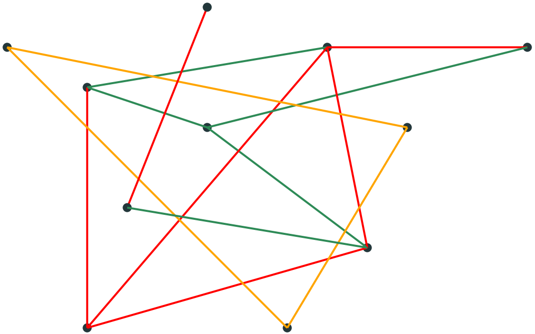
Choose red

Optimization Example (30)



Colour the edge red and uncolour any conflicting edges

Optimization Example (31)



Done!

Optimization - Heuristics

Conflict score:

$$\sum_{\substack{e' \in C_i \\ (e',e) \in E(G')}} 1 + q(e')^2$$

$q(e')$ is the number of times e' was uncoloured during the current "infeasible" stage.

Conflict score:

$$\sum_{\substack{e' \in C_i \\ (e',e) \in E(G')}} 1 + q(e')^2$$

$q(e')$ is the number of times e' was uncoloured during the current "infeasible" stage.

Alternative:

$$\sum_{\substack{e' \in C_i \\ (e',e) \in E(G')}} 1$$

Are we using the geometry?

Initialization stage: Yes, explicitly.

Are we using the geometry?

Initialization stage: Yes, explicitly.

Conflict optimization stage: Kind of. . . this algorithm seems to perform best on geometric data.

Comparison to Standard Vertex Colouring Approaches

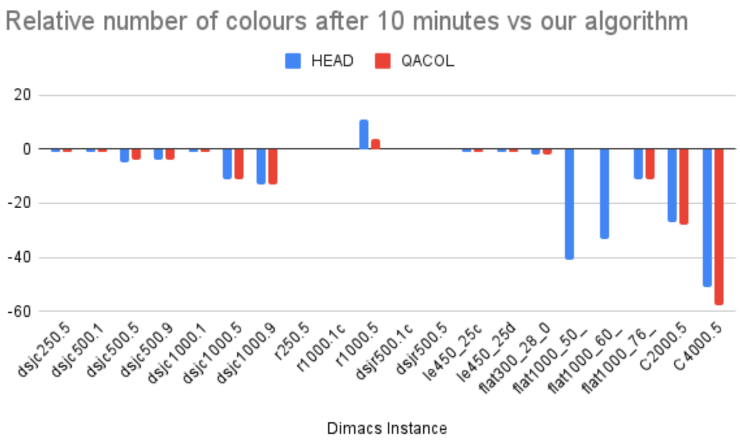


Figure 1: 10 minutes of our algorithm versus standard approaches on dimacs graph colouring instances.

Thank you for listening

Fin.