Monte Carlo Graph Coloring

Tristan Cazenave, Benjamin Negrevergne, Florian Sikora

LAMSADE, Université Paris Dauphine, CNRS – France

Outline

Graph Coloring...

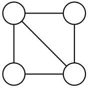
...meets Monte Carlo

Competitors

Experimental results

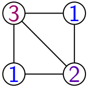
Graph Coloring

- Given a graph



Graph Coloring

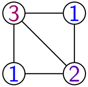
- Given a graph



- give a **color** to each vertex
- such that **adjacent** vertices receive **different colors**
- **proper** coloration

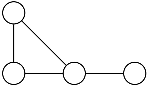
Graph Coloring

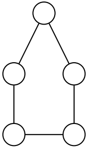
- Given a graph



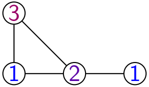
- give a **color** to each vertex
- such that **adjacent** vertices receive **different colors**
- **proper** coloration
- **Chromatic number** : smallest number of colors needed to have a proper coloration

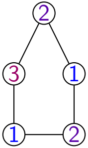
Examples



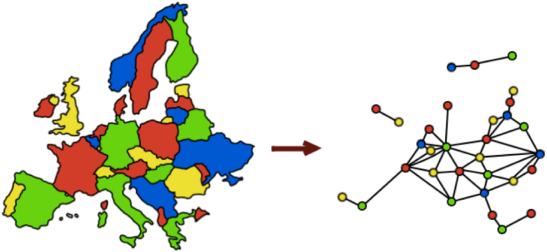


Examples

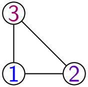




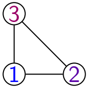
4-color Theorem

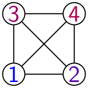


Cliques are lower bounds...

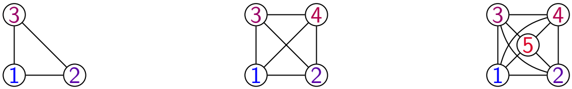


Cliques are lower bounds...



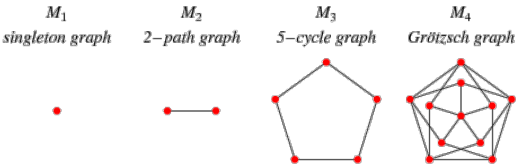


Cliques are lower bounds...



... but not necessary

- There are graphs (e.g. Mycielski graph) **without triangles** but with arbitrarily large chromatic number

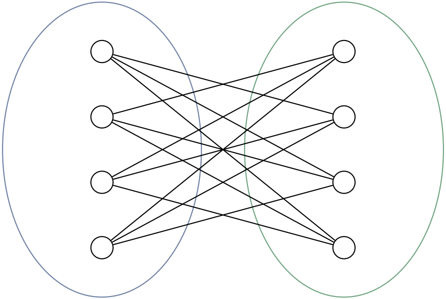


...

(chromatic number of 1, 2, 3, 4...)

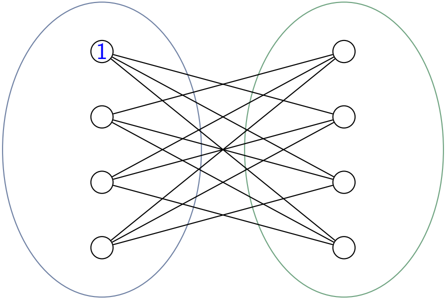
Order matters

- Suppose you give always the lowest possible color



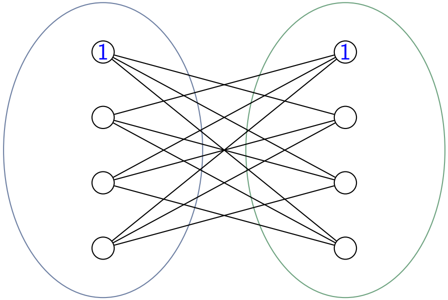
Order matters

- Suppose you give always the lowest possible color



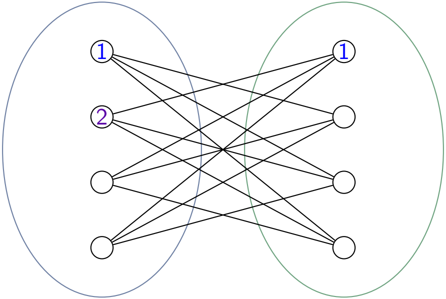
Order matters

- Suppose you give always the lowest possible color



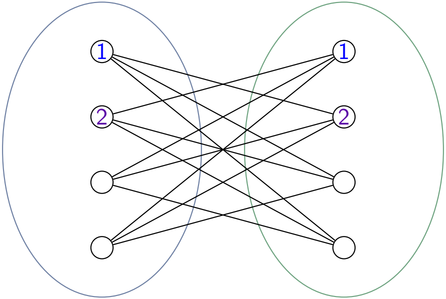
Order matters

- Suppose you give always the lowest possible color



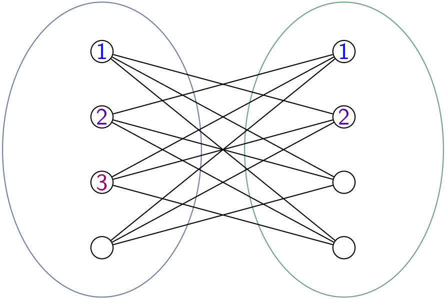
Order matters

- Suppose you give always the lowest possible color



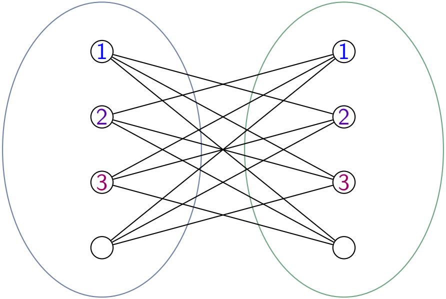
Order matters

- Suppose you give always the lowest possible color



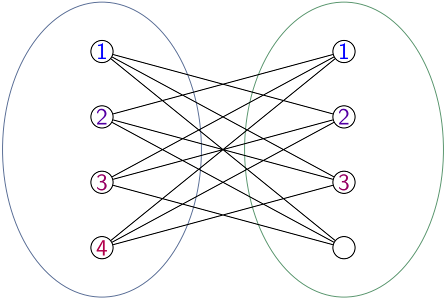
Order matters

- Suppose you give always the lowest possible color



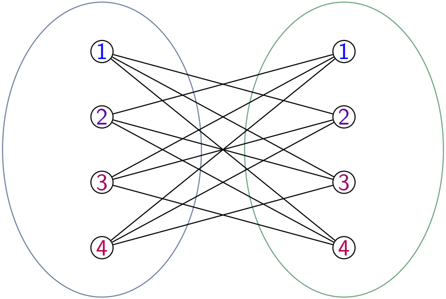
Order matters

- Suppose you give always the lowest possible color



Order matters

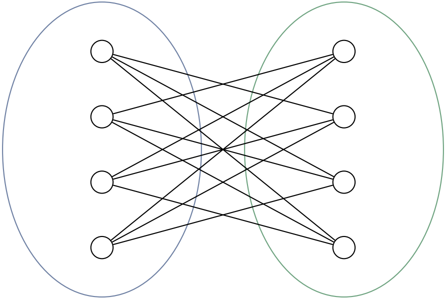
- Suppose you give always the lowest possible color



- Use 4 (even $n/2$) colors

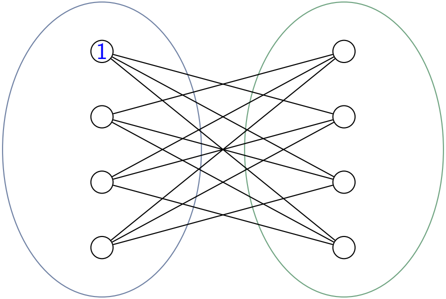
Order matters

- Suppose you give always the lowest possible color



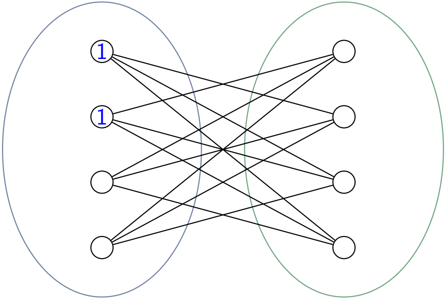
Order matters

- Suppose you give always the lowest possible color



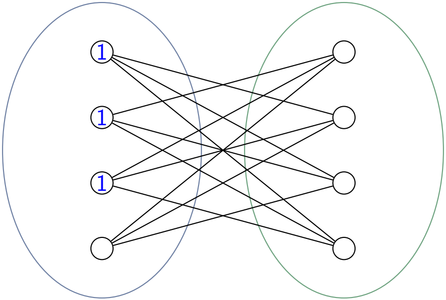
Order matters

- Suppose you give always the lowest possible color



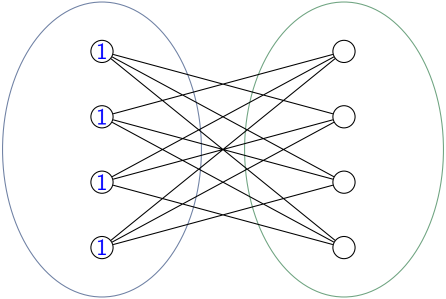
Order matters

- Suppose you give always the lowest possible color



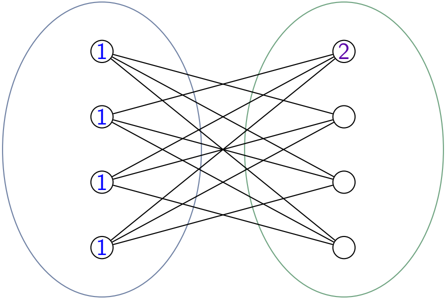
Order matters

- Suppose you give always the lowest possible color



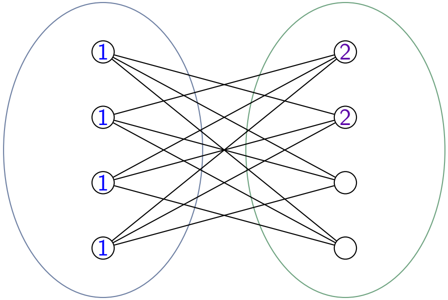
Order matters

- Suppose you give always the lowest possible color



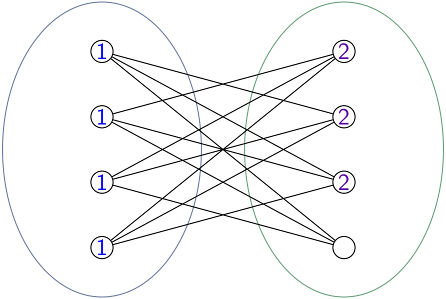
Order matters

- Suppose you give always the lowest possible color



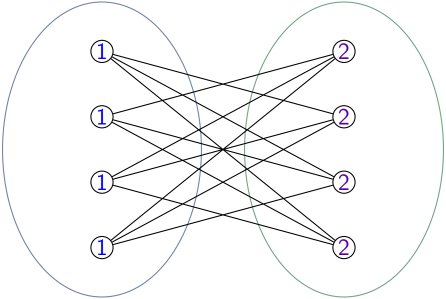
Order matters

- Suppose you give always the lowest possible color



Order matters

- Suppose you give always the lowest possible color

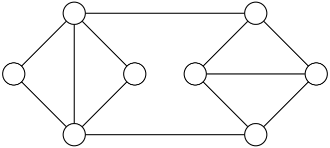


- Use 2 colors !

Order matters

- DSatur [Brélaz 79] greedily colors the graph with the smallest possible color
- The chosen vertex is the "most constrained" (max degree to break ties)

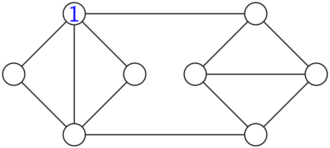
Often good...



Order matters

- DSatur [Brélaz 79] greedily colors the graph with the smallest possible color
- The chosen vertex is the "most constrained" (max degree to break ties)

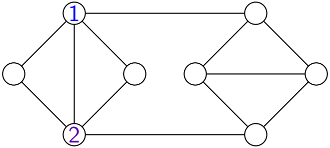
Often good...



Order matters

- DSatur [Brélaz 79] greedily colors the graph with the smallest possible color
- The chosen vertex is the "most constrained" (max degree to break ties)

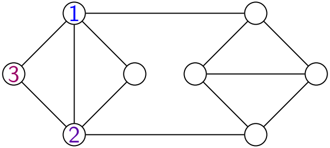
Often good...



Order matters

- DSatur [Brélaz 79] greedily colors the graph with the smallest possible color
- The chosen vertex is the "most constrained" (max degree to break ties)

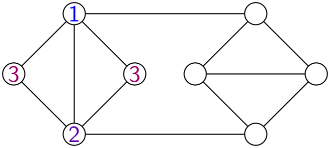
Often good...



Order matters

- DSatur [Brélaz 79] greedily colors the graph with the smallest possible color
- The chosen vertex is the “most constrained” (max degree to break ties)

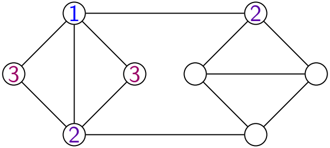
Often good...



Order matters

- DSatur [Brélaz 79] greedily colors the graph with the smallest possible color
- The chosen vertex is the "most constrained" (max degree to break ties)

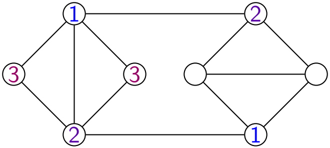
Often good...



Order matters

- DSatur [Brélaz 79] greedily colors the graph with the smallest possible color
- The chosen vertex is the "most constrained" (max degree to break ties)

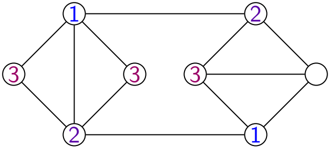
Often good...



Order matters

- DSatur [Brélaz 79] greedily colors the graph with the smallest possible color
- The chosen vertex is the "most constrained" (max degree to break ties)

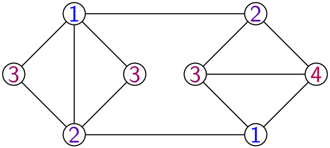
Often good...



Order matters

- DSatur [Brélaz 79] greedily colors the graph with the smallest possible color
- The chosen vertex is the "most constrained" (max degree to break ties)

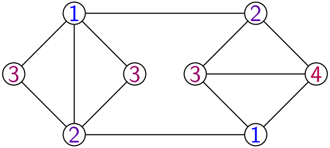
Often good...



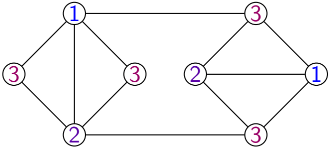
Order matters

- DSatur [Brélaz 79] greedily colors the graph with the smallest possible color
- The chosen vertex is the "most constrained" (max degree to break ties)

Often good...



...but not optimal!



Outline

Graph Coloring...

...meets Monte Carlo

Competitors

Experimental results

Problem to solve

- **Optimisation**: finding the chromatic number of the graph
- **Decision**: given a graph and an integer k, can the graph be colored with k colors
- We focus on the latter (we can simulate the former with decreasing values of k)

Coloring as a game

Settings:

- The number of allowed colors k is **fixed**
- **Move**: put color c to a vertex v
- **Playout**: given a vertex ordering, color each node of the graph
- **Score**: number of edges minus number of improperly colored edges

(Legal) Moves

2 possibilities:

1. All $|V| * k$ possible moves
2. **Fix an ordering** and vertex after vertex, set its legal colors (colors not in the neighborhood) as **legal moves**
   - If no available colors, set all k coloration as legal moves (it will decreases the score)

(Legal) Moves

2 possibilities:

1. All $|V| * k$ possible moves
2. **Fix an ordering** and vertex after vertex, set its legal colors (colors not in the neighborhood) as **legal moves**
   - If no available colors, set all k coloration as legal moves (it will decreases the score)

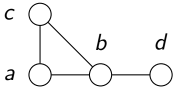
- Option 2. is more efficient (less moves) but relies on the ordering (we use DSatur ordering)

(Legal) Moves

2 possibilities:

1. All $|V| * k$ possible moves
2. **Fix an ordering** and vertex after vertex, set its legal colors (colors not in the neighborhood) as **legal moves**
   - If no available colors, set all k coloration as legal moves (it will decreases the score)

- Option 2. is more efficient (less moves) but relies on the ordering (we use DSatur ordering)



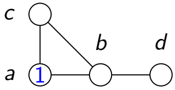
$k = 3$, legal moves for a : 1,2,3

(Legal) Moves

2 possibilities:

1. All $|V| * k$ possible moves
2. **Fix an ordering** and vertex after vertex, set its legal colors (colors not in the neighborhood) as **legal moves**
   - If no available colors, set all k coloration as legal moves (it will decreases the score)

- Option 2. is more efficient (less moves) but relies on the ordering (we use DSatur ordering)



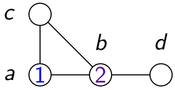
$k = 3$, legal moves for b : 2, 3

(Legal) Moves

2 possibilities:

1. All $|V| * k$ possible moves
2. **Fix an ordering** and vertex after vertex, set its legal colors (colors not in the neighborhood) as **legal moves**
   - If no available colors, set all k coloration as legal moves (it will decreases the score)

- Option 2. is more efficient (less moves) but relies on the ordering (we use DSatur ordering)



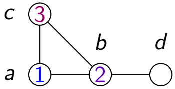
$k = 3$, legal move for c : 3

(Legal) Moves

2 possibilities:

1. All $|V| * k$ possible moves
2. **Fix an ordering** and vertex after vertex, set its legal colors (colors not in the neighborhood) as **legal moves**
   - If no available colors, set all k coloration as legal moves (it will decreases the score)

- Option 2. is more efficient (less moves) but relies on the ordering (we use DSatur ordering)



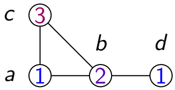
$k = 3$, legal moves for d : 2, 3

(Legal) Moves

2 possibilities:

1. All $|V| * k$ possible moves
2. **Fix an ordering** and vertex after vertex, set its legal colors (colors not in the neighborhood) as **legal moves**
   - If no available colors, set all k coloration as legal moves (it will decreases the score)

- Option 2. is more efficient (less moves) but relies on the ordering (we use DSatur ordering)



Outline

Graph Coloring...

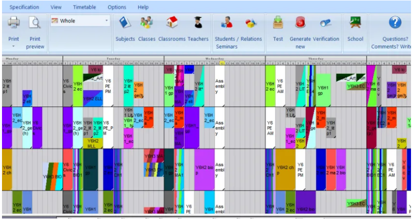
...meets Monte Carlo

Competitors

Experimental results

Complexity

- Very **hard**, even for 3 colors
- Not approximable at all
- However, largely used in applications, important to solve (scheduling, timetabling...)



In practice

- Exact **moderately exponential** algorithms
- Mathematical programming
- **Approximation** on special graph classes
- Parameterized complexity and data reduction
- **Heuristics**, meta-heuristics...

- Exact methods are limited after few hundred of vertices

Competitors

1. **Greedy** Coloring according to the DSatur ordering
2. Naïve **SAT** encoding solved with MiniSAT
3. **HEAD** [Moalic and Gondran 18]
   - Local search (Tabu-Search) mixed with an evolutionary algorithm and **specialized graph-coloring operators**

Competitors

1. **Greedy** Coloring according to the DSatur ordering
2. Naïve **SAT** encoding solved with MiniSAT
3. **HEAD** [Moalic and Gondran 18]
   - Local search (Tabu-Search) mixed with an evolutionary algorithm and **specialized graph-coloring operators**

To be compared with:

1. Nested Monte Carlo Search (**NMCS**) [Cazenave 09]
2. Nested Rollout Policy Adaptation (**NRPA**) [Rosin 11]

Outline

Graph Coloring...

...meets Monte Carlo

Competitors

Experimental results

Settings

- Benchmark of graphs by [GUALANDI AND CHIARANDINI] (from DIMACS) from different sources (road network, random, latin square...)
- Optimum is known for most of them

Settings

- Benchmark of graphs by [Gualandi and Chiarandini] (from DIMACS) from different sources (road network, random, latin square...)
- Optimum is known for most of them
- We **start from** k given by the **greedy** coloring
- We use the **decision algorithm** for each lower k as long as it returns YES (30min TO)
- "cheat" : we stop when we reach the known optimum

Results

- **Monte Carlo solve** all instances marked “easy” and “medium” in the benchmark
- Contrary to experiments of [Edelkamp et al. 17], **NMCS showed less good** results than NRPA , especially on hard instances
- **NRPA often better than the SAT** approach
- **HEAD is better** in general but NRPA is sometimes better

Some results

					NMCS		NRPA		SAT		HEAD	
Instance	$\lvert V\rvert$	$\lvert E\rvert$	χ	**UBI**	**UB**	**Reached**	**UB**	**Reached**	**UB**	**Reached**	**UB**	**Reached**
flat300_28_0	300	21695	28	41	38	20%	35	20%	39	100%	**31**	100%
r1000.5	1000	238267	234	248	243	20%	**240**	40%	247	100%	248	–
r250.5	250	14849	65	67	**65**	100%	**65**	100%	**65**	100%	66	40%
DSJR500.5	500	58862	122	132	125	60%	**122**	40%	126	100%	124	60%
DSJR500.1c	500	121275	85	88	88	–	87	60%	**86**	100%	**86**	80%
DSJC125.5	125	3891	17	23	19	100%	18	100%	19	100%	**17**	100%
DSJC125.9	125	6961	44	50	45	40%	**44**	100%	46	100%	**44**	100%
DSJC250.9	250	27897	72	90	84	20%	76	20%	86	100%	**72**	100%
queen10_10	100	2940	11	14	**11**	60%	**11**	40%	12	100%	**11**	100%
queen11_11	121	3960	11	14	13	100%	13	100%	13	100%	**12**	100%

NRPA sometimes the best
HEAD often the best

Some results

Instance	$\|V\|$	$\|E\|$	χ	UBI	NMCS UB	NMCS Reached	NRPA UB	NRPA Reached	SAT UB	SAT Reached	HEAD UB	HEAD Reached
le450_5a	450	5714	5	10	6	20%	**5**	100%	**5**	100%	**5**	100%
le450_5b	450	5734	5	7	6	40%	**5**	20%	**5**	100%	**5**	100%
le450_15b	450	8169	15	17	**15**	100%	**15**	100%	**15**	100%	**15**	100%
le450_15c	450	16680	15	24	22	100%	21	100%	22	100%	**15**	100%
le450_15d	450	16750	15	24	22	100%	20	20%	22	100%	**15**	100%
le450_25c	450	17343	25	28	27	100%	**26**	100%	27	100%	**26**	100%
le450_25d	450	17425	25	29	27	100%	**26**	100%	27	100%	**26**	100%
qg.order60	3600	212400	60	63	**60**	40%	62	100%	61	100%	**60**	100%
qg.order100	10000	990000	100	106	–	20%	102	20%	–	20%	**100**	100%
Avg. ratio to χ					1.7626		1.0963		1.1300		1.0089	

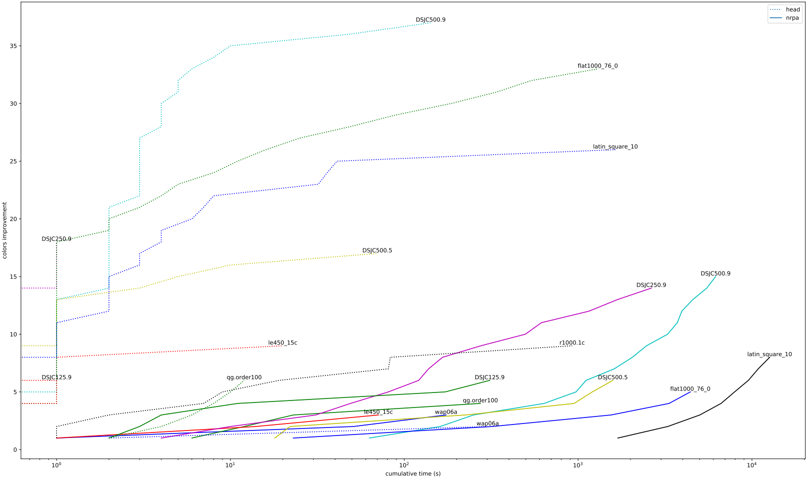
NRPA better than NMCS and SAT, but HEAD dominates

Some results

instance	$\|V\|$	$\|E\|$	χ_{LB}	UBI	NMCS UB	NMCS Reached	NRPA UB	NRPA Reached	SAT UB	SAT Reached	HEAD UB	HEAD Reached
DSJC250.1	250	3218	4	10	9	100%	**8**	40%	9	100%	**8**	100%
DSJC250.5	250	15668	26	37	34	100%	32	100%	35	100%	**28**	100%
DSJC500.1	500	12458	9	16	14	40%	14	100%	15	100%	**12**	100%
DSJC500.5	500	62624	43	65	62	20%	59	80%	63	100%	**48**	100%
DSJC500.9	500	112437	123	163	161	20%	148	20%	163	–	**126**	100%
DSJC1000.1	1000	49629	10	25	25	–	24	100%	25	–	**21**	100%
DSJC1000.5	1000	249826	73	114	114	–	112	40%	114	–	**83**	60%
DSJC1000.9	1000	449449	216	301	301	–	299	40%	301	–	**223**	20%
flat1000_50_0	1000	245000	15	113	113	–	111	40%	113	–	**50**	100%

NMCS struggle with large graphs

Hope



On some instances, NRPA continues to “learn” over time

Conclusion

- **Monte Carlo competes** without any specialized rules

Conclusion

- **Monte Carlo competes** without any specialized rules
- Mix with heuristic like HEAD?
- Using Neural Network to learn the order of coloring?
- Other graph problems?

Thanks!