A Survey of Various High-Performance Graph Colouring Algorithms and Related Timetabling Issues

Rhyd Lewis

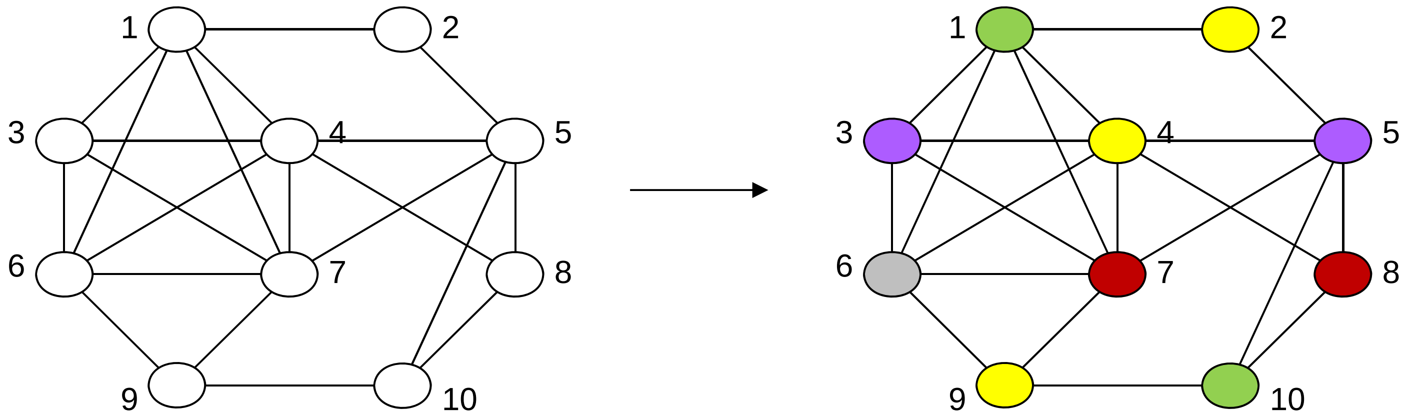
Operational Research Group,
Cardiff School of Mathematics,
Cardiff University

Talk Outline

- The graph colouring problem
- Different types of graph and connections to practical problems
- Issues in the graph colouring literature
- Algorithmic comparison
  - R. Lewis, J. Thompson, C. Mumford, and J. Gillard, “A Wide-Ranging Computational Comparison of High-Performance Graph Colouring Algorithms”, *Computers & Operations Research,* DOI: 10.1016/j.cor.2011.08.010.
- Practical issues related to search-space connectivity: a timetabling example

The graph colouring problem

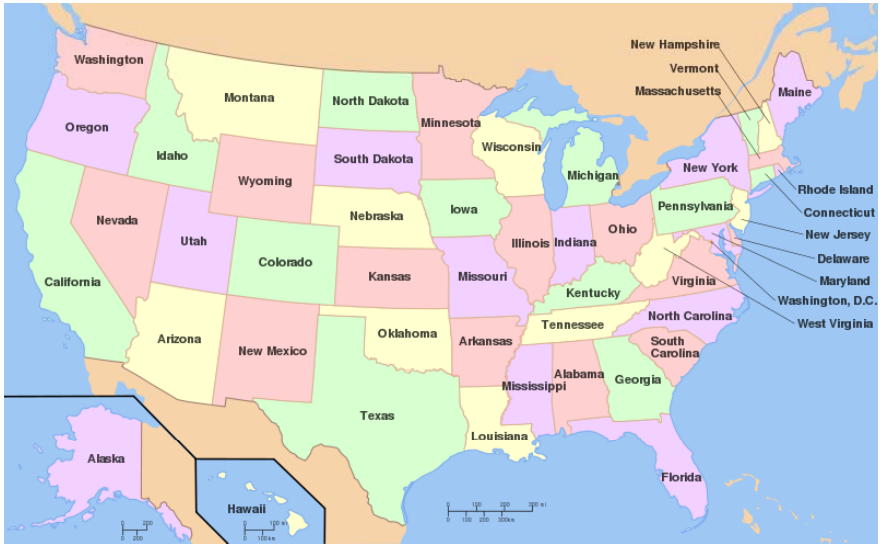
- Given a simple graph G = (V,E), paint each vertex with a colour such that
  - No adjacent vertex has the same colour
  - The number of different colours used is minimal
- This problem (and derivations of it) are NP-hard/NP-complete



- In the above case, the colouring on the right (using 5 colours) is "optimal": no solution with fewer colours exists.

Why “Colouring”?

- Graph colouring originates from the colouring of maps of countries and counties, which can be represented as “planar graphs”
- All planar graphs (and maps) can be coloured using just four colours
- However, other types of graphs require different numbers of colours

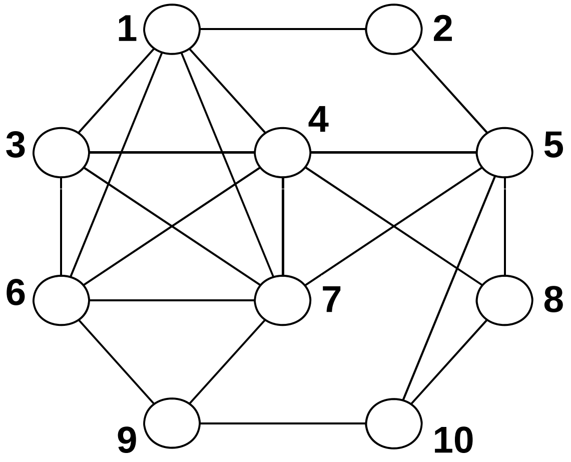


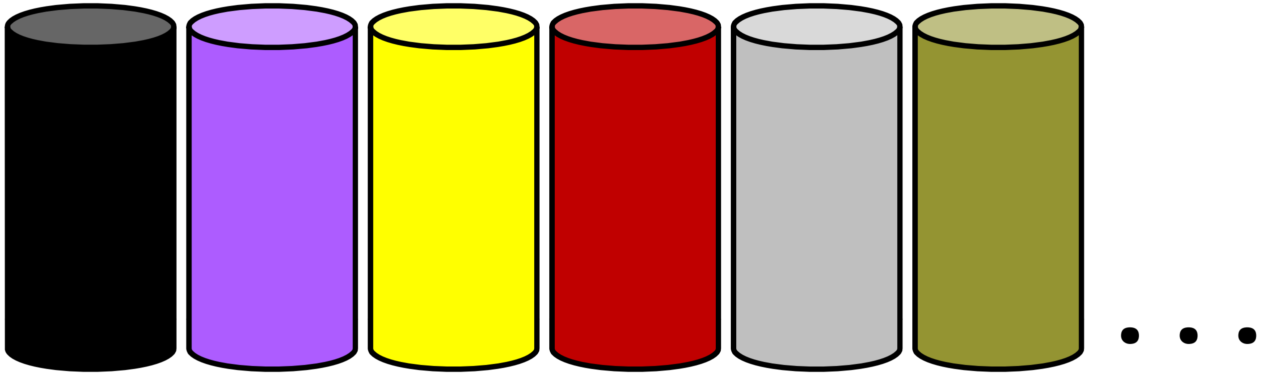
The 50 US States coloured using four colours (*source wikipedia*)

Example: The Greedy Algorithm

1) Take some permutation of the vertices
2) For each vertex v, assign v it to the first colour seen to be feasible (i.e. no clash is induced), opening new colours when necessary

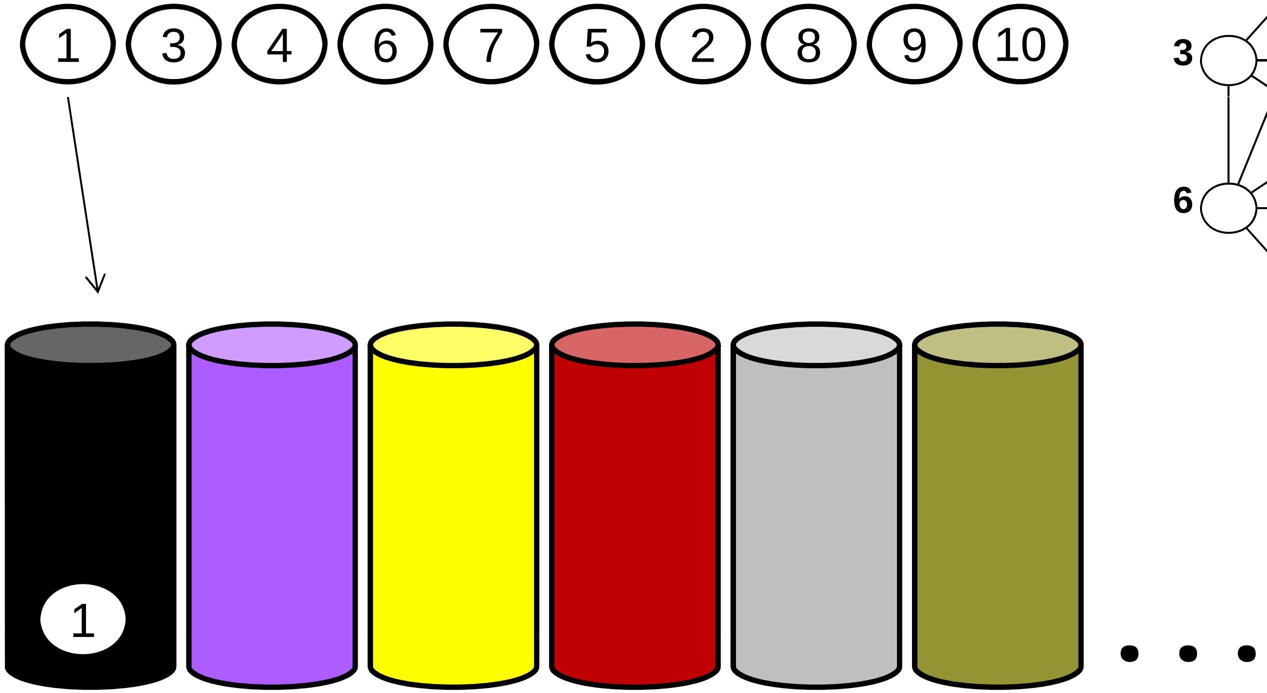
1 3 4 6 7 5 2 8 9 10

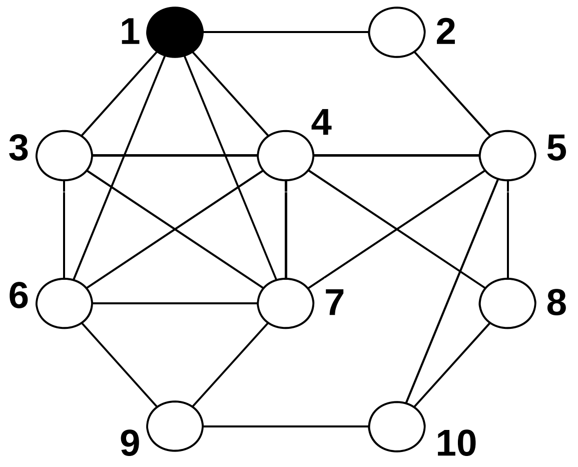




Example: The Greedy Algorithm

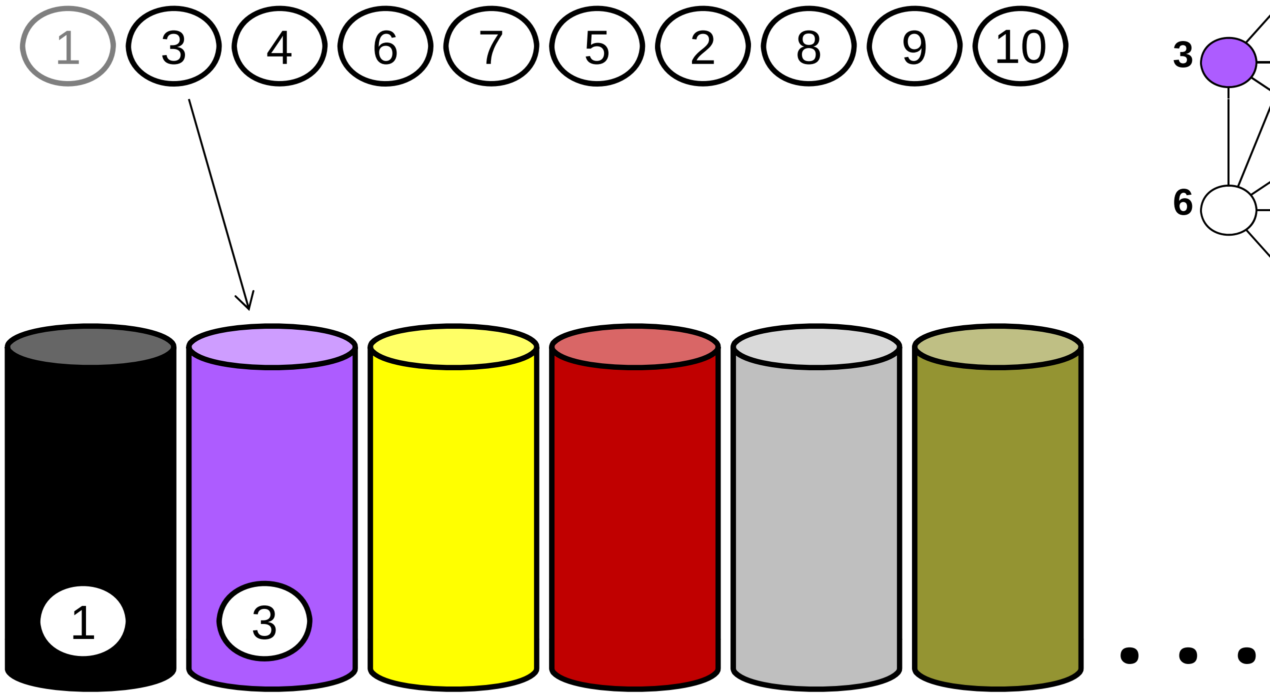
1) Take some permutation of the vertices
2) For each vertex v, assign v it to the first colour seen to be feasible (i.e. no clash is induced), opening new colours when necessary

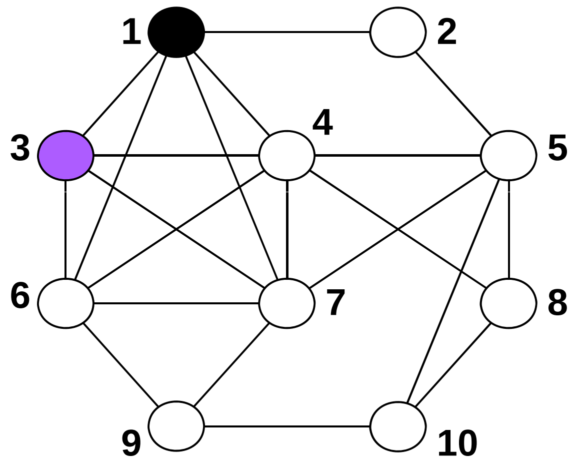




Example: The Greedy Algorithm

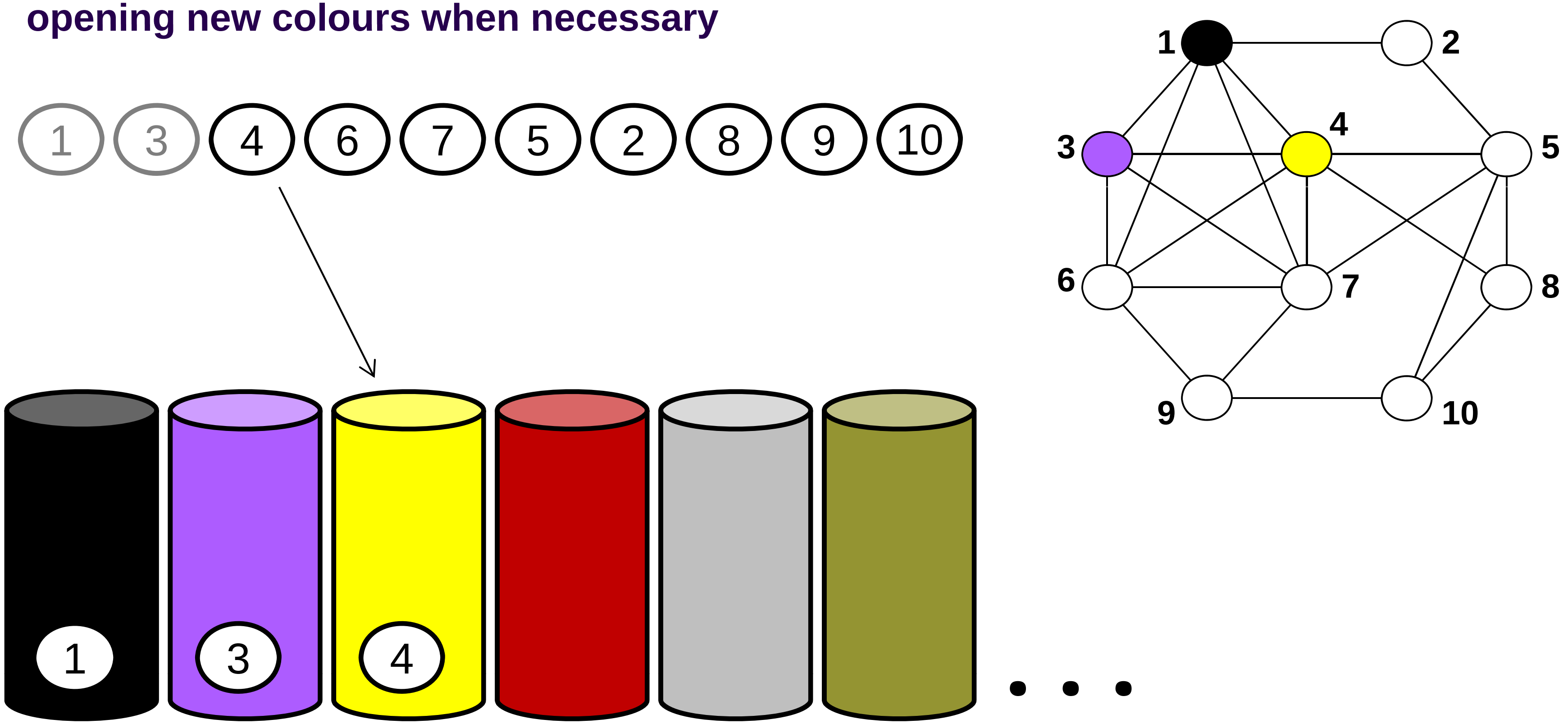
1) Take some permutation of the vertices
2) For each vertex v, assign v it to the first colour seen to be feasible (i.e. no clash is induced), opening new colours when necessary





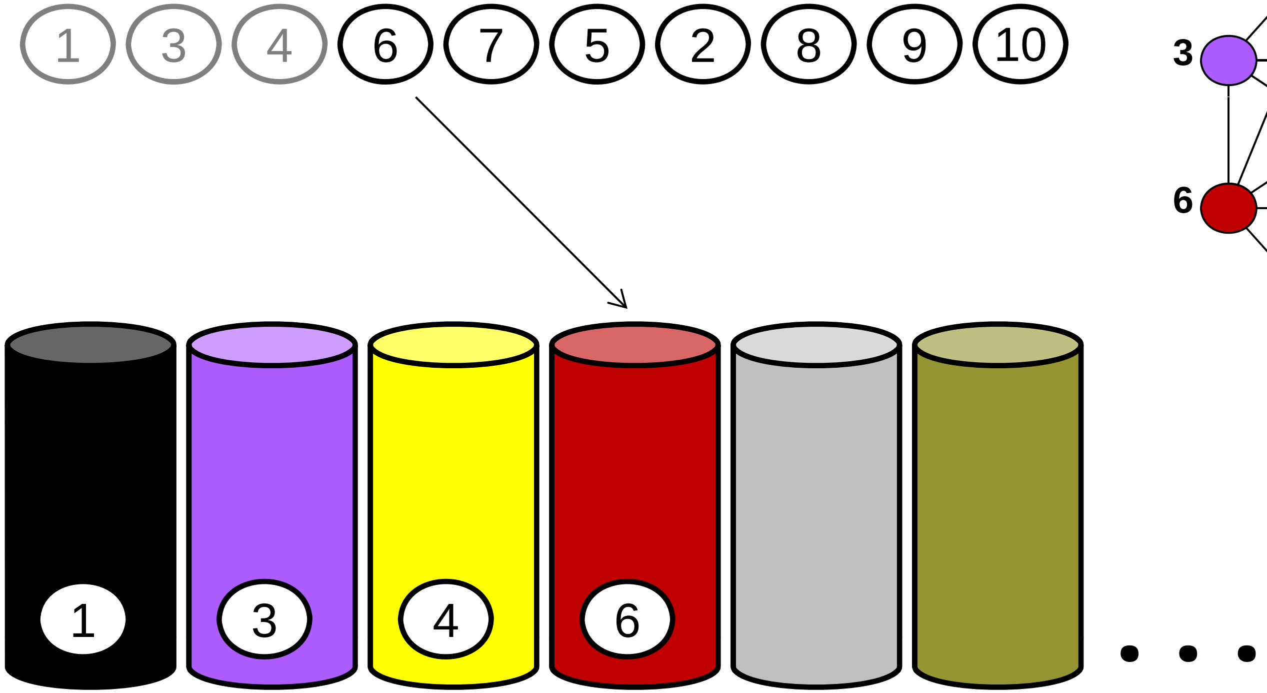
Example: The Greedy Algorithm

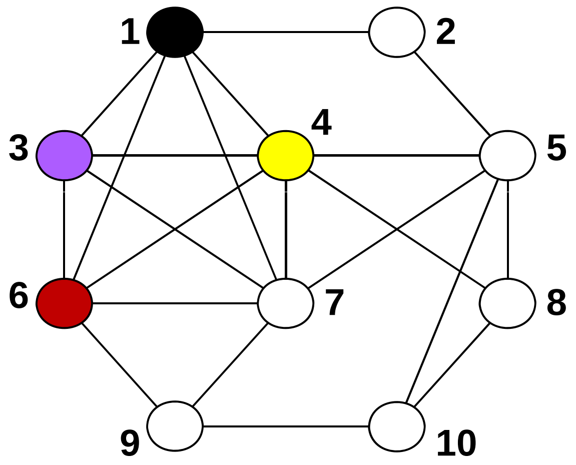
1) Take some permutation of the vertices
2) For each vertex v, assign v it to the first colour seen to be feasible (i.e. no clash is induced), opening new colours when necessary



Example: The Greedy Algorithm

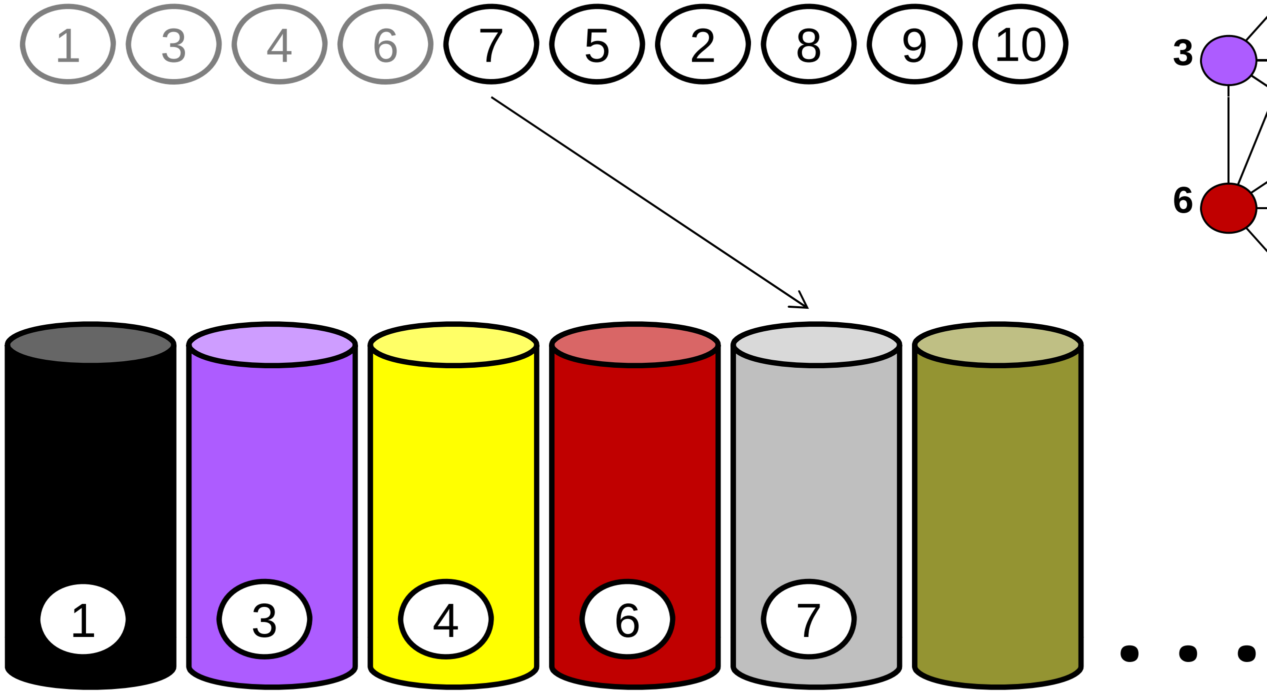
1) Take some permutation of the vertices
2) For each vertex v, assign v it to the first colour seen to be feasible (i.e. no clash is induced), opening new colours when necessary

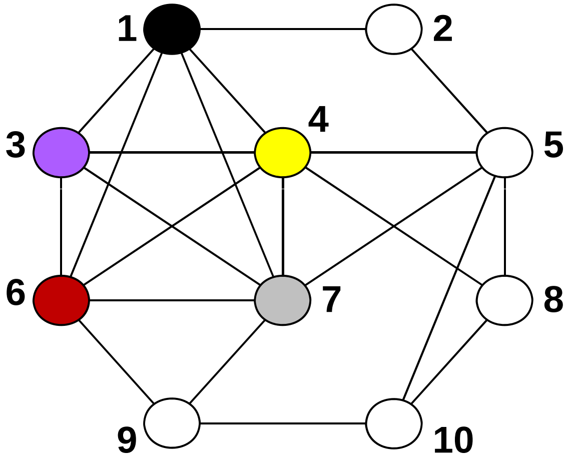




Example: The Greedy Algorithm

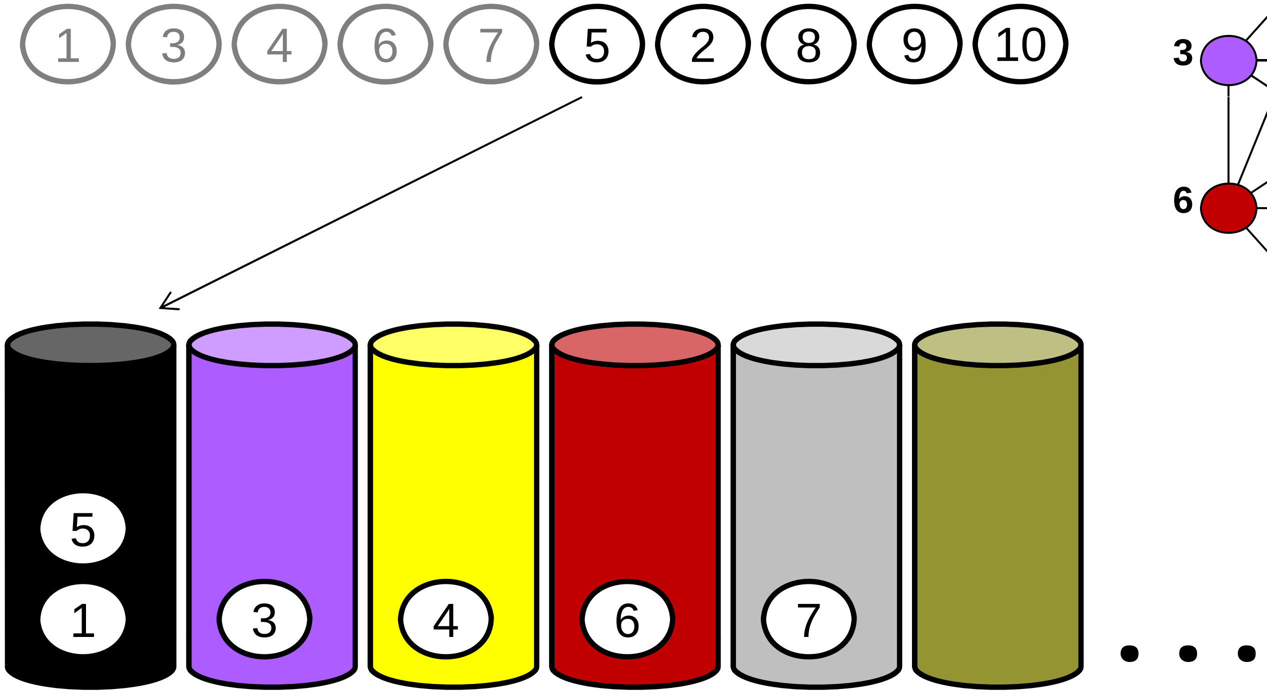
1) Take some permutation of the vertices
2) For each vertex v, assign v it to the first colour seen to be feasible (i.e. no clash is induced), opening new colours when necessary

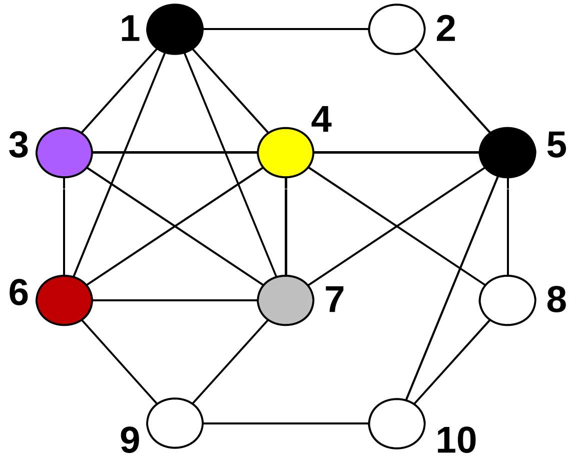




Example: The Greedy Algorithm

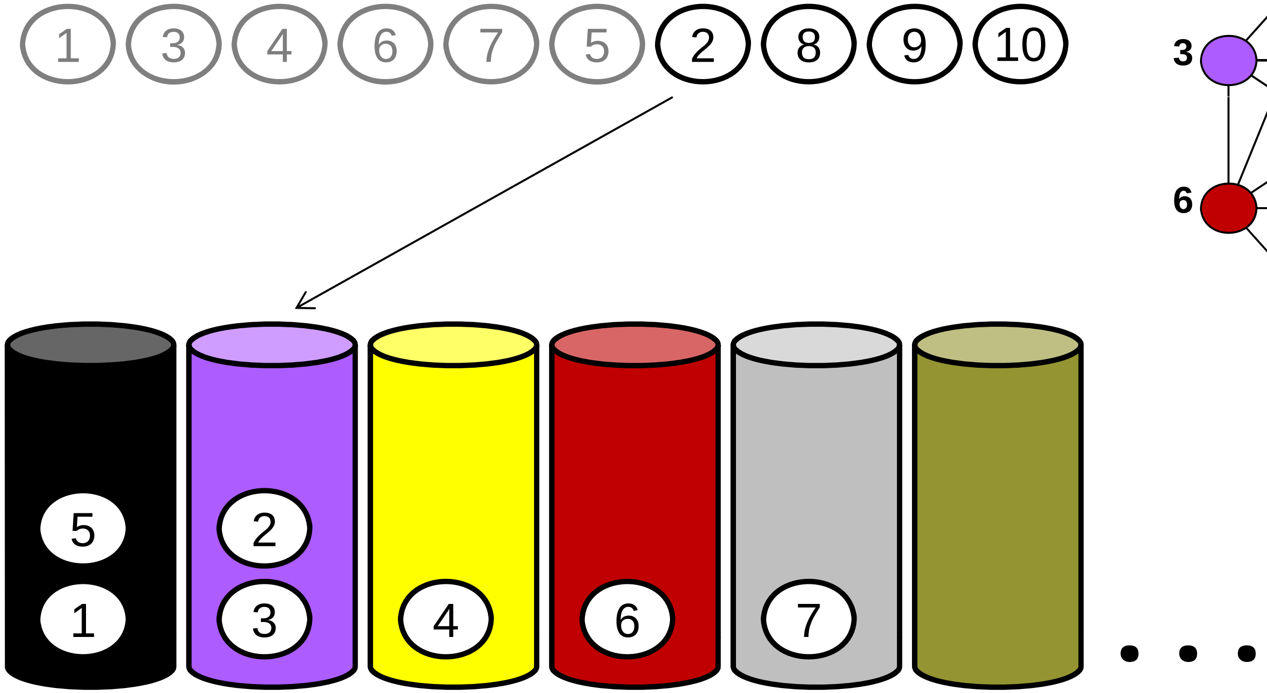
1) Take some permutation of the vertices
2) For each vertex v, assign v it to the first colour seen to be feasible (i.e. no clash is induced), opening new colours when necessary

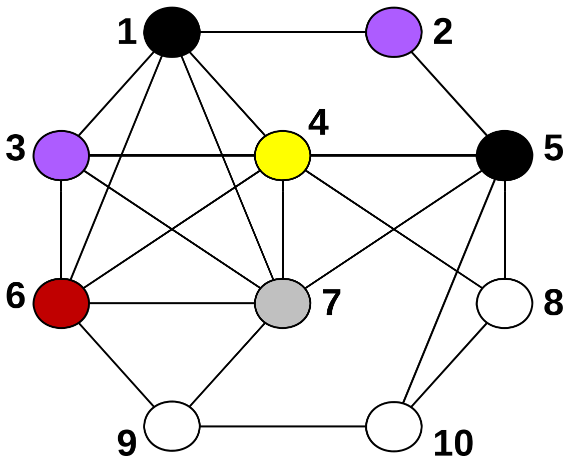




Example: The Greedy Algorithm

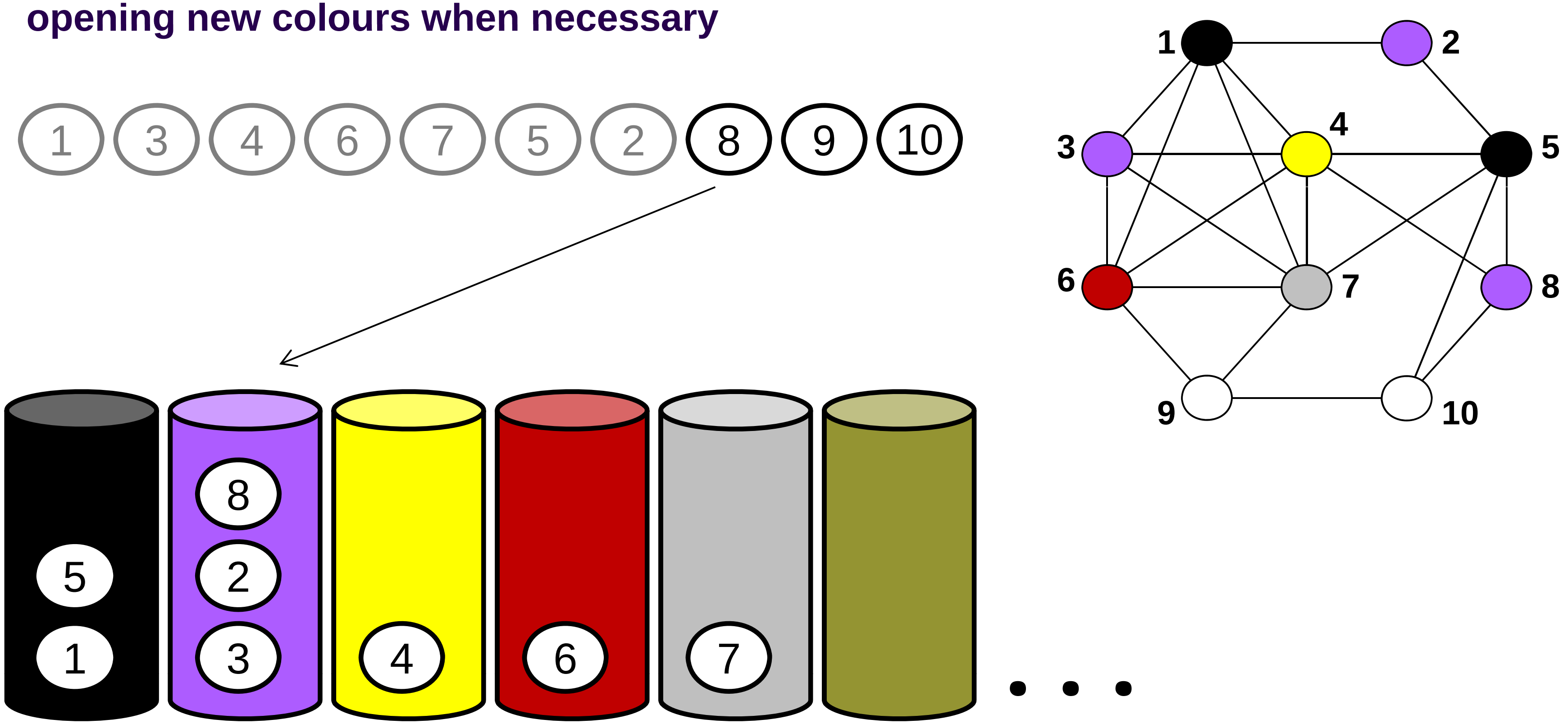
1) Take some permutation of the vertices
2) For each vertex v, assign v it to the first colour seen to be feasible (i.e. no clash is induced), opening new colours when necessary





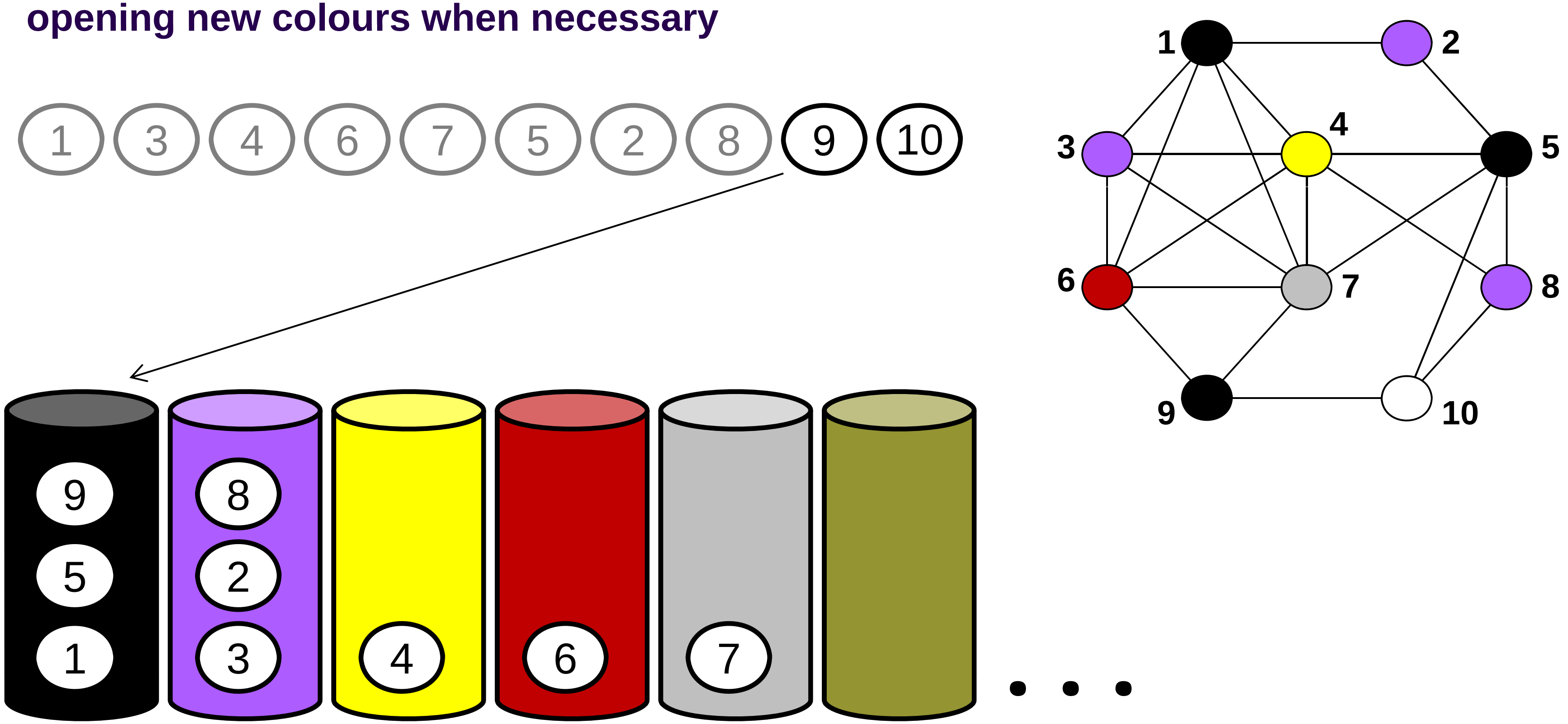
Example: The Greedy Algorithm

1) Take some permutation of the vertices
2) For each vertex v, assign v it to the first colour seen to be feasible (i.e. no clash is induced), opening new colours when necessary



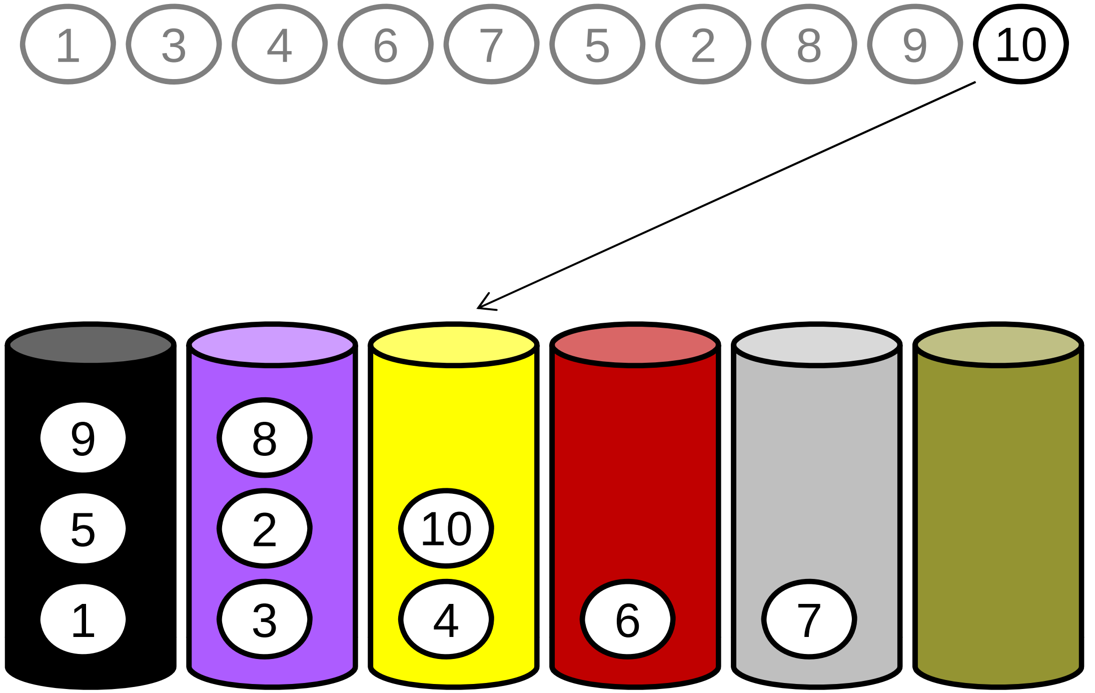
Example: The Greedy Algorithm

1) Take some permutation of the vertices
2) For each vertex v, assign v it to the first colour seen to be feasible (i.e. no clash is induced), opening new colours when necessary

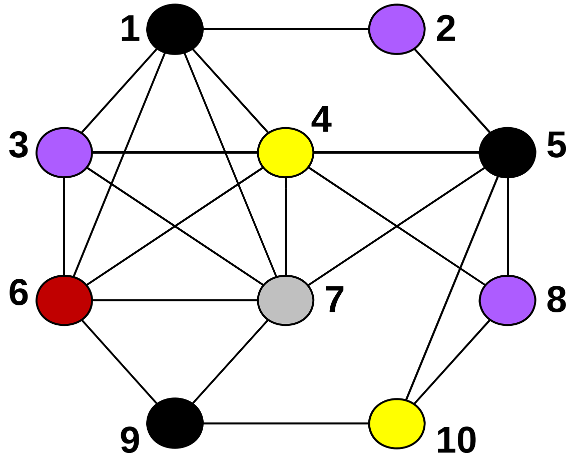


Example: The Greedy Algorithm

1) Take some permutation of the vertices
2) For each vertex v, assign v it to the first colour seen to be feasible (i.e. no clash is induced), opening new colours when necessary



. . .



Solution Optimal!!!
(This won’t always be the case and depends on the ordering in the permutation)

Talk Motivations

- Many different algorithmic schemes have been developed for graph colouring.
- However only a limited set of benchmark instances are typically considered in comparisons (e.g. DIMACS)
- Comparisons between algorithms are also difficult to draw because:
  - Different experimental conditions are used
  - Often only the good results are reported
  - Researchers choose their own cut-off points and only report final (best) solutions
  - Algorithms are often tuned to get the best results on each instance
- To alleviate this, we will
  - Consider a wide variety of problem instance, both artificial and “real world”
  - Compare the algorithms using an unbiased experimental framework with algorithms treated as black-boxes.

Examples of Artificial Graphs

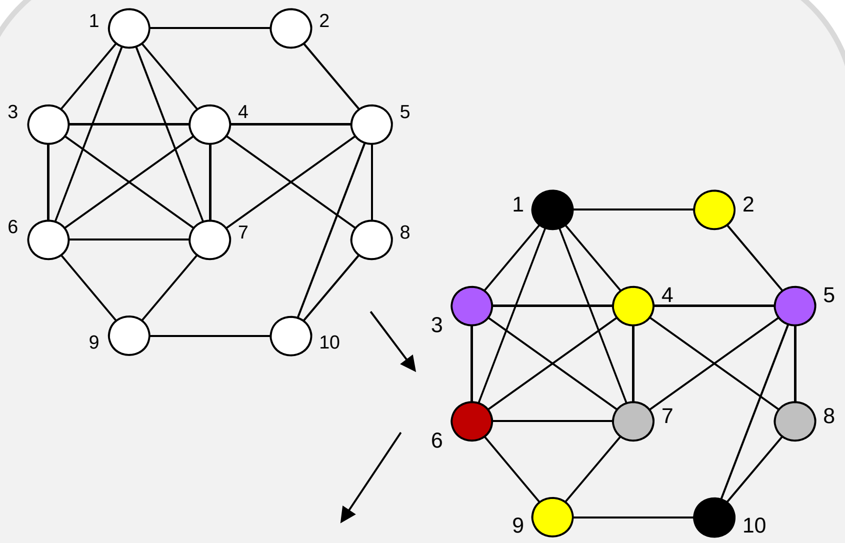
- Random Graphs
  - Start with a graph with **|V|** vertices and no edges.
  - Go through each pair of vertices and add an edge with probability **p**
  - **|V|** and **p** are defined by the user

- Flat graphs
  - Take **|V|** vertices and partition them into **K** equal-sized groups.
  - Now add edges between pairs of vertices in different groups with probability **p**
  - Add edges such that the variance in vertex degrees is kept to a minimum (this makes vertices “look the same”)

- Surprisingly, nearly all algorithmic comparisons in the literature are concerned with just a small number of artificial graphs.

Graph Colouring in Practical OR problems

Educational Timetabling: Arrange lessons into timeslots, avoiding conflicts (e.g. don't ask a student to be in 2 places at once)

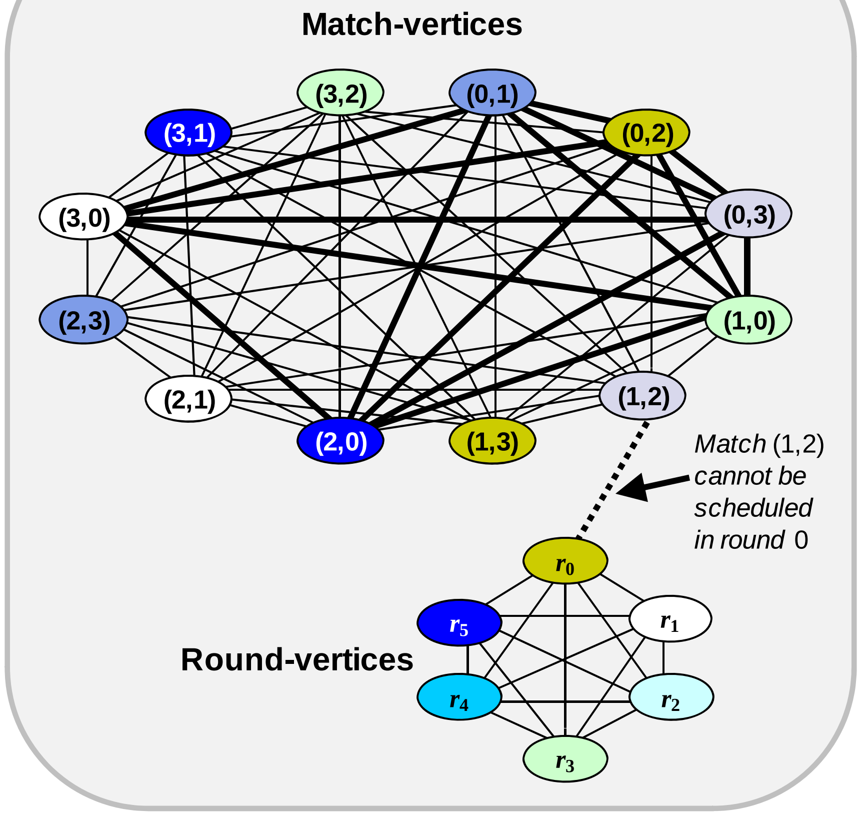


← timeslots →

↑ rooms ↓

	1	2	3	4	5
1	Lesson 1	Lesson 4	Lesson 5	Lesson 7	Lesson 6
2	Lesson 10	Lesson 9	Lesson 3	Lesson 8	
3		Lesson 2			

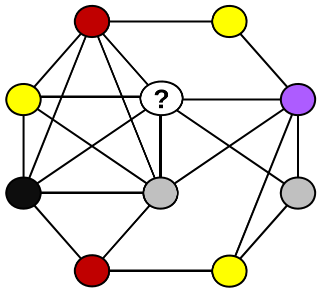
Sports League Scheduling: Schedule a series of matches between teams such that teams play at most once in every round of the league



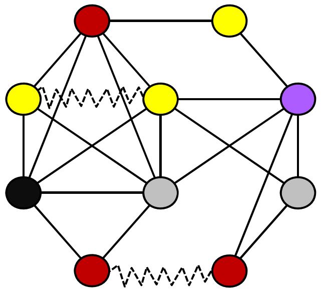
Match (1,2) *cannot be scheduled in round* 0

Some Useful terms

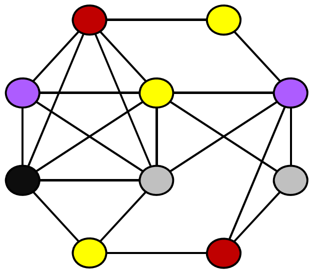
- A colouring is **complete** if all vertices are assigned a colour, else it is **partial**
- A colouring is **proper** if no pair of adjacent vertices is assigned the same colour, else it is **improper**
- A colouring is **feasible** if and only if it is both complete and proper
- A solution is **optimal** if it is feasible, and the number of colours being used is minimal
- The number of colours used in an optimal colouring is termed the **chromatic number**



Partial proper colouring using 5 colours (1 uncoloured)



Complete improper colouring using 5 colours (2 clashes)



Feasible (and optimal) colouring using 5 colours

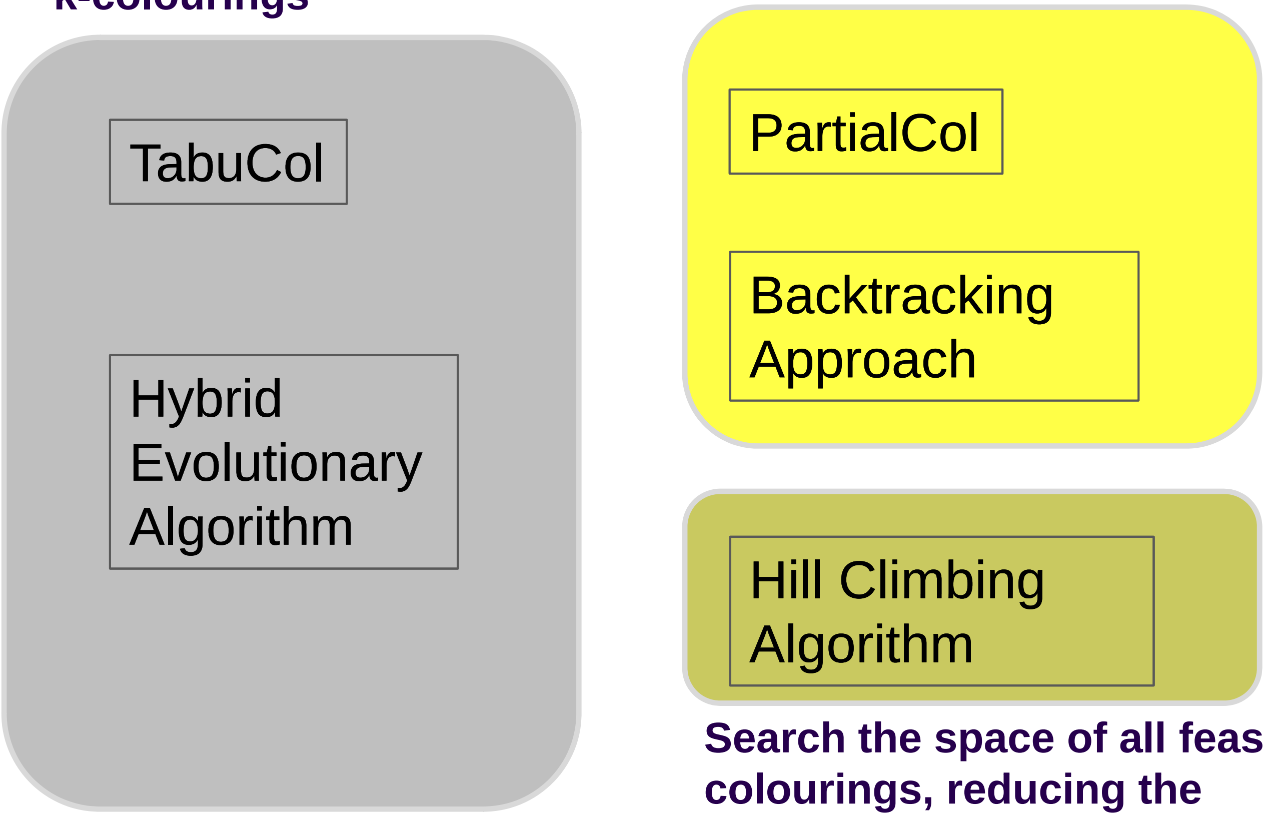
Algorithmic Schemes for Graph Colouring

- Constructive Approaches
    - Assign each vertex a colour one-by-one, backtracking if desired/necesarry
    - E.g. Greedy (first-fit) algorithm, Dsatur, recursive largest first (RLF).
- Optimisation Based Methods
    - Feasible-only Search Space
        - Explore the space of feasible solutions and attempt to reduce the number of colours.
    - Space of complete, improper solutions
        - Colour all vertices using a fixed number of colours and attempt to eliminate all clashes
        - Reduce the number of colours if successful
    - Space of partial, proper solutions
        - Colour all vertices using a fixed number of colours. If no colour is available for a vertex (all colours would induce a clash), leave it uncoloured. Now attempt to eliminate all uncoloured vertices.
        - Reduce the number of colours if successful
    - Combinations of these are also possible…

Algorithms Tested

- **TabuCol** ***(Hertz and de Werra, 1987)***
  - Search the space of complete improper **k**-colourings using tabu search, attempting to eliminate clashes.
  - Decrease **k** and restart when a colouring with zero clashes is achieved
- **PartialCol** ***(Blochliger and Zufferey, 2008)***
  - Search the space of partial, proper **k**-colourings using tabu search, attempting to colour uncoloured vertices while never allowing any clashes
  - Decrease **k** and restart when a complete colouring is achieved
- **Hybrid Evolutionary Algorithm** ***(Galinier and Hao, 1999)***
  - Evolve a population of complete, improper **k**-colourings via a specialised crossover operator
  - Locally improve each solution using **TabuCol** for a fixed number of iterations
- **Hill Climbing Approach** ***(Lewis, 2009)***
  - Search the space of feasible colourings, attempting to reduce the number of colours being used;
  - Only perform alterations to a solution that maintain feasibility (e.g. Kempe chain moves)
- **Backtracking Approach** ***(Brelaz, 1979; Korman, 1979)***
  - Produce an initial solution using the Dsatur heuristic
  - Attempt to improve the solution by systematically reassigning vertices to different colours
  - This is a complete algorithm given excess time
- **AntCol** (Based on Ant Colony Optimisation. Reported in paper but not considered here)

Algorithms Tested



Search the space of complete, improper k-colourings

TabuCol

Hybrid Evolutionary Algorithm

Search the space of partial, proper k-colourings

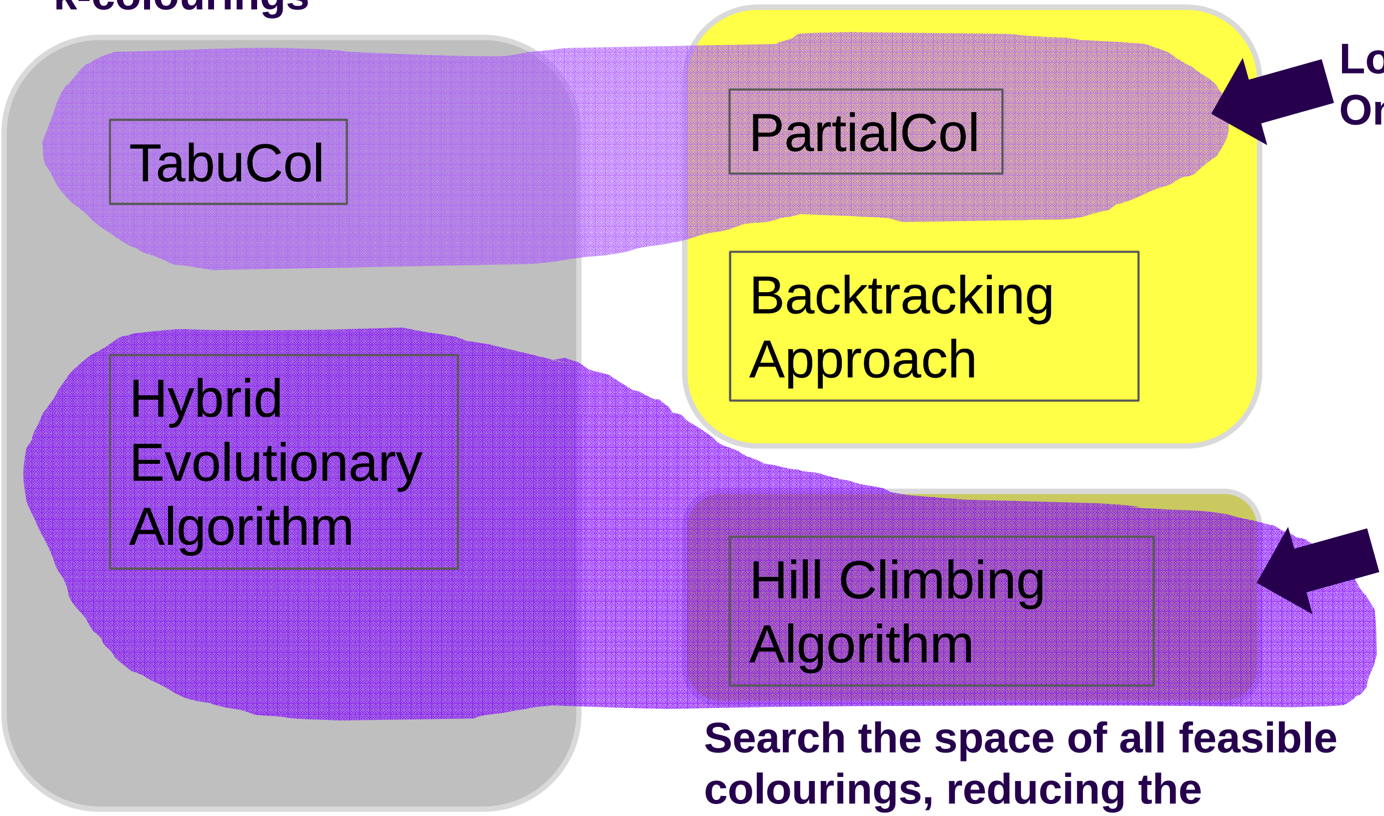
PartialCol

Backtracking Approach

Hill Climbing Algorithm

Search the space of all feasible colourings, reducing the number of colours being used

Algorithms Tested



Search the space of complete, improper k-colourings

Search the space of partial, proper k-colourings

TabuCol

PartialCol

Backtracking Approach

Hybrid Evolutionary Algorithm

Hill Climbing Algorithm

Local Search Only

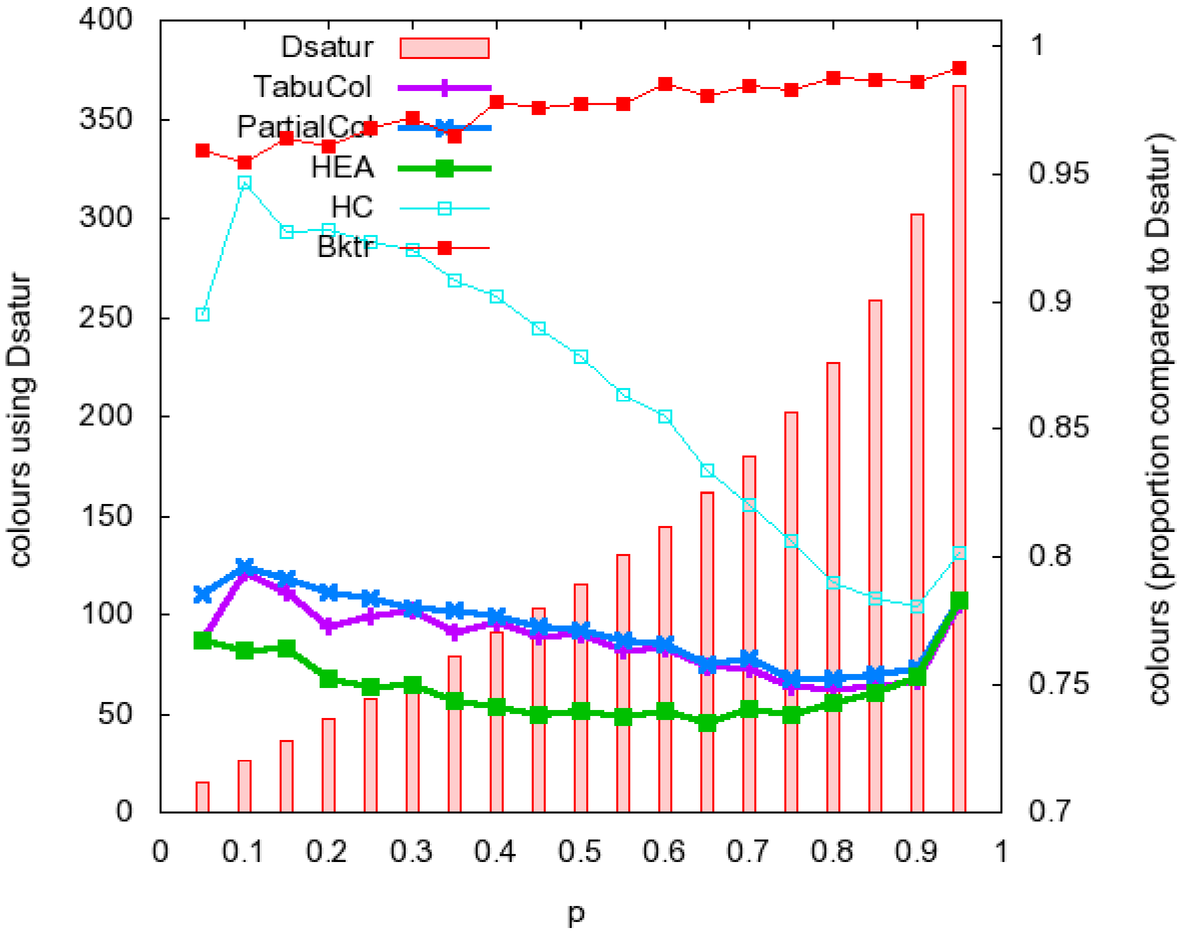
Local Search hybridised with “large move operators”

Search the space of all feasible colourings, reducing the number of colours being used

Experimental Setup

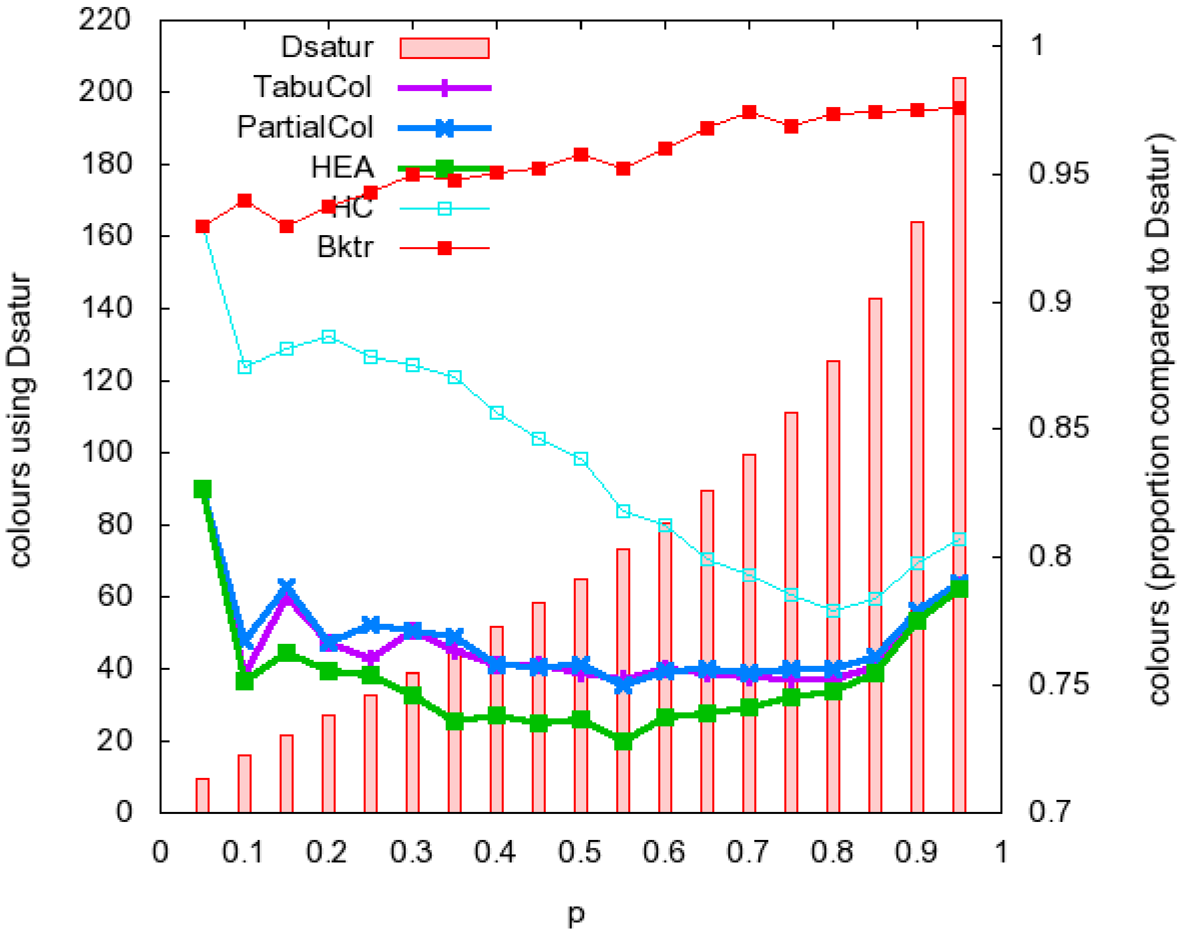
- 6 different algorithms (+variants); 5000 different problem instances ("artificial" and "real world"); 40,000+ individual trials.
- Over a decade of computing time consumed in total
- Platform-independent measure of computational effort used (number of constraint-checks)
- Large cut-off points used in each trial to give some notion of excess time (5 x 10^{11} checks – at least a few hours for each trial)
- Algorithms treated as black-boxes: no "manual tuning" for different instances

Performance on Random Graphs (|V|=1000)



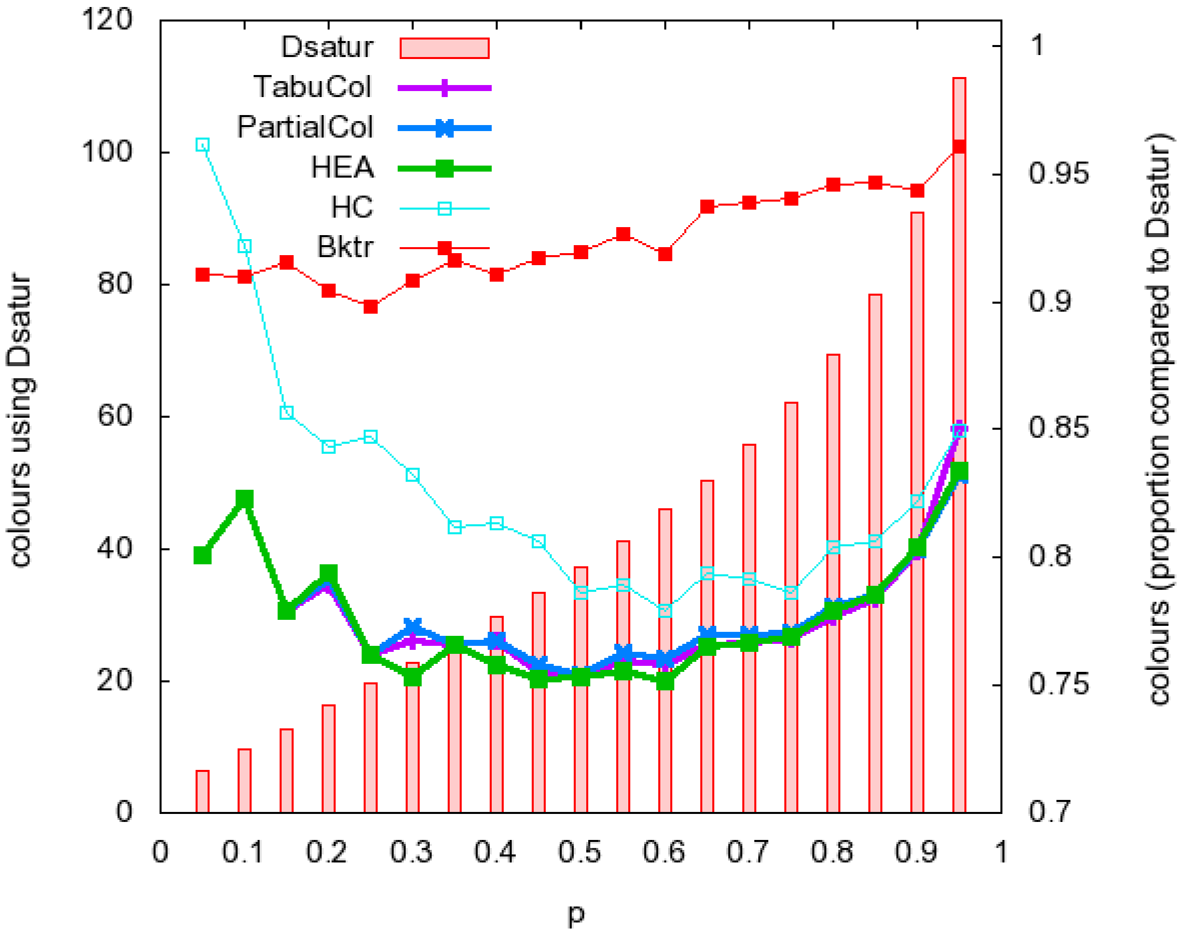
- Clearly, methods that explore infeasible search-spaces (partial-proper or complete-improper) are more successful
- Feasible-only space (HC) features lower connectivity and brings worse results
- Backtracking algorithm not competitive
- HEA's combination of LS and global operators is favourable

Performance on Random Graphs (|V|=500)



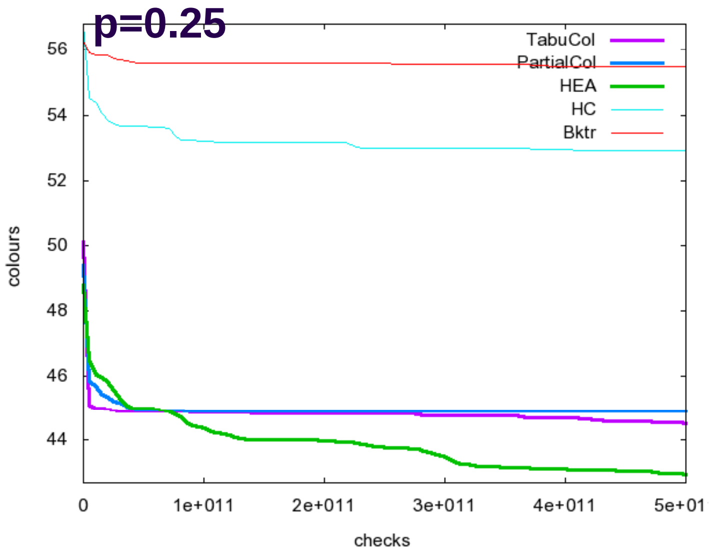
- Clearly, methods that explore infeasible search-spaces (partial-proper or complete-improper) are more successful
- Feasible-only space (HC) features lower connectivity and brings worse results
- Backtracking algorithm not competitive
- HEA's combination of LS and global operators is favourable

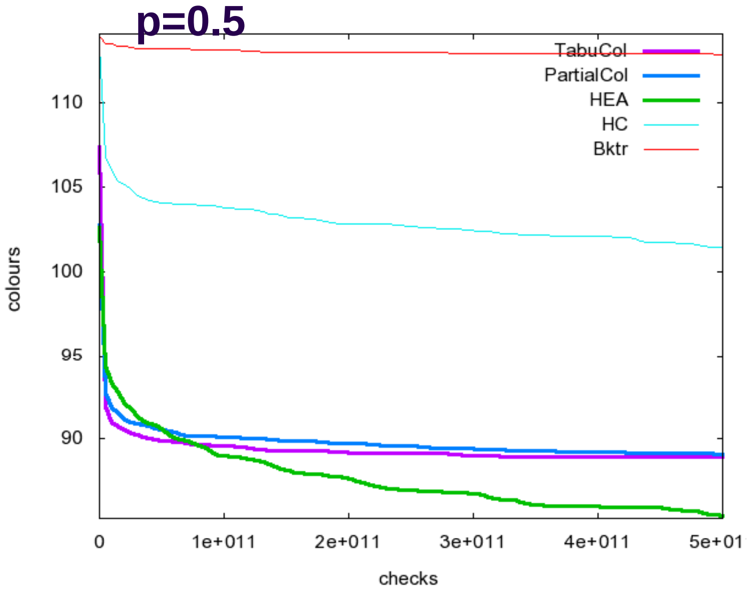
Performance on Random Graphs (|V|=250)

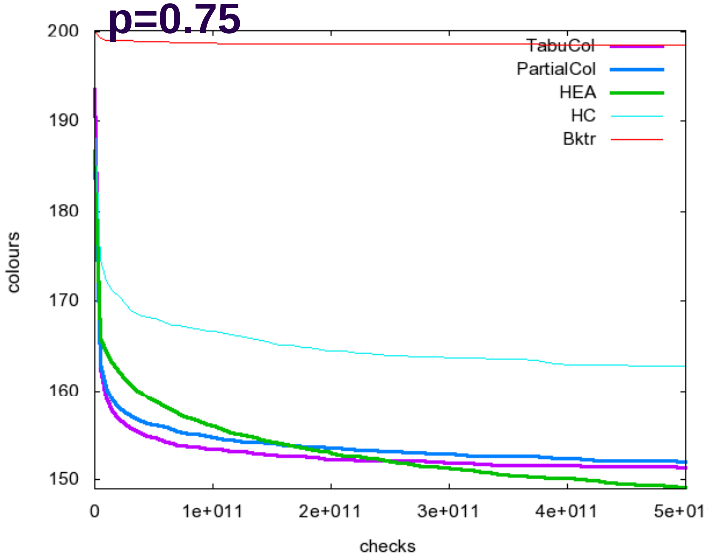


- Clearly, methods that explore infeasible search-spaces (partial-proper or complete-improper) are more successful
- Feasible-only space (HC) features lower connectivity and brings worse results
- Backtracking algorithm not competitive
- HEA's combination of LS and global operators is favourable

Performance on Random Graphs over time (|V|=1000)


p=0.25

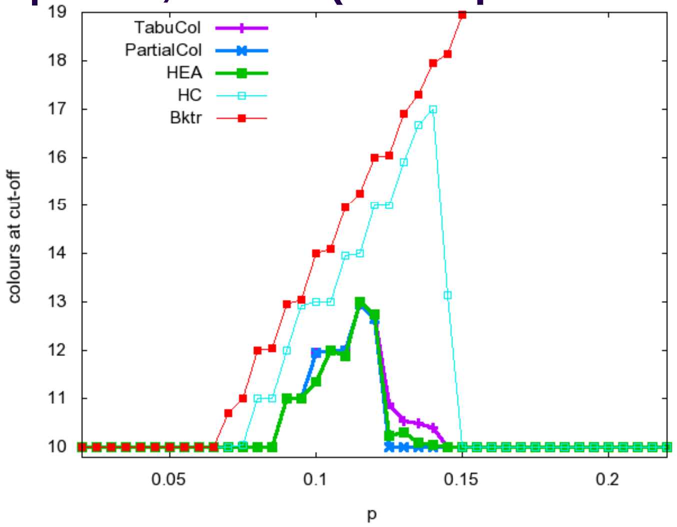

p=0.5


p=0.75

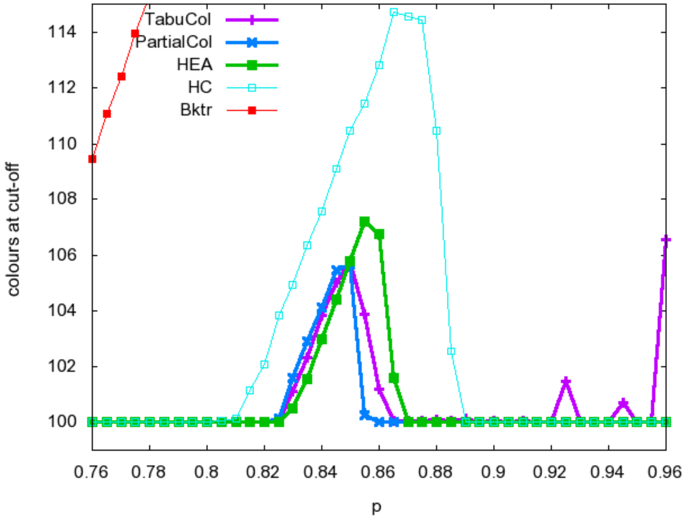
- Over time, local-search based methods make the quickest improvements
- However, given enough time the HEA returns better results
- Hill Climbing method and Backtracking are again inferior

Performance on Flat Graphs

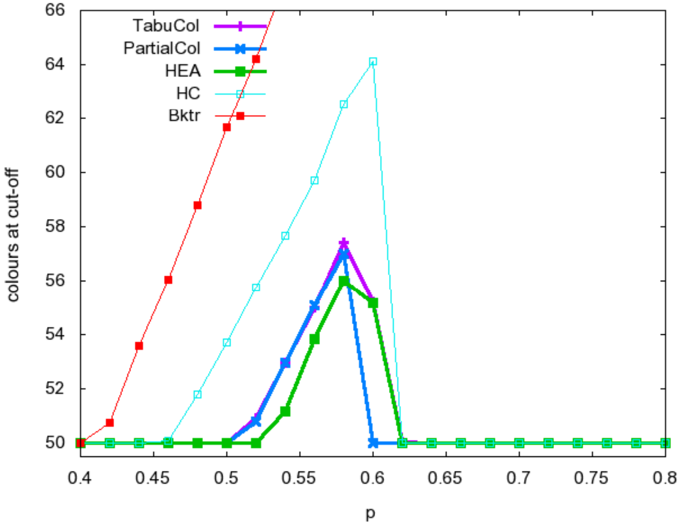
|V| = 500, k = 10. (50 v’s-per-colour)



|V| = 500, k = 100. (5 v’s-per-colour)



|V| = 500, k = 50. (10 v’s-per-colour)



- Experiments with flat graphs within the phase transition regions reveal similar characteristics to random graphs

Analysis of Results

- Clearly, better results are gained by algorithms that explore spaces of infeasible colourings (i.e. TabuCol, PartialCol, HEA)
- HEA, which also involves a global search operator (recombination) is superior to local search on its own in some cases.
- In contrast, the feasible-only search space is more disconnected, thus improvements are more difficult to make
- However, random and flat graphs are very particular types of graphs that feature high levels of vertex-degree homogeneity
  - For example, degrees of vertices in random graphs are modelled by D ~ **B**(|V| – 1, p), giving low variance for high and low p-values;
  - For the flat graphs, degree variance is also kept deliberately low;
  - In the considered instances, the degree coefficient of variation (CV = SD / mean x 100) is always less than 29%
- Do the same performance features hold for other types of graph?

Some real world timetabling instances (Carter 1996)

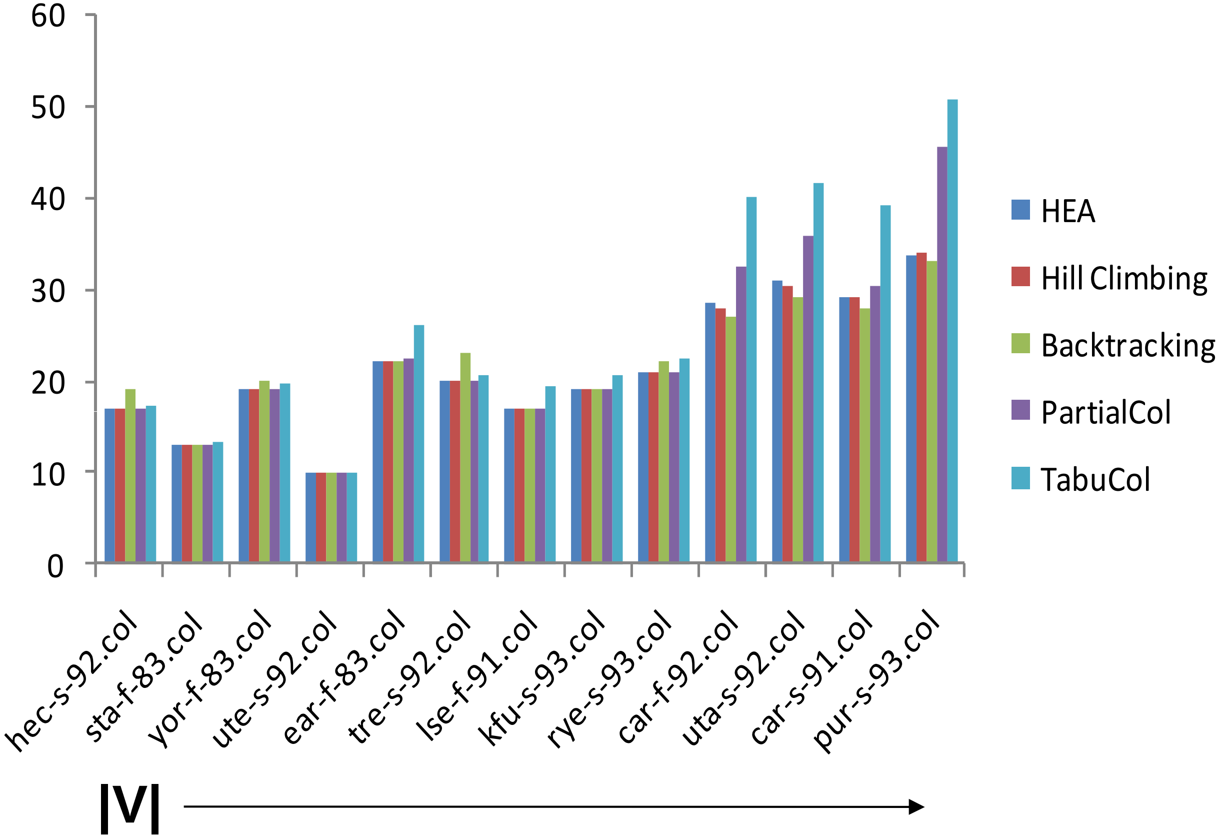
Name	|V|	Density	Degree (min;med;max)	Degree mean	Degree CV
hec-s-92	81	0.415	9; 33; 62	33.7	36.30%
sta-f-83	139	0.143	7; 16; 61	19.9	67.40%
yor-f-83	181	0.287	7; 51; 117	52	35.20%
ute-s-92	184	0.084	2; 13; 58	15.5	69.10%
ear-f-83	190	0.266	4; 45; 134	50.5	56.10%
tre-s-92	261	0.18	0; 45; 145	47	59.60%
lse-f-91	381	0.062	0; 16; 134	23.8	93.20%
kfu-s-93	**461**	**0.055**	**0; 18; 247**	**25.6**	**120.00%**
rye-s-93	486	0.075	0; 24; 274	36.5	111.80%
car-f-92	543	0.138	0; 64; 381	74.8	75.30%
uta-s-92	622	0.125	1; 65; 303	78	73.70%
car-s-91	682	0.128	0; 77; 472	87.4	70.90%
pur-s-93	2419	0.029	0; 47; 857	71.3	129.50%

- *“Consider the instance* ***kfu-s-93****, by no means the hardest or largest in the set. It involves 5349 students sitting 461 exams, ideally fitted into 20 timeslots. The problem contains 2 cliques of size 19 and huge numbers of smaller ones. There are 16 exams that clash with over 100 others.”* (Ross et al. 2003).

Performance of Algorithms on the 13 timetabling Instances

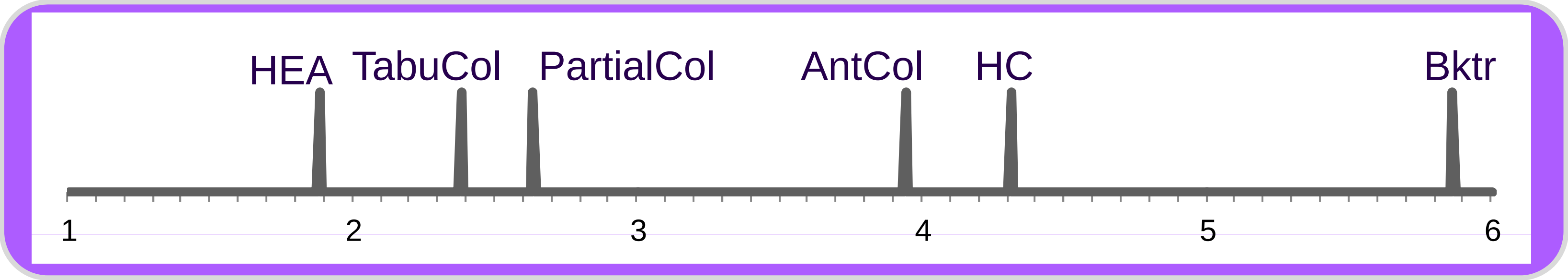
- Here, backtracking is able to produce competitive results (more heuristic information available)
- The methods that combine LS **and** global operators are the most consistent
- Methods based solely on LS produce disappointing results…

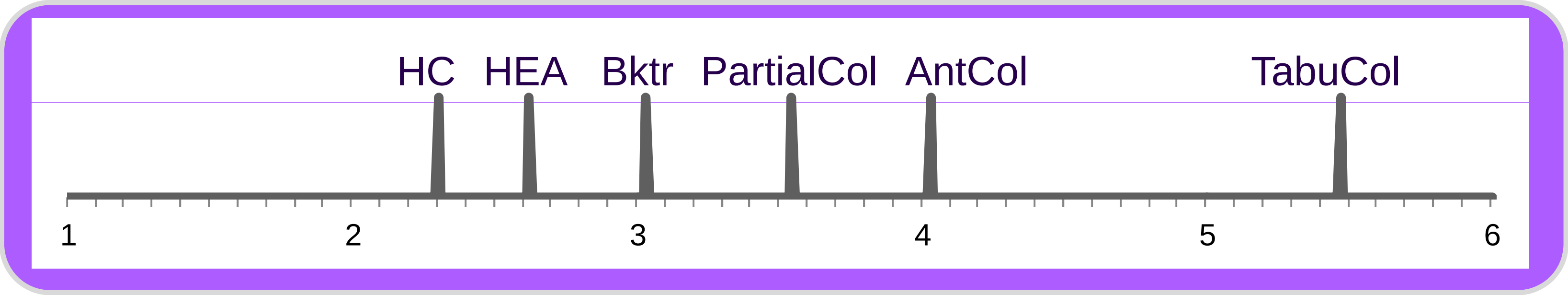
Number of colours at cut-off point (mean of 50 runs on each instance)



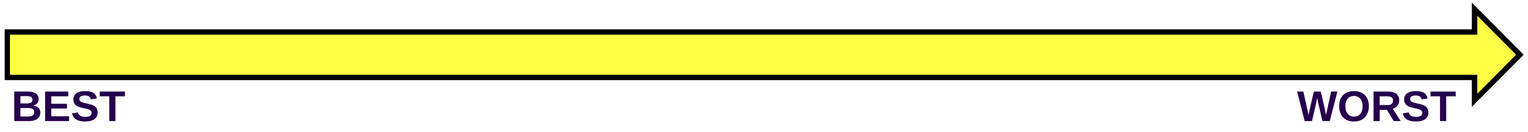
Performance of Algorithms on the 13 timetabling Instances

Average rank (compared to other algorithms) across all random and flat graphs





Average rank (compared to other algorithms) across the 13 timetabling graphs

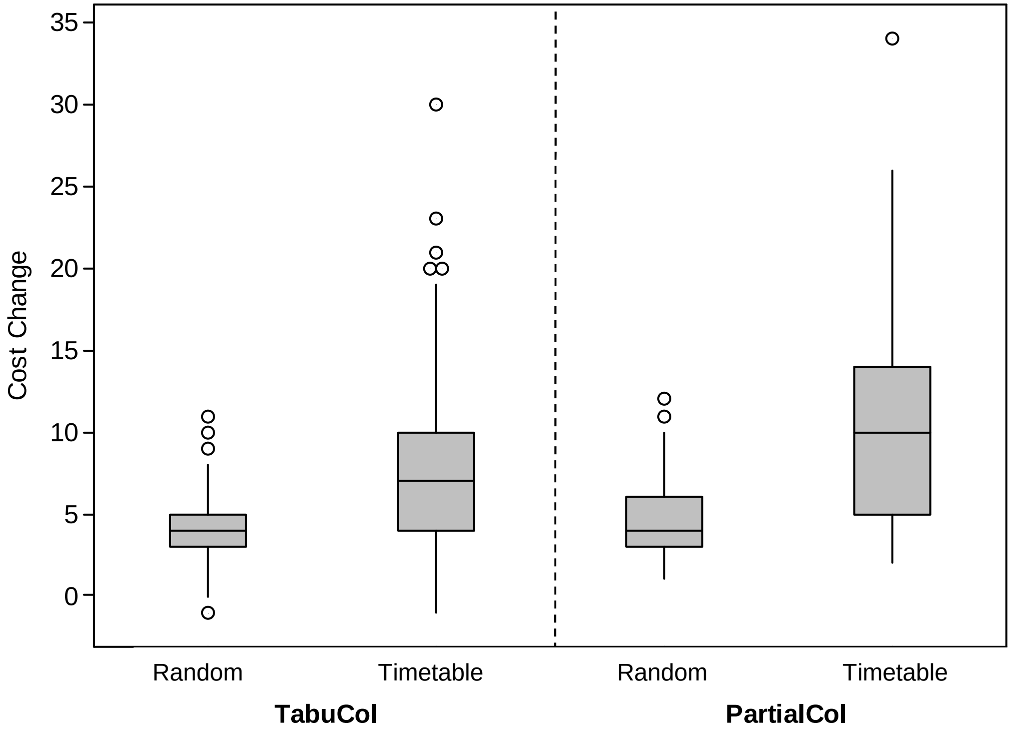


Performance of Algorithms on the 13 timetabling Instances

- The higher degree-variance in timetabling graphs tends to increase the variance in cost

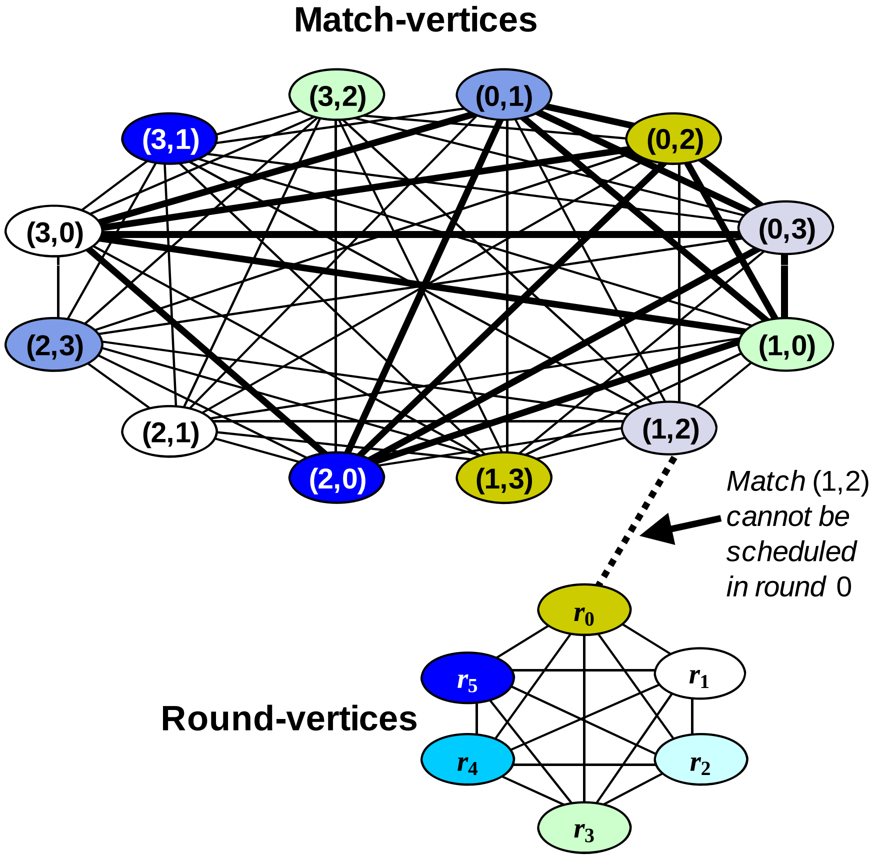
- This implies a more spiky cost landscape that the LS techniques on their own seem to have difficulty coping with

Cost change distributions for a random graph and a timetabling graph with similar densities (0.15 and 0.138 resp.)



Sports Scheduling

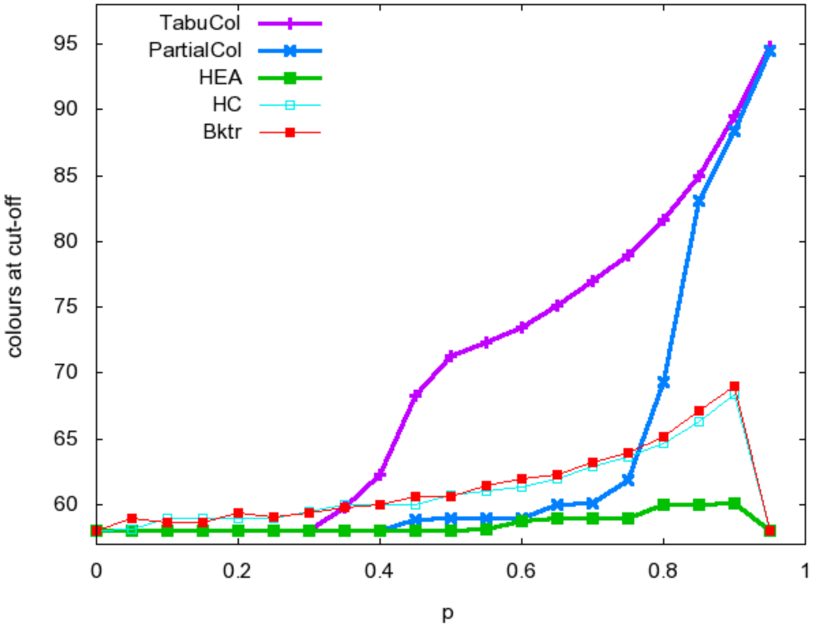
Sports scheduling problem with 4 teams playing each other twice (n=4; m=2)



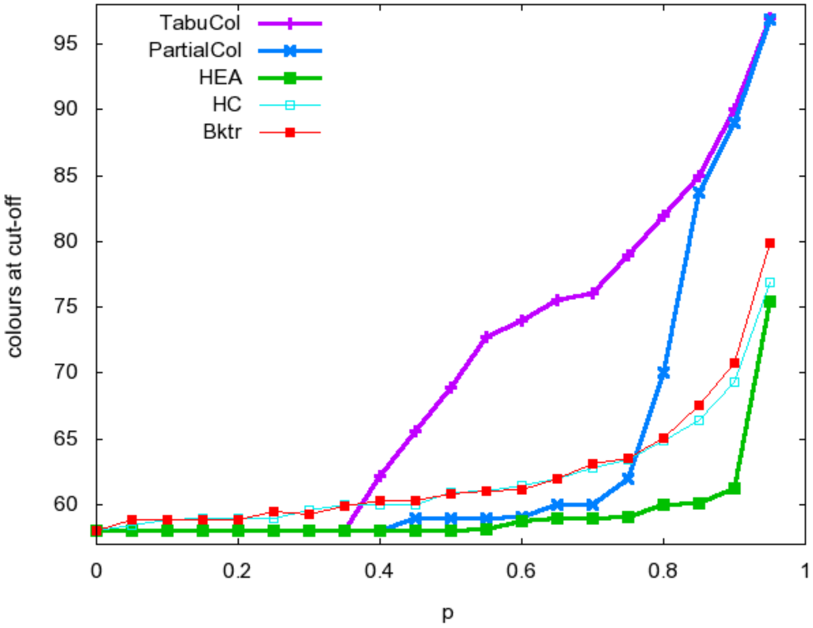
- Given **n** teams, arrange matches so that each team plays each other **m** times in **m(n – 1)** rounds
- Represented as a graph by introducing **½mn(n – 1)** vertices, one for each match
- Edges are imposed between matches that cannot be played simultaneously
- Extra vertices can also be added to add further constraints concerning the rounds
- In our case, edges are added between match- and round-vertices with probability **p**

Sports Scheduling

Problem using n=30, m=2, (|V| = 928) with chromatic number = 58



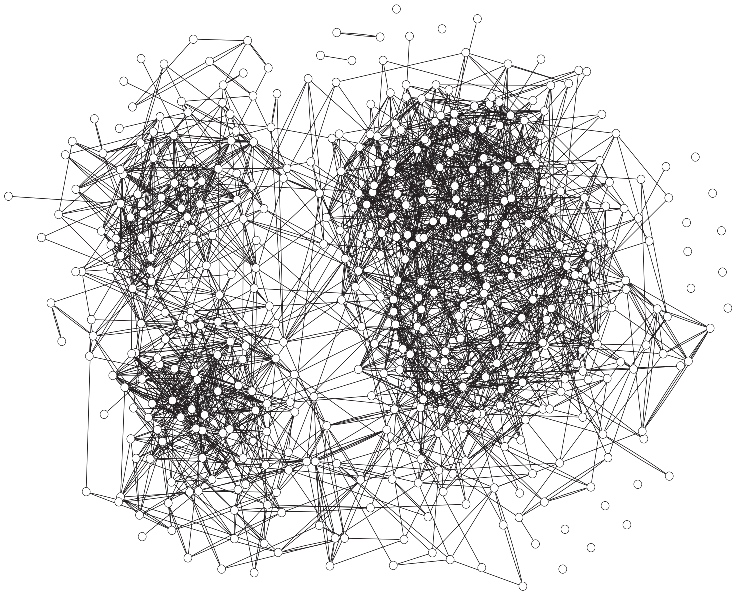
Problem using n=30, m=2, (|V| = 928) with unknown chromatic number

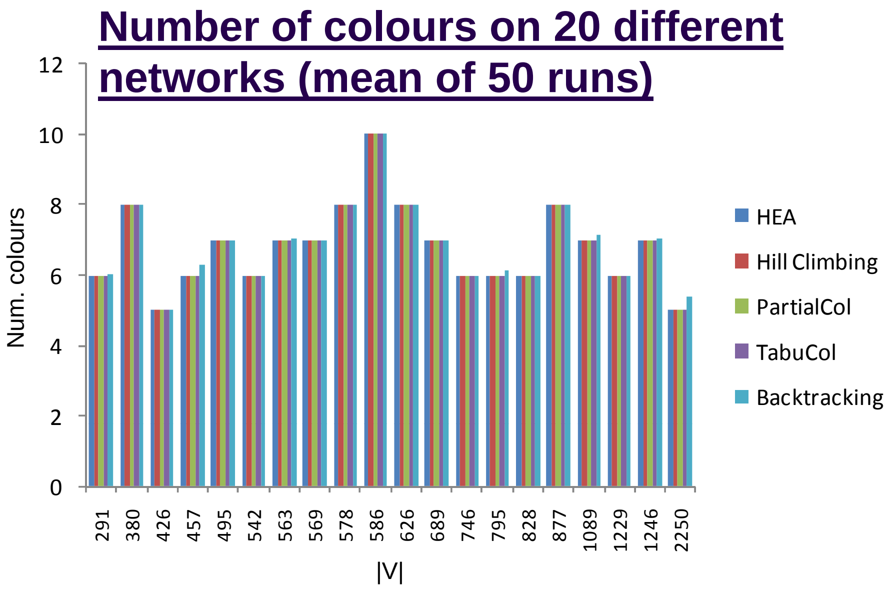


- Increases to **p** also lead to increases in degree variance (and coefficient of variation)
- Methods based solely on LS clearly perform poorly when **p** is high
- These issues are alleviated when using a global operator (as with HEA)

Social Networks

- Take all the pupils from a school and add edges between those who are friends
- Group children so they are away (or with) their friends
- Such graphs appear to be easy to colour optimally (sparse, low degrees, regular structures)





Number of colours on 20 different networks (mean of 50 runs)

General Thoughts…

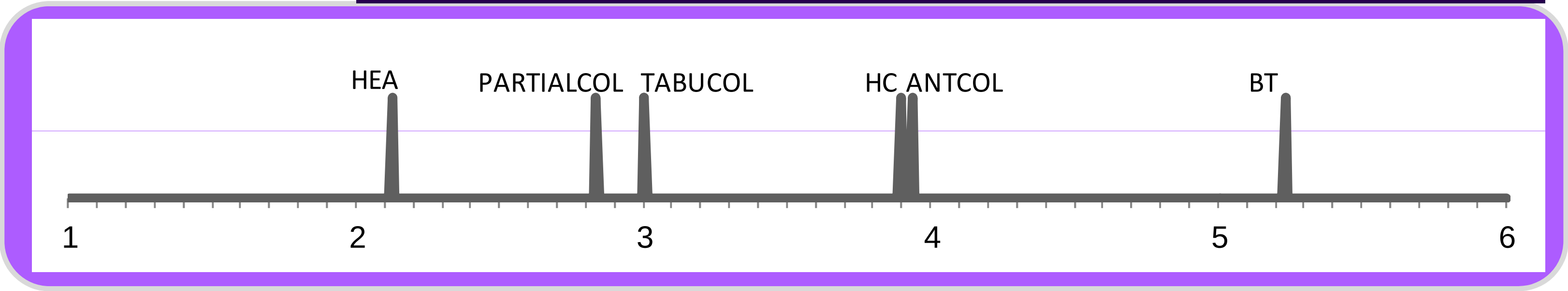
- Has the field of graph colouring been too focussed on "artificial" instances?

- Graph-structure seems to have a large effect on algorithm performance. However this is often hard to assess

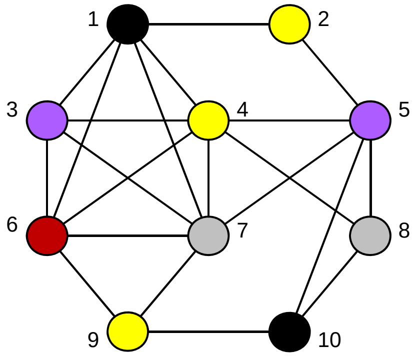
- Methods relying solely on local search are not always sufficient (e.g. graphs with spiky landscapes)

- HEA generally the best approach from our trials
  - Recombination operator exploits the problem structure
  - Combination of LS and global operators
  - **Search space connectivity is higher than the alternative "feasible-only space"**

Average ranking over all instance classes (equally weighted)



Case Study: Post Enrolment-based course timetabling



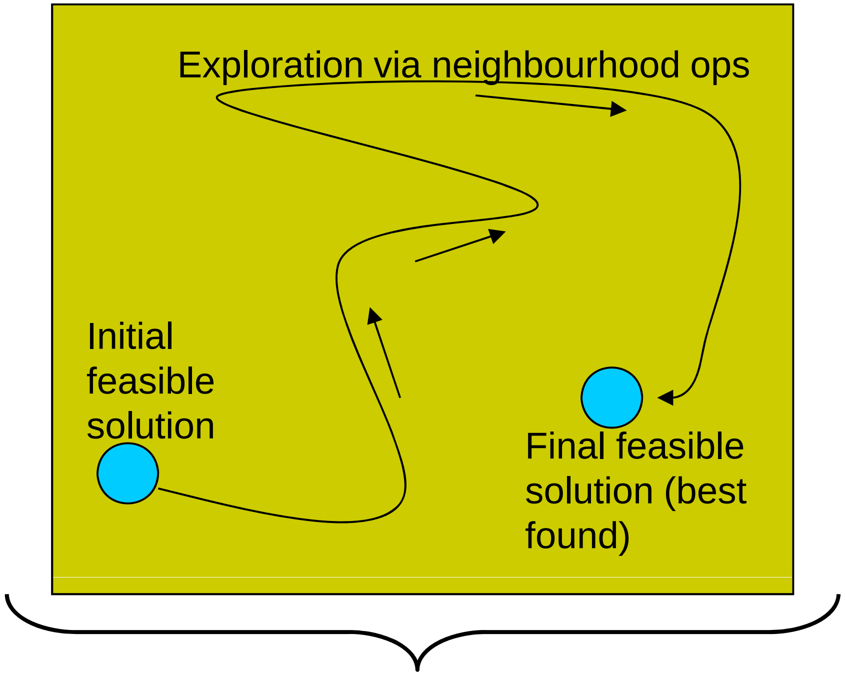
← timeslots →

← rooms →

	1	2	3	4	5
1	Lesson 1	Lesson 4		Lesson 7	
2		Lesson 9	Lesson 3	Lesson 8	
3	Lesson 10	Lesson 2	Lesson 5		Lesson 6

- Used in the International Timetabling Competition 2007
- The problem features a number of “hard” (mandatory) constraints and “soft” (optional) constraints.
- The problem is related to graph colouring, though room availability in each timeslot is also important (amongst other things).
- A maximum matching algorithm can be used to determine whether all vertices in a particular colour class can be assigned to acceptable rooms in a timeslot.

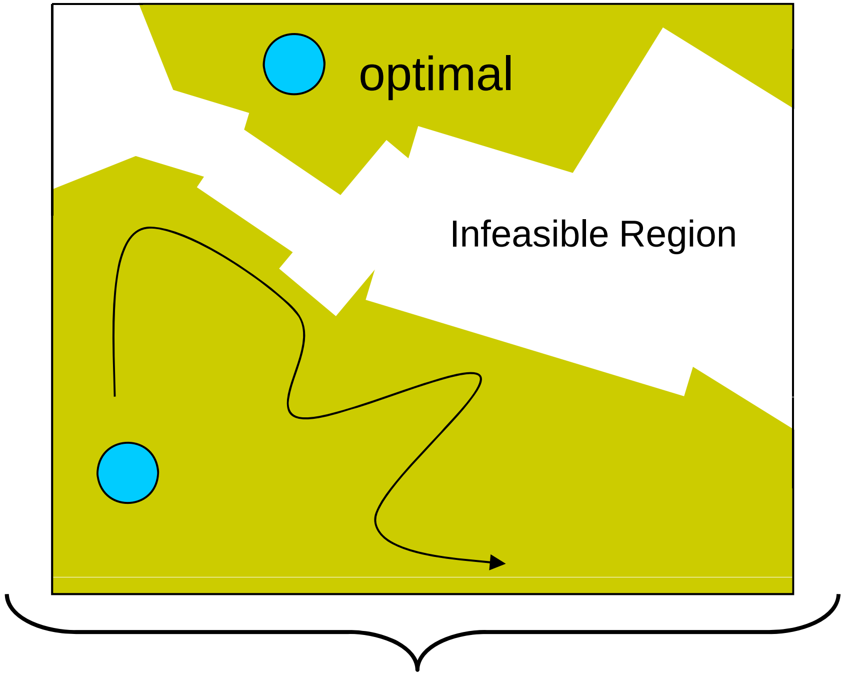
Solution Strategy


Exploration via neighbourhood ops
Initial feasible solution
Final feasible solution (best found)

Space of all feasible solutions

- First, produce a feasible solution (one that obeys all hard constraints)
- Next, search the space of feasible solutions attempting to eliminate soft constraint violations

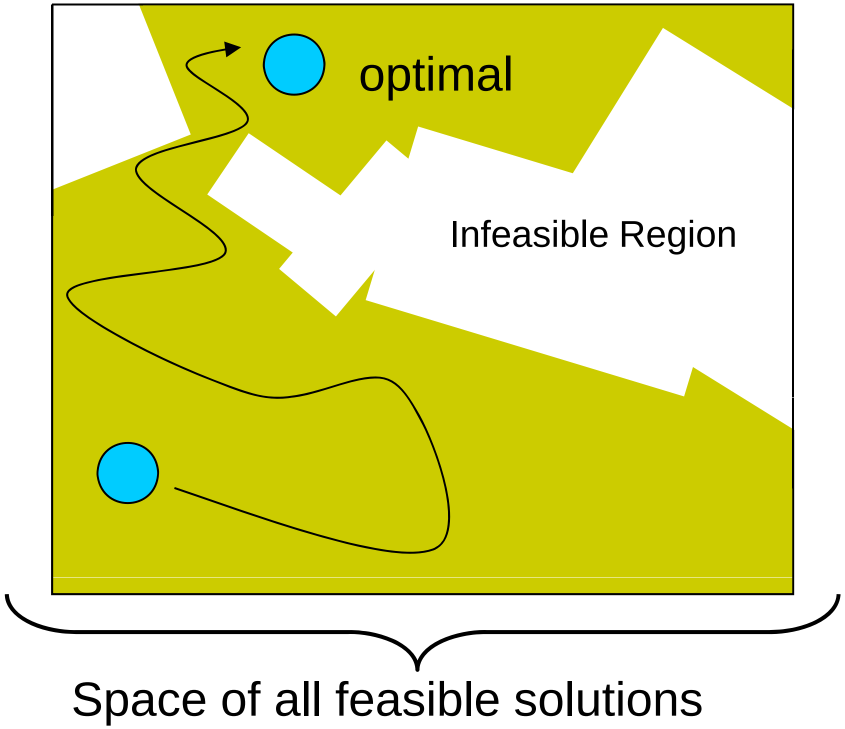
Solution Strategy



Space of all feasible solutions

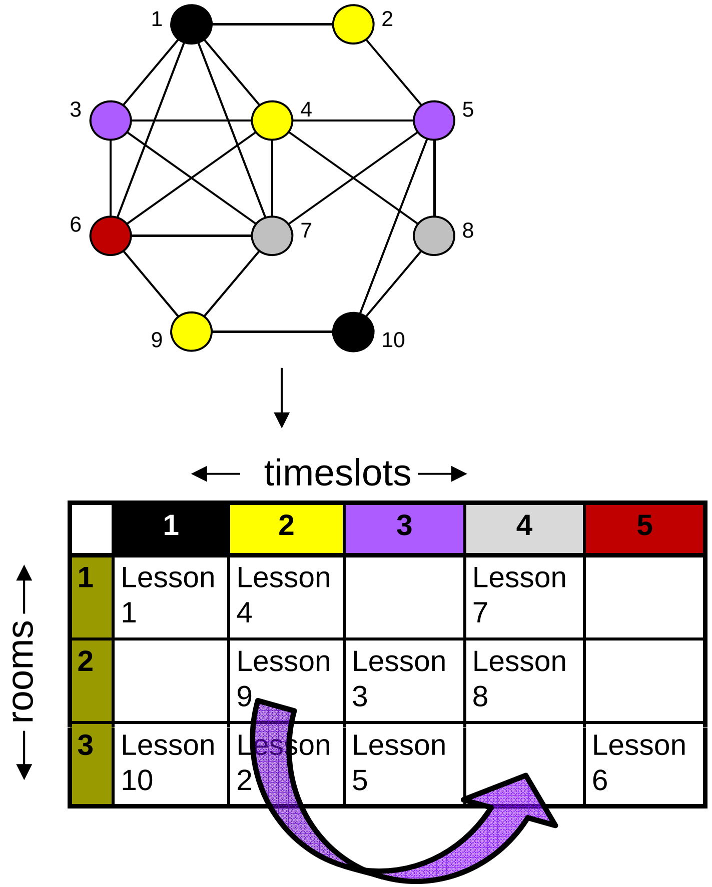
- First, produce a feasible solution (one that obeys all hard constraints)
- Next, search the space of feasible solutions attempting to eliminate soft constraint violations
- **Depending on the problem and method of exploration, the "feasible-only" search space may feature low levels of connectivity (or could be disconnected)**

Solution Strategy



- First, produce a feasible solution (one that obeys all hard constraints)
- Next, search the space of feasible solutions attempting to eliminate soft constraint violations
- Depending on the problem and method of exploration, the “feasible-only” search space may feature low levels of connectivity (or could be disconnected)
- **It thus makes sense to increase this connectivity if possible**

Simple Neighbourhood search algorithm for timetabling

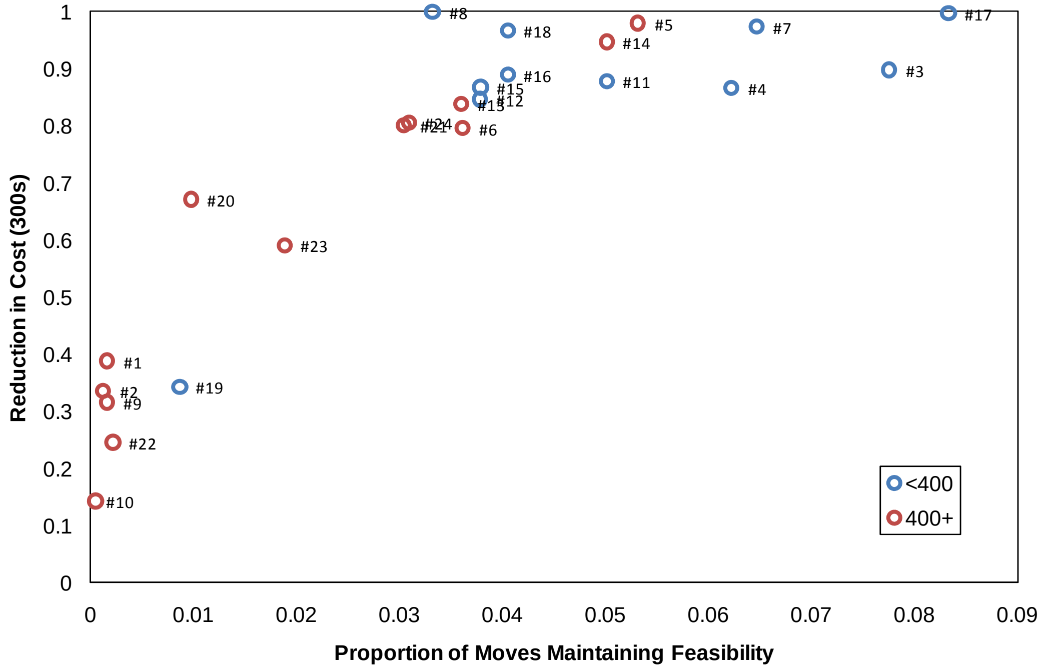


	1	2	3	4	5
1	Lesson 1	Lesson 4		Lesson 7	
2		Lesson 9	Lesson 3	Lesson 8	
3	Lesson 10	Lesson 2	Lesson 5		Lesson 6

- Based on Simulated Annealing, run for a fixed time limit (300s)
- Neighbourhood Operator
  - Choose a random lesson (vertex) ***v*** and try to move it to a new timeslot (colour) ***t***
  - **Option 1**: Inspect each empty room in timeslot ***t***, if any are suitable for ***v*** such that feasibility is maintained, perform the move; **else reject**
  - **Option 2**: As above, but also perform the maximum matching algorithm if a suitable room was not identified.
  - We can also perform “swaps” in the same manner
- Clearly option 2 allows more feasible moves to be identified, bringing higher search space connectivity
- However, repeated applications of a (polynomially-bounded) maximum matching algorithm still adds expense

Search Space Connectivity and Timetabling Problems

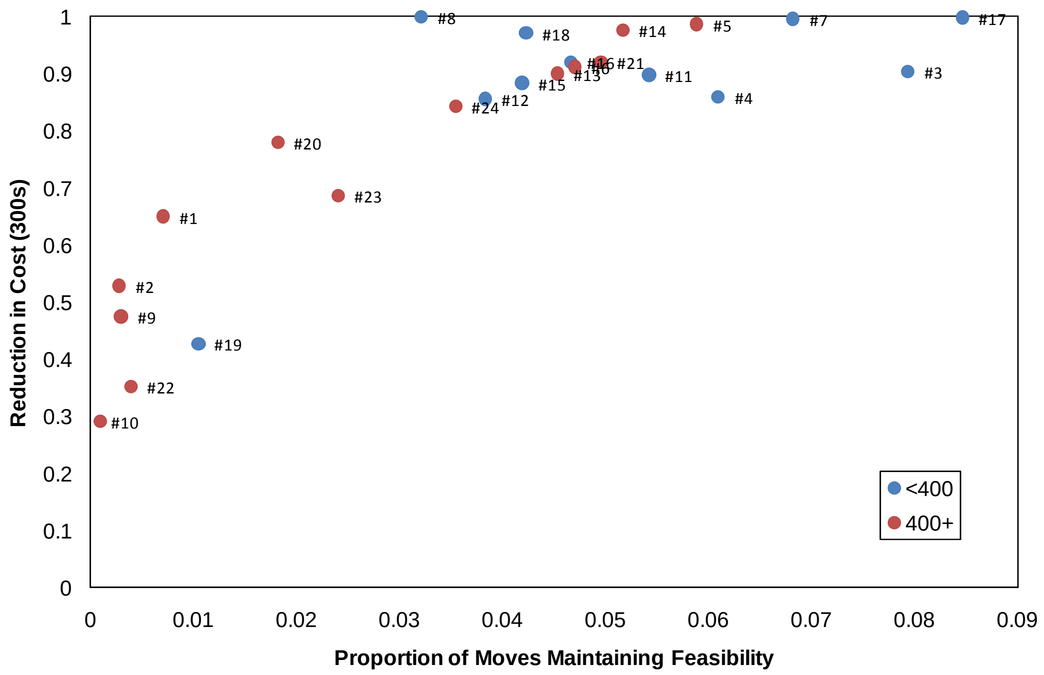
Scatter-plot comparing search-space connectivity (modelled as the proportion of moves seen to retain feasibility) vs the overall reduction in cost



- **Option 1** (no maximum matching algorithm)
  - 24 benchmark instances (of varying size and "constrained-ness" were considered"
  - Clearly there is a strong positive correlation between "connectivity" and the amount the solution is ultimately improved

Search Space Connectivity and Timetabling Problems

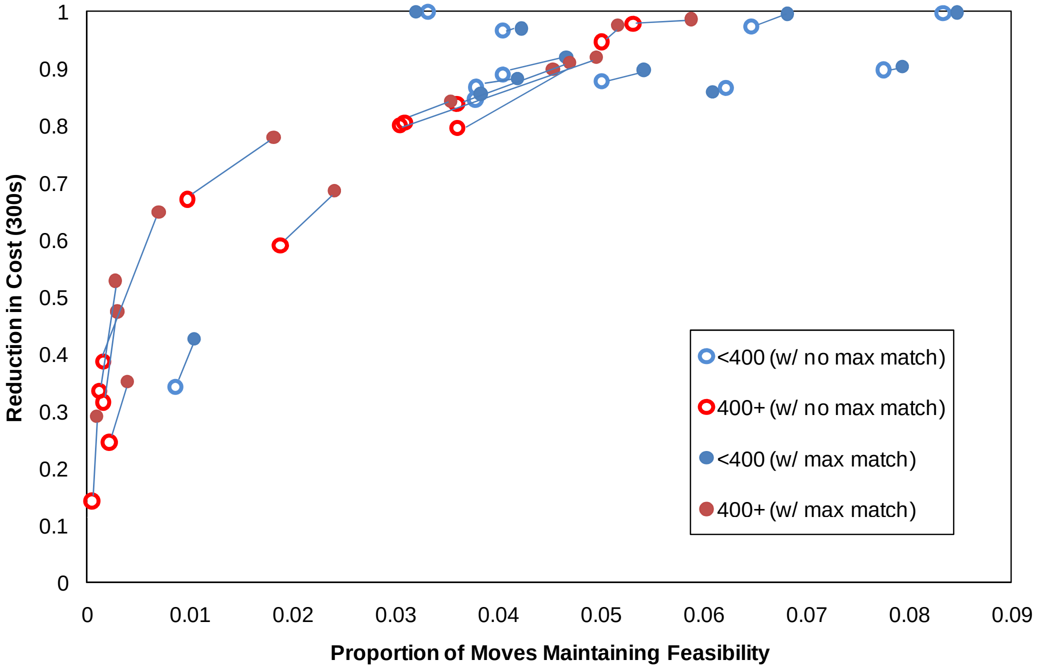
Scatter-plot comparing search-space connectivity (modelled as the proportion of moves seen to retain feasibility) vs the overall reduction in cost



- **Option 2** (w/ maximum matching algorithm)
  - 24 benchmark instances (of varying size and "constrained-ness" were considered"
  - Clearly there is a strong positive correlation between "connectivity" and the amount the solution is ultimately improved

Search Space Connectivity and Timetabling Problems

Scatter-plot comparing search-space connectivity (modelled as the proportion of moves seen to retain feasibility) vs the overall reduction in cost



- **Comparison**
  - The use of the maximum matching algorithm improves the connectivity of the search space
  - Clearly, this allows the algorithm to reduce the cost significantly more than the first option

Further Points on Timetabling and Related Problems

- Search space connectivity can be improved by exploiting underlying problem structure
  - Maximum matching algorithm
  - Kempe-chain and S-chain interchanges
  - Ejection-chain and optimal-reassignment algorithms (Hungarian, Auction algorithms)
- Are there any “useful” recombination-type operators that can also maintain feasibility in timetabling?
- Comparison of algorithms is a problem.
  - Competitions can be useful for generating new ideas without the fear of “are these ideas publishable??!!?”
  - On the other hand, are we interested in understanding algorithm behaviour, or do we just want to beat other algorithms?

References…

- **Carter’s Timetabling Instances `ftp://ftp.mie.utoronto.ca/pub/carter/testprob/all_file.zip`**
- **Dimacs Graph Colouring Instances `http://dimacs.rutgers.edu/challenges/`**
- **International Timetabling Competition `http://www.cs.qub.ac.uk/itc2007/`**
- I. Blochliger and N. Zufferey.”A graph coloring heuristic using partial solutions and a reactive tabu scheme.” *Computers and Operations Research*, 35:960--975, 2008.
- D. Brelaz. “New methods to color the vertices of a graph.” Commun. ACM, 22(4):251--256, 1979.
- P. Galinier and J-K. Hao. “Hybrid evolutionary algorithms for graph coloring.” *Journal of Combinatorial Optimization*, 3:379--397, 1999.
- A. Hertz and D. de Werra. “Using tabu search techniques for graph coloring.” *Computing*, 39(4):345--351, 1987.
- J. Moody and D. White. “Structural cohesion and embeddedness: A hierarchical concept of social groups.” *American Sociological Review*, 68(1):103--127, 2003.
- S. Korman. Combinatorial Optimization. The Graph-Colouring Problem, pages 211--235. Wiley, New York, 1979.
- R. Lewis. “A general-purpose hill-climbing method for order independent minimum grouping problems: A case study in graph colouring and bin packing.” *Computers and Operations Research*, 36(7):2295--2310, 2009.
- R. Lewis and J. Thompson. “On the application of graph colouring techniques in round-robin sports scheduling.” *Computers and Operations Research*, 38(1):190--204, 2010.
- **R. Lewis, J. Thompson, C. Mumford, and J. Gillard, “A Wide-Ranging Computational Comparison of High-Performance Graph Colouring Algorithms”, *Computers & Operations Research* DOI: 10.1016/j.cor.2011.08.010.**
- J. Thompson and K. Dowsland. “An improved ant colony optimisation heuristic for graph colouring.” *Discrete Applied Mathematics*, 156:313--324, 2008.