Shadoks Approach to Minimum Partition into Plane Subgraphs

Loïc Crombez – LIMOS, Université Clermont Auvergne

Guilherme D. da Fonseca – LIS, Aix-Marseille Université

Yan Gerard – LIMOS, Université Clermont Auvergne

Aldo Gonzalez-Lorenzo – LIS, Aix-Marseille Université

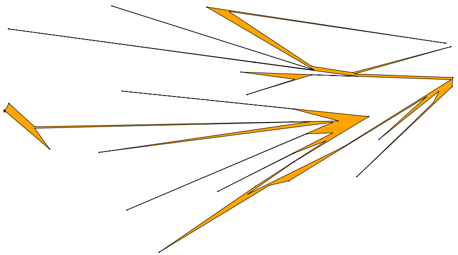
CG:SHOP 2022

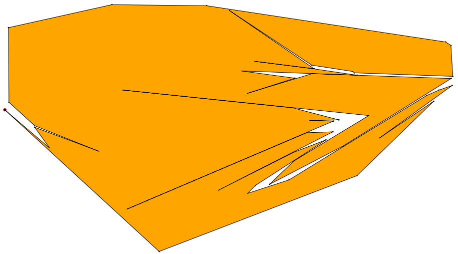
CG:SHOP Competition

- Part of SoCG (International Symposium on Computational Geometry)
- 4th year, started in 2018
- Hard geometric optimization problems
- Different problem each year
- ~ 200 instances given
- ~ 3 months to compute solutions
- Send our solutions (not the code)
- Score based on the quality of the solutions
- Top teams invited to publish in SoCG proceedings and ACM Journal of Experimental Algorithmics
- This talk is about the 2022 competition, but let's look at previous years...

Minimum (or Maximum) Area Polygon:

- Input: A set of points $S \subset \mathbb{R}^2$
- Output: A simple polygon with vertex set S
- Goal: Minimize (or maximize) the area



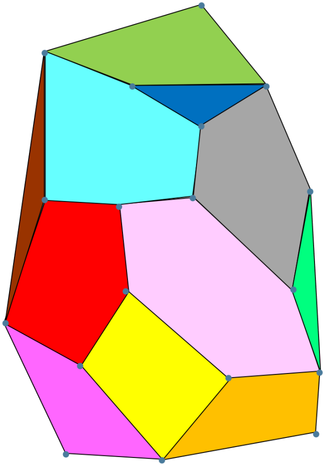


- Related to Euclidean TSP
- Two categories: minimization, maximization
- We got 2nd place
- Techniques: greedy and local search

Minimum Convex Partition:

- Input: A set of points $S \subset \mathbb{R}^2$
- Output: A simple partition of the convex hull of S into convex regions with vertex set S
- Goal: Minimize the number of regions

- We got 4th place
- Used Mixed Integer Programming



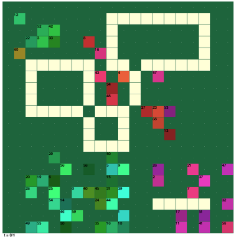
11 convex regions

Coordinated Motion Planning:

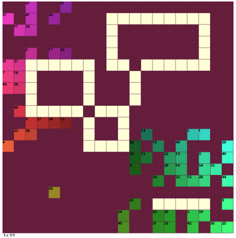
- Input: Sets $S, T \subset \mathbb{Z}^2$ of start and target locations for n robots and possibly a set of obstacles
- Output: A sequence of movements for all robots from start to target avoiding collisions
- Goal: Minimize the total time (makespan) or the total number of movements (energy)

- 1st place in makespan category,
  3rd place in energy category
- Used storage network and conflict optimizer

Start:



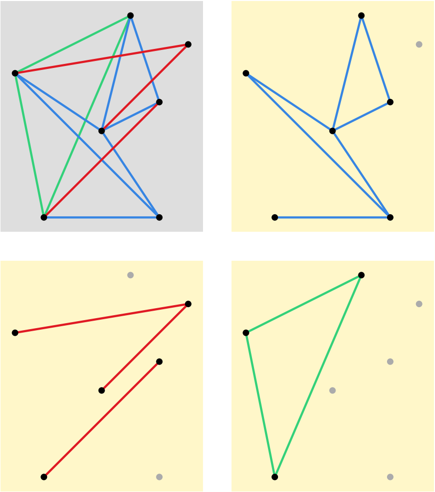
Target:



Partition Into Plane Graphs:

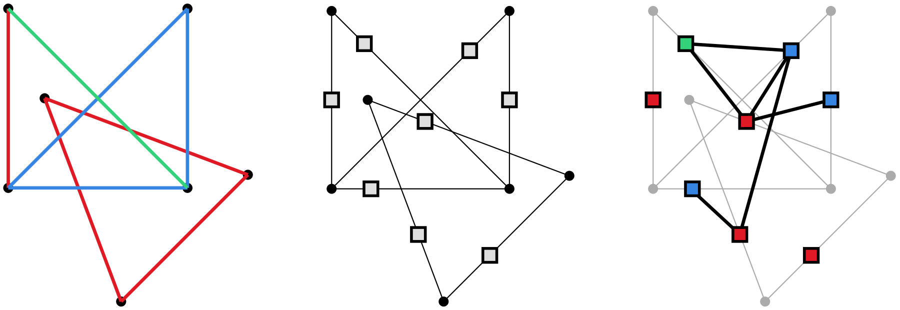
- Input: A graph G embedded in the plane with straight edges
- Output: A partition of G into plane graphs
- Goal: Minimize the number of partitions (colors)

- We won 1st place
- Best solution among all teams to every instance



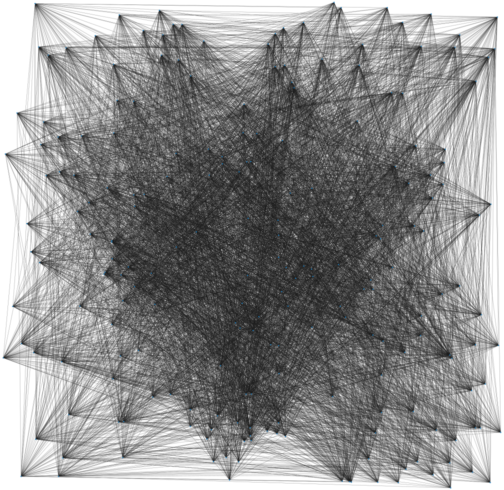
Reduction to Vertex Coloring

- Each segment becomes a vertex
- Two segments that “cross” define an edge



Instances

- 225 instances
- From 2518 to 74166 segments
- Based on random points or polygons
  - Random points: density $\sim 40\%$
  - Polygons: density $\sim 15\%$
- Number of colors from 38 to 650
- Impossible to see the colorings



Random points instance: 4641 segments

Strategy

- Find initial solutions:
  - Greedy
  - DSATUR
  - Convex hull area
  - Squeaky wheel
- Improve existing solutions
  - Conflict optimizer
    (technique from previous year)

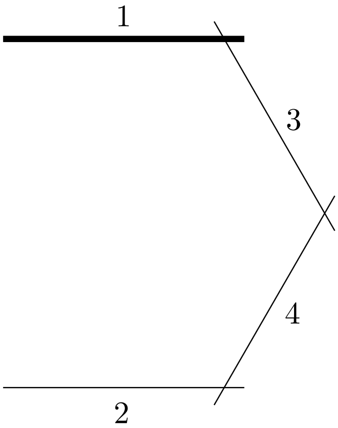
Polygon instance: 5013 segments

Greedy

Greedy coloring: [MIT76]

- For each segment s:
  - color[s] $\leftarrow$ first valid color

- Order of the n segments is very important
  Optimal order always exists!
- May not be optimal even for 2 colors!

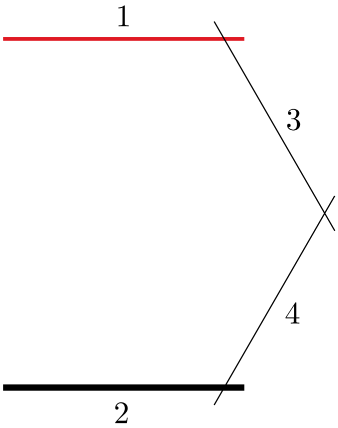


Greedy

Greedy coloring: [MIT76]

- For each segment s:
  - color[s] $\leftarrow$ first valid color

- Order of the n segments is very important
  Optimal order always exists!
- May not be optimal even for 2 colors!

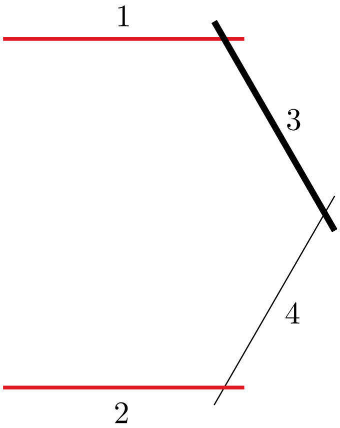


Greedy

Greedy coloring: [MIT76]

- For each segment s:
  - color[s] $\leftarrow$ first valid color

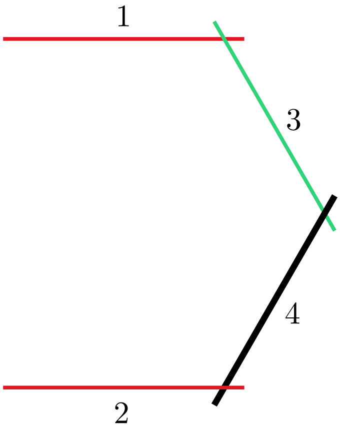
- Order of the n segments is very important
  Optimal order always exists!
- May not be optimal even for 2 colors!



Greedy coloring: [MIT76]

- For each segment s:
  - color[s] $\leftarrow$ first valid color

- Order of the n segments is very important
  Optimal order always exists!
- May not be optimal even for 2 colors!

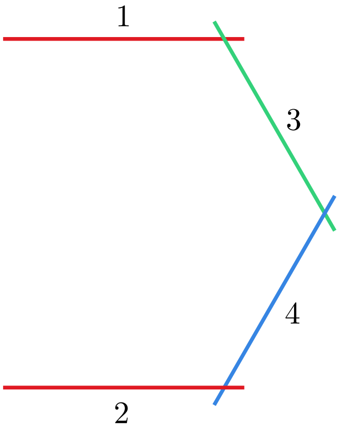


Greedy

Greedy coloring: [MIT76]

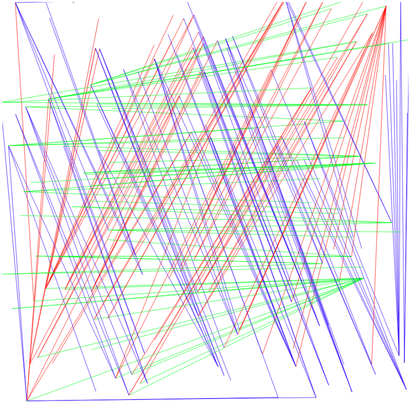
- For each segment s:
  - color[s] $\leftarrow$ first valid color

- Order of the n segments is very important
  Optimal order always exists!
- May not be optimal even for 2 colors!



Angle

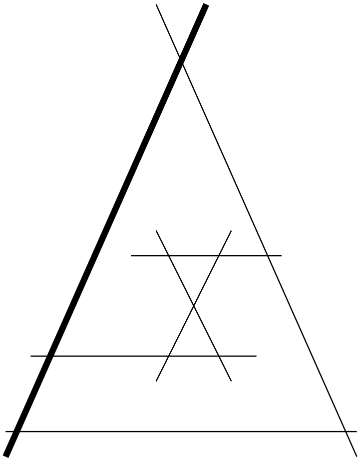
- Sorting by high to low degree is common
  Slow since all pairs of segments are tested
- Sorting by angle works well for the challenge
  Complexity still $O(n^2)$, but fast in practice
- 5.5 seconds for 74166 segments and 537 colors
- Since it is fast, we can run many times,
  for example with random starting angles
- 10 attempts take 55 seconds: 502 colors



3 colors produced by angle

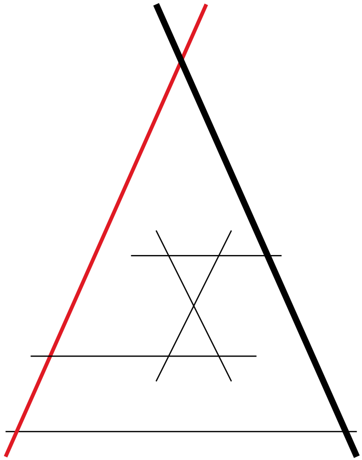
DSatur [Bré79]

- Greedy coloring with a dynamic choice of which segment to color next
- Color the segment that maximizes:
  - Number of different colors crossed
  - Break ties by number of crossings
- Optimal for bipartite, cycles, and wheels
- Complexity increases to $O(n^2k)$ for k colors
- 90 seconds for 74166 segments and also got 502 colors



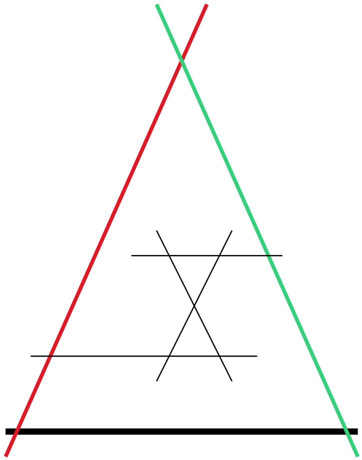
DSatur [Bré79]

- Greedy coloring with a dynamic choice of which segment to color next
- Color the segment that maximizes:
  - Number of different colors crossed
  - Break ties by number of crossings
- Optimal for bipartite, cycles, and wheels
- Complexity increases to $O(n^2k)$ for k colors
- 90 seconds for 74166 segments and also got 502 colors



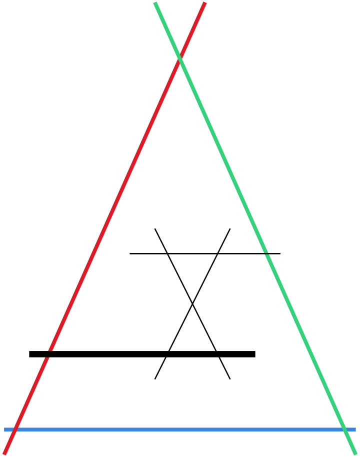
DSatur [Bré79]

- Greedy coloring with a dynamic choice of which segment to color next
- Color the segment that maximizes:
  - Number of different colors crossed
  - Break ties by number of crossings
- Optimal for bipartite, cycles, and wheels
- Complexity increases to $O(n^2k)$ for k colors
- 90 seconds for 74166 segments and also got 502 colors



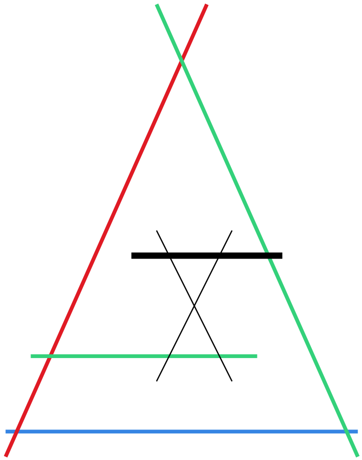
DSatur [Bré79]

- Greedy coloring with a dynamic choice of which segment to color next
- Color the segment that maximizes:
  - Number of different colors crossed
  - Break ties by number of crossings
- Optimal for bipartite, cycles, and wheels
- Complexity increases to $O(n^2k)$ for k colors
- 90 seconds for 74166 segments and also got 502 colors



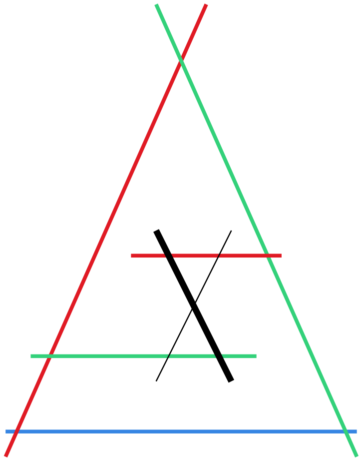
DSatur [Bré79]

- Greedy coloring with a dynamic choice of which segment to color next
- Color the segment that maximizes:
  - Number of different colors crossed
  - Break ties by number of crossings
- Optimal for bipartite, cycles, and wheels
- Complexity increases to $O(n^2k)$ for k colors
- 90 seconds for 74166 segments and also got 502 colors



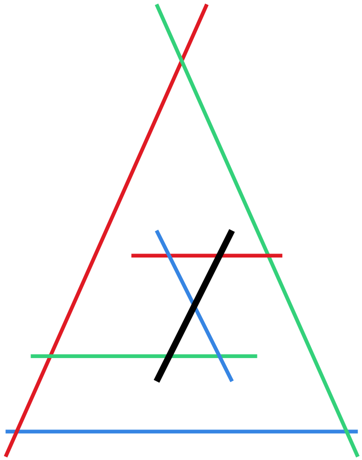
DSatur [Bré79]

- Greedy coloring with a dynamic choice of which segment to color next
- Color the segment that maximizes:
  - Number of different colors crossed
  - Break ties by number of crossings
- Optimal for bipartite, cycles, and wheels
- Complexity increases to $O(n^2k)$ for k colors
- 90 seconds for 74166 segments and also got 502 colors



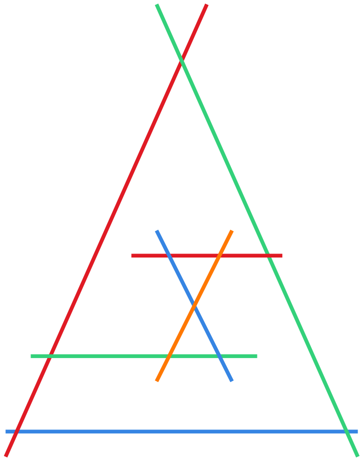
DSatur [Bré79]

- Greedy coloring with a dynamic choice of which segment to color next
- Color the segment that maximizes:
  - Number of different colors crossed
  - Break ties by number of crossings
- Optimal for bipartite, cycles, and wheels
- Complexity increases to $O(n^2k)$ for k colors
- 90 seconds for 74166 segments and also got 502 colors



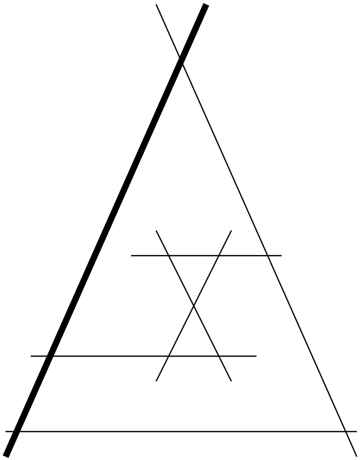
DSatur [Bré79]

- Greedy coloring with a dynamic choice of which segment to color next
- Color the segment that maximizes:
  - Number of different colors crossed
  - Break ties by number of crossings
- Optimal for bipartite, cycles, and wheels
- Complexity increases to $O(n^2k)$ for k colors
- 90 seconds for 74166 segments and also got 502 colors



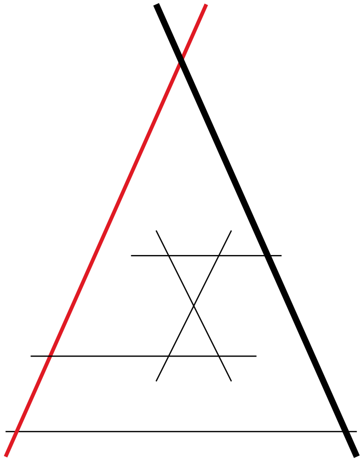
DSatHull

- Uses DSatur ordering to color segments
- Does not assign the first valid color
- Instead, colors asegment s with the valid color C that minimizes $\mathrm{area}(\mathrm{hull}(C \cup \{s\})) - \mathrm{area}(\mathrm{hull}(C))$
- Uses the geometry of the instances
- Same complexity as DSATUR and barely slower
- 97 seconds for 74166 segments and 488 colors



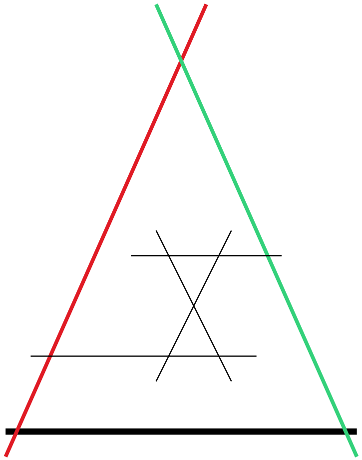
DSatHull

- Uses DSatur ordering to color segments
- Does not assign the first valid color
- Instead, colors asegment s with the valid color C that minimizes $\mathrm{area}(\mathrm{hull}(C \cup \{s\})) - \mathrm{area}(\mathrm{hull}(C))$
- Uses the geometry of the instances
- Same complexity as DSATUR and barely slower
- 97 seconds for 74166 segments and 488 colors



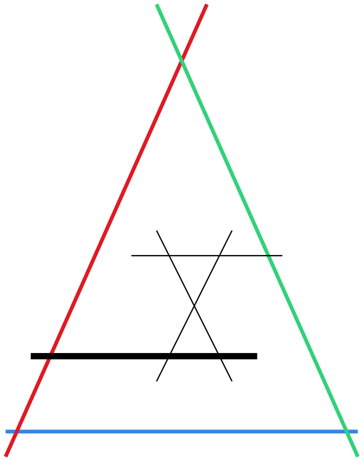
DSatHull

- Uses DSatur ordering to color segments
- Does not assign the first valid color
- Instead, colors asegment s with the valid color C that minimizes $\mathrm{area}(\mathrm{hull}(C \cup \{s\})) - \mathrm{area}(\mathrm{hull}(C))$
- Uses the geometry of the instances
- Same complexity as DSATUR and barely slower
- 97 seconds for 74166 segments and 488 colors



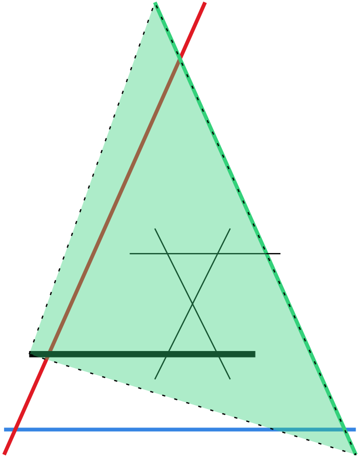
DSatHull

- Uses DSatur ordering to color segments
- Does not assign the first valid color
- Instead, colors asegment s with the valid color C that minimizes $\text{area}(\text{hull}(C \cup \{s\})) - \text{area}(\text{hull}(C))$
- Uses the geometry of the instances
- Same complexity as DSATUR and barely slower
- 97 seconds for 74166 segments and 488 colors



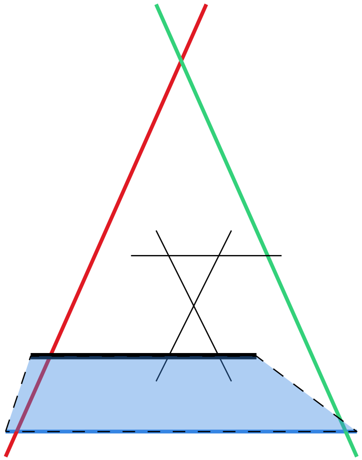
DSatHull

- Uses DSatur ordering to color segments
- Does not assign the first valid color
- Instead, colors asegment s with the valid color C that minimizes $\text{area}(\text{hull}(C \cup \{s\})) - \text{area}(\text{hull}(C))$
- Uses the geometry of the instances
- Same complexity as DSATUR and barely slower
- 97 seconds for 74166 segments and 488 colors



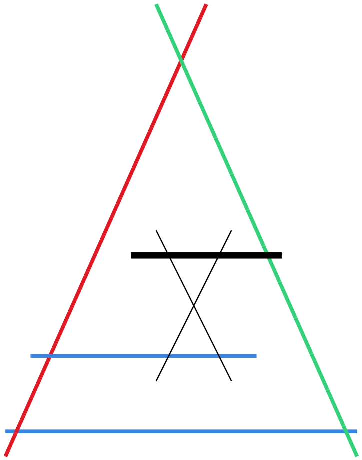
DSatHull

- Uses DSatur ordering to color segments
- Does not assign the first valid color
- Instead, colors asegment s with the valid color C that minimizes $\mathrm{area}(\mathrm{hull}(C \cup \{s\})) - \mathrm{area}(\mathrm{hull}(C))$
- Uses the geometry of the instances
- Same complexity as DSATUR and barely slower
- 97 seconds for 74166 segments and 488 colors



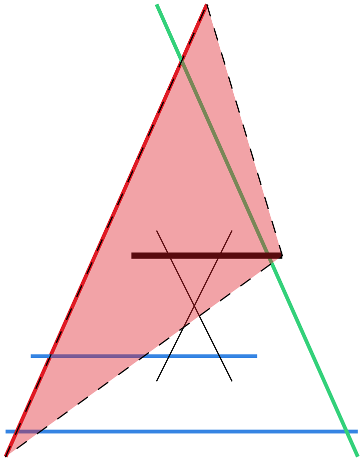
DSatHull

- Uses DSatur ordering to color segments
- Does not assign the first valid color
- Instead, colors asegment s with the valid color C that minimizes $\text{area}(\text{hull}(C \cup \{s\})) - \text{area}(\text{hull}(C))$
- Uses the geometry of the instances
- Same complexity as DSATUR and barely slower
- 97 seconds for 74166 segments and 488 colors



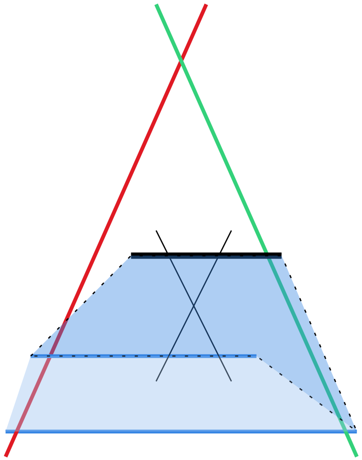
DSatHull

- Uses DSatur ordering to color segments
- Does not assign the first valid color
- Instead, colors asegment s with the valid color C that minimizes $\mathrm{area}(\mathrm{hull}(C \cup \{s\})) - \mathrm{area}(\mathrm{hull}(C))$
- Uses the geometry of the instances
- Same complexity as DSATUR and barely slower
- 97 seconds for 74166 segments and 488 colors



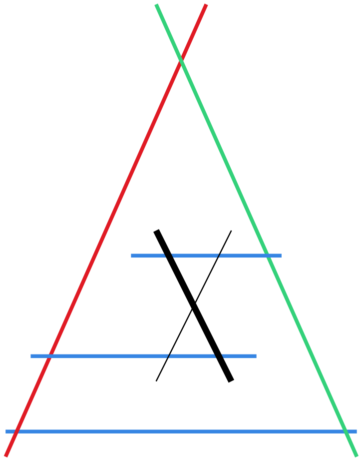
DSatHull

- Uses DSatur ordering to color segments
- Does not assign the first valid color
- Instead, colors asegment s with the valid color C that minimizes $\text{area}(\text{hull}(C \cup \{s\})) - \text{area}(\text{hull}(C))$
- Uses the geometry of the instances
- Same complexity as DSATUR and barely slower
- 97 seconds for 74166 segments and 488 colors



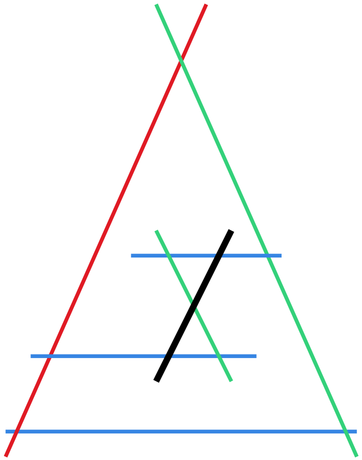
DSatHull

- Uses DSatur ordering to color segments
- Does not assign the first valid color
- Instead, colors asegment s with the valid color C that minimizes $\mathrm{area}(\mathrm{hull}(C \cup \{s\})) - \mathrm{area}(\mathrm{hull}(C))$
- Uses the geometry of the instances
- Same complexity as DSATUR and barely slower
- 97 seconds for 74166 segments and 488 colors



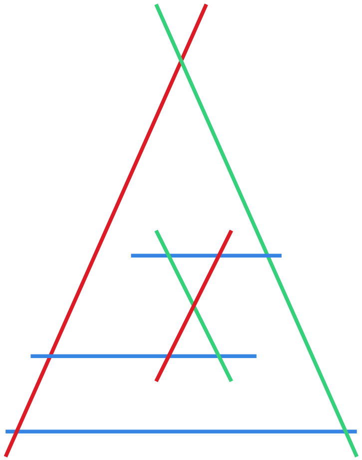
DSatHull

- Uses DSatur ordering to color segments
- Does not assign the first valid color
- Instead, colors asegment s with the valid color C that minimizes $\mathrm{area}(\mathrm{hull}(C \cup \{s\})) - \mathrm{area}(\mathrm{hull}(C))$
- Uses the geometry of the instances
- Same complexity as DSATUR and barely slower
- 97 seconds for 74166 segments and 488 colors



DSatHull

- Uses DSatur ordering to color segments
- Does not assign the first valid color
- Instead, colors asegment s with the valid color C that minimizes $\mathrm{area}(\mathrm{hull}(C \cup \{s\})) - \mathrm{area}(\mathrm{hull}(C))$
- Uses the geometry of the instances
- Same complexity as DSATUR and barely slower
- 97 seconds for 74166 segments and 488 colors



Squeaky Wheel Paradigm

Squeaky Wheel: [JoDa98]

- Solve the problem using a certain order
- Find elements that were not solved well
- Move these elements earlier in the ordering
- In the end, return the best solution found (not the last)

- One way to do apply it to coloring:
  Move all elements with last color to the beginning of the list and repeat:
  Never increases the number of colors used
- We did something different though

“The squeaky wheel gets the grease.”

Bad (the name of the heuristic)

Bad:

- $Good$ and Bad are two sets of segments always ordered by angle
- Initially, all segments are $Good$
- Greedy color Bad and then $Good$
- Move segments with last color to Bad and repeat

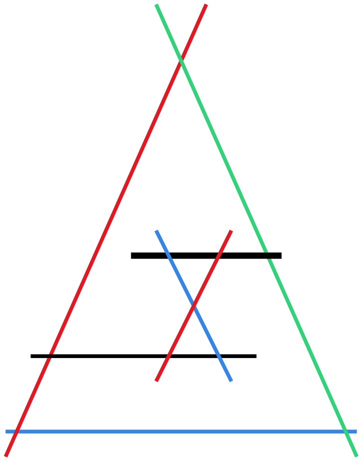
- Better than different starting random angles
- Needs several (~ 50) repetitions
- Number of colors may increase because Bad is sorted

Conflict Optimizer [CFGGLL]

- Goal: modify a given coloring to reduce the number of colors from k to $k-1$
- Partial coloring: a valid coloring of a subset of the segments

Algorithm:

- Uncolor all segments of color c
- While there is an uncolored segment s
- Color s minimizing the "number" of conflicts
- Uncolor the conflicting segments

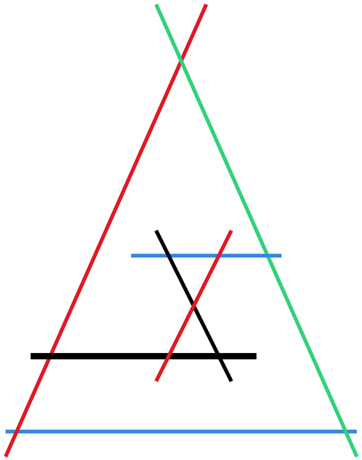


Conflict Optimizer [CFGGLL]

- Goal: modify a given coloring to reduce the number of colors from k to $k-1$
- Partial coloring: a valid coloring of a subset of the segments

Algorithm:

- Uncolor all segments of color c
- While there is an uncolored segment s
- Color s minimizing the "number" of conflicts
- Uncolor the conflicting segments

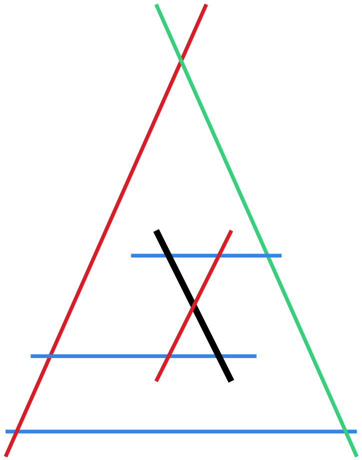


Conflict Optimizer [CFGGLL]

- Goal: modify a given coloring to reduce the number of colors from k to $k-1$
- Partial coloring: a valid coloring of a subset of the segments

Algorithm:

- Uncolor all segments of color c
- While there is an uncolored segment s
- Color s minimizing the "number" of conflicts
- Uncolor the conflicting segments

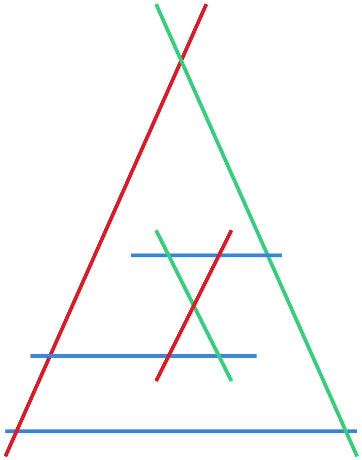


Conflict Optimizer [CFGGLL]

- Goal: modify a given coloring to reduce the number of colors from k to $k-1$
- Partial coloring: a valid coloring of a subset of the segments

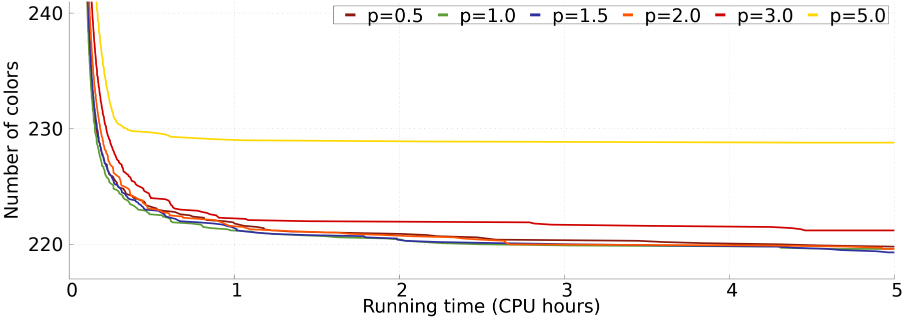
Algorithm:

- Uncolor all segments of color c
- While there is an uncolored segment s
- Color s minimizing the "number" of conflicts
- Uncolor the conflicting segments



Conflict Optimizer Details

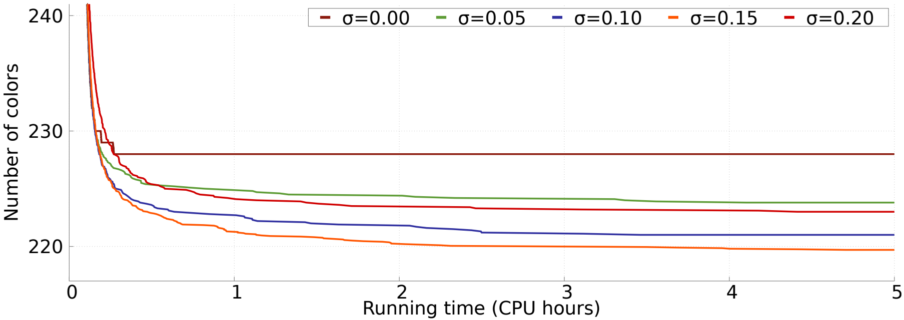
- Uncolored segments are in a queue
- Let $q(s)$ be the number of times a segment s is uncolored
- $weight(s) = 1 + q(s)^p$, where $p = 1.2$ is the default
- Minimize sum of weights of conflicting segments



Parameter analysis for p on an instance with 13806 segments

Conflict Optimizer Improvements

1. Apply Gaussian noise of variance $\sigma = .15$ to $weight(s)$
2. Find a large clique and set the weight of its segments to ∞
3. Iteratively remove segments of degree at most $k - 2$ and color them later
4. Perform a bounded depth first search when coloring



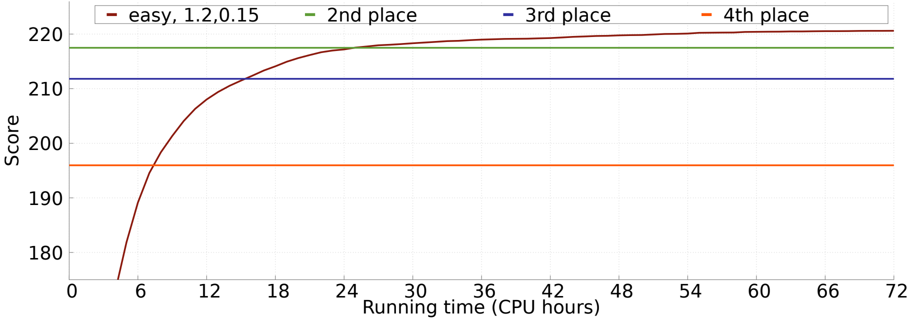
Parameter analysis for σ on an instance with 13806 segments

Some Numbers of Colors

instance	density	Greedy	Angle	Bad	DSatur	DSHull	Best	Clique
rsqrpecn8051	41%	342	205	203	213	201	**175**	173
vispecn13806	19%	427	308	300	289	283	**218**	177
rsqrp14364	50%	294	139	139	165	157	**136**	134
vispecn19370	13%	370	285	278	265	248	**192**	169
visp26405	7%	154	101	97	94	92	**81**	78
visp31334	5%	152	90	88	99	98	**81**	77
visp38574	14%	287	148	146	168	168	**133**	118
sqrpecn45700	47%	952	504	500	562	522	**462**	460
reecn51526	24%	642	361	359	388	360	**310**	308
vispecn58391	12%	789	607	594	499	494	**367**	305
vispecn65831	12%	916	647	637	578	564	**439**	357
sqrp72075	47%	609	280	280	363	337	**269**	264

Scores

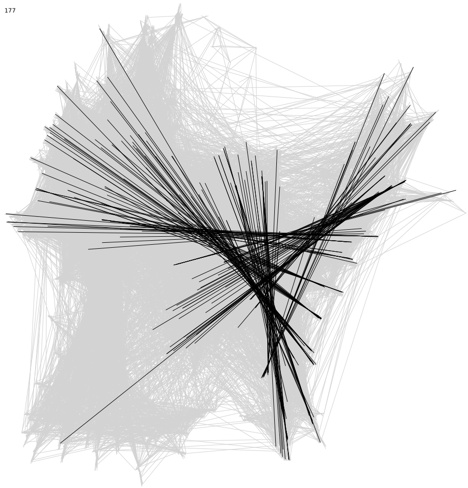
- 225 instances
- Each instance gets a score between 0 and 1, total score is the sum
- Starting from a score of 1, we lose 5% of the score for each 1% more colors compared to the best submitted solution
- We achieved a perfect 225 score



How much CPU core time (per instance) we need to win?

Cliques

- We also worked on finding large cliques
- Useful as lower bounds and to improve algorithms by fixing the colors of the clique segments
- Used mixed integer programming, simulated annealing, branch and bound...



Clique with 177 segments out of 13806

Bibliography

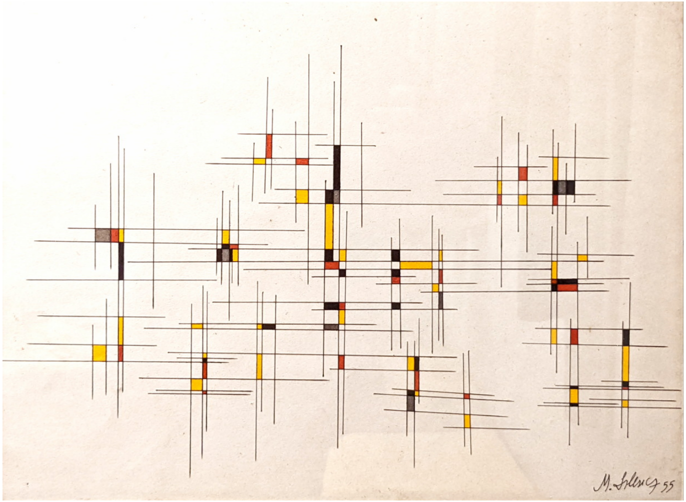
[Mit76] Mitchem, John. On various algorithms for estimating the chromatic number of a graph. *The Computer Journal*, 19.2, 182–183, 1976.

[Bré79] Brélaz, Daniel. New methods to color the vertices of a graph. *Communications of the ACM*, 22.4, 251–256, 1979.

[JoDa98] Joslin, David E., and David P. Clements. Squeaky Wheel Optimization. *AAAI/IAAI*, 1998.

[CFGGLL] Shadoks Approach to Low-Makespan Coordinated Motion Planning Loïc Crombez, Guilherme D. da Fonseca, Yan Gerard, Aldo Gonzalez-Lorenzo, Pascal Lafourcade, and Luc Libralesso; CG:SHOP 2021 special issue of ACM Journal of Experimental Algorithmics, to appear.

Thank You!



Art by Mário Silésio