
Annals of Mathematics and Artificial Intelligence **34:** 327–340, 2002.

Coloration neighbourhood search with forward checking

Steven Prestwich

Department of Computer Science, National University of Ireland at Cork, Ireland
E-mail: s.prestwich@cs.ucc.ie

Two contrasting search paradigms for solving combinatorial problems are *systematic backtracking* and *local search*. The former is often effective on highly structured problems because of its ability to exploit consistency techniques, while the latter tends to scale better on very large problems. Neither approach is ideal for all problems, and a current trend in artificial intelligence is the hybridisation of search techniques. This paper describes a use of forward checking in local search: pruning coloration neighbourhoods for graph colouring. The approach is evaluated on standard benchmarks and compared with several other algorithms. Good results are obtained; in particular, one variant finds improved colourings on geometric graphs, while another is very effective on equipartite graphs. Its application to other combinatorial problems is discussed.

Keywords: graph colouring, forward checking, coloration neighbourhood

1. Introduction

Graph colouring is an NP-hard combinatorial optimisation problem with real-world applications such as timetabling, scheduling, frequency assignment, computer register allocation, printed circuit board testing and pattern matching. A graph $G = (V, E)$ consists of a set V of vertices and a set E of edges between vertices. Two vertices connected by an edge are said to be *adjacent*. The aim is to assign a colour to each vertex in such a way that no two adjacent vertices have the same colour. A *graph colouring problem* for a graph G is the problem of finding a k-colouring with k as small as possible. The chromatic number $\chi(G)$ of a graph is the minimum number of colours required to colour it.

Many algorithms have been proposed for the graph colouring problem. Systematic backtracking gives good results on small graphs but scales poorly to large problems. Most colouring algorithms are stochastic in nature, searching in a nonsystematic way with a variety of heuristics. The simplest type of stochastic search is *local search*: hill climbing, often augmented with heuristics for escaping local minima. Local search explores the *neighbourhood* of a point σ in a space by making *local moves*. The neighbourhood consists of the set of points σ' that can be reached by a single local move. The aim is to minimise (or equivalently to maximise) some *objective function* $f(\sigma)$ on the space. A local move $\sigma \rightarrow \sigma'$ can be classified as *backward*, *forward* or *sideways*, depending on whether $f(\sigma') - f(\sigma)$ is positive, negative or zero. Some algorithms choose

a forward move that yields the greatest reduction in value, a strategy sometimes called *greedy* or *steepest descent*. A drawback of local (and other stochastic) search is that it may converge on a *local minimum*: a point that has lower value than all its neighbours but is not a global minimum. The aim of backward moves is to escape from local minima by providing *noise*, while sideways moves are often used to traverse function plateaus.

We might classify colouring algorithms by the search spaces they explore. The space of *total colorations* consists of the possible colour assignments to all the vertices of a graph, using a fixed number of colours. This approach is used by the TABU algorithm, which tries to minimise the number of conflicts. A *conflict* is an adjacent pair of vertices with the same colour. TABU generates a selection of possible single-vertex reassignments and selects the best, even if this leads to more conflicts. It also maintains a list of recent moves and avoids reversing them, which helps it to escape local minima.

The *consistent total colorations* are the total colorations that contain no conflicts. This space is explored by the greedy (or *sequential*) algorithm, which tries to colour each vertex with a colour already used for a previous vertex; if this is not possible then a new colour is used. Heuristics control the vertex order and colour selection. The iterative greedy algorithm iteratively applies the greedy algorithm, using vertex orderings that are guaranteed to generate a sequence of colorations using a nonincreasing number of colours. Brélaz's DSATUR algorithm [2] explores a similar space. It orders vertices dynamically by maximum *saturation* (number of distinct colours assigned to adjacent vertices), breaking ties by choosing a vertex of greatest forward degree. The *degree* of a vertex is the number of its adjacent vertices, and its *forward degree* is the number of its *uncoloured* adjacent vertices. DSATUR has also been extended by backtracking. Mehrotra and Trick's version [15] uses full (exhaustive) search and begins by computing a clique which is never recoloured. Another version described by Culberson, Beacham and Papp [3] uses limited backtracking.

The space of *consistent partial colorations* consists of consistent colorations of subsets of the vertices that do not use more than a specified number of colours. This space is explored by the IMPASSE colouring algorithms. The objective function to be minimised is the sum of the forward degrees of the uncoloured vertices. Two such algorithms are Morgenstern's distributed IMPASSE [17] and Lewandowski and Condon's parallel IMPASSE [14]. Distributed IMPASSE performs limited searches on a distributed architecture, each search starting from previous good coloration, which are maintained in a pool. Parallel IMPASSE is a hybrid of IMPASSE and systematic search; the two execute in parallel and communicate colouring improvements to each other.

Finally, the *independent sets* of a graph are the sets of pairwise nonadjacent vertices. They exploit the fact that all the vertices of an independent set can be assigned the same colour. The space of independent sets is explored by algorithms such as XRLF of Johnson et al. [10] and MAXIS of Culberson and Luo [4] using (exhaustive or restricted) backtracking and iteration. LPCOLOR of Mehrotra and Trick [15] uses column generation and branch-and-bound to explore this space.

The motivation for this classification is that we shall describe a local search algorithm that explores a new space. This is a subspace of the consistent partial colorations,

reduced by applying the constraint programming technique of *forward checking*. We describe the new algorithm in section 2 and evaluate its performance in section 3. The algorithm, results and related work are discussed in section 4. This paper is an extension of earlier work [19], but the algorithm is described in greater detail and evaluated more systematically.

2. FC-consistent partial coloration neighbourhood search

We first discuss a simple consistency technique from constraint programming: *forward checking* (FC). FC is commonly used with systematic backtracking [9], and this combination can be applied to graph colouring as follows. Each vertex has an associated *domain* of possible colours, initialised to the full set of available colours. On colouring a vertex with one of the colours in its domain, that colour is deleted from the domains of the adjacent vertices. No colour assignment is permitted if it causes the domain of some uncoloured vertex to become empty. This *domain wipe-out* often occurs long before the vertex in question is due to be coloured, greatly reducing the search space. On backtracking, a vertex is uncoloured and its assigned colour restored to the domains of adjacent vertices. An analogue to the Brélaz heuristic may be used to select vertices for colouring: select a vertex with smallest domain, breaking ties by selecting a vertex with greatest forward degree.

FC is a simple and inexpensive algorithm, and sometimes out-performs theoretically more powerful techniques. FC with backtracking also has the advantage of *completeness*. That is, all k-colourings will eventually be found, and if there is no k-colouring then the algorithm will eventually prove this by terminating without finding a solution. However, systematic backtracking algorithms often suffer from poor scalability. For example, the FC algorithm is effective on small N-queens problems, but cannot solve problems with more than approximately 100 queens; in contrast, a local search algorithm solves up to 10^6 queens in linear time [16]. Backtracking and local search are complementary search techniques for solving colouring and other combinatorial problems, and considerable research has been devoted to combining advantages of the two.

We describe how to exploit FC within a local search algorithm for graph colouring. The idea is to explore the subspace of the consistent partial colorations that are also consistent under forward checking. That is, for each of the currently uncoloured vertices, there is at least one available colour that can be consistently assigned to it; colorations causing vertex domain wipe-out are avoided. We shall call this subspace the *FC-consistent partial colorations*. Our reasoning is that by reducing the search space we may avoid some local minima.

Before describing the particular algorithm used to explore this space, we discuss a complication that arises when applying local search to it. In systematic backtracking it is simple to maintain vertex domains: the order in which colours are restored from domains on backtracking is the reverse of the order in which they were deleted during assignment. It is sufficient to maintain a boolean variable for each colour in each domain,

denoting whether or not the colour is currently in the domain. However, local search is nonsystematic, and from a given coloration we may wish to uncolour *any* vertex, not just the most recently coloured one. To do this we need a new way of maintaining domains. A number we shall call a *conflict count* is maintained for each vertex–colour pair (v, c) recording how many currently coloured vertices the assignment $v = c$ would conflict with; initially all conflict counts are zero. A colour is classed as *deleted* from a vertex domain if and only if its conflict count is greater than zero. A domain size is the number of its nondeleted colours. The memory requirement for conflict counts is not excessive: for n vertices and k colours $k \times n$ conflict counts are needed, which is roughly the amount of memory required to represent the problem. They may be updated incrementally: on colouring/uncolouring a vertex, the conflict count for that colour in each adjacent vertex is incremented/decremented. However, they cause a significant runtime overhead compared to standard forward checking, because they are updated for the domains of uncoloured *and coloured* vertices.

We can now design a local search algorithm on FC-consistent partial colorations. The algorithm chosen is rather simple, starting exactly as standard FC: it selects a vertex for colouring, finds a colour that can be used without causing domain wipe-out, colours the vertex, updates the domains of adjacent vertices, and repeats by selecting another vertex. The only difference so far is that domains are maintained by conflict counts. However, on reaching a *dead-end*, where the selected vertex cannot be coloured, the new algorithm behaves differently to standard FC. It *heuristically* selects a vertex to be uncoloured, instead of selecting the most recently coloured vertex. No attempt is made to backtrack systematically, so completeness is lost. Because there is now no obvious criterion for deciding when to stop backtracking and start colouring vertices again, we introduce a parameter $B \geqslant 1$. On reaching a dead-end B vertices are uncoloured, and colouring resumes. Note that the vertices selected for colouring and uncolouring may follow different heuristics, so that the set of coloured vertices may change rapidly during search. B plays the part of a *noise* parameter (or the temperature in simulated annealing), controlling the permitted disruption to the state on reaching a local minimum.

It remains to fill in details by describing three heuristics: selecting B coloured vertices for uncolouring (CVERTEX), selecting an uncoloured vertex for colouring (UVERTEX), and selecting colours to try when colouring a vertex (COLOUR). We consider two alternative UVERTEX rules:

- *Brélaz*: select the vertex with smallest current domain; break ties by selecting the vertex adjacent to the greatest number of uncoloured vertices; break further ties randomly.
- *Nonsingleton*: randomly select a vertex with more than one colour in its current domain; if none exists then select a vertex randomly.

The Brélaz heuristic is an obvious choice. The idea behind the nonsingleton heuristic is to emulate the MAXIS algorithm, which constructs independent sets of vertices, whereas DSATUR constructs cliques. By selecting vertices using an inverse of the Brélaz heuristic, and thus focusing on vertices that are as independent as possible from those currently

coloured, we might expect to obtain a forward-checking analogue to MAXIS. This was tested and, perhaps surprisingly, performed rather well on random graphs, whereas the Brélaz heuristic performed poorly. However, the weaker nonsingleton heuristic performs better, possibly because of its greater flexibility in selecting a vertex. It is discussed further in section 4.

Given a free choice of vertices for uncolouring, which should be selected? An obvious idea is to use an inverse of Brélaz: uncolour a vertex with large domain and small degree (note that because conflict counts are updated irrespective of whether a vertex is coloured, coloured vertices also have domain sizes). In tests this often caused stagnation, but the weaker nonsingleton heuristic (applied to coloured vertices) works well. To further reduce stagnation, with probability $1/n$ (where n is the number of vertices in the graph) the CVERTEX rule selects a vertex randomly instead of by domain size.

A random ordering on domain values works well, but performance can be improved by remembering the previous colour of each vertex (if it was coloured earlier). The COLOUR rule flips between two modes: initially it prefers different colours to those remembered for each vertex; if a different colour is successfully used, the rule flips to preferring the remembered colour; when CVERTEX is next invoked it flips back to preferring a different colour. The aim of this rule is to minimise disruption to colorations as the set of coloured vertices changes, while avoiding null local moves.

The new algorithm FCNS (FC-consistent partial coloration Neighbourhood Search) is shown in figure 1. $k \geqslant 1$ is the permitted number of colours and $B \geqslant 1$ is the noise parameter. C is the current set of coloured vertices, initialised to {}. U is the current set of uncoloured vertices, initialised to the full set of n vertices $\{v_1, \ldots, v_n\}$. Each ver-

```
Function FCNS(B,k)
  let C = {} and U = {v_1, ..., v_n}
  for i = 1 to n let domain(v_i) = {1, ..., k}
  while U ≠ {}
    let u = UVERTEX(U)
    let D = {d ∈ domain(u) | colouring u to d does not cause domain wipe-out}
    if D = {}
      for i = 1 to min(B, |C|)
        let c = CVERTEX(C)
        uncolour c and update domains
        let C = C − {c} and U = U ∪ {c}
    else
      colour u to COLOUR(D) and update domains
      let C = C ∪ {u} and U = U − {u}
  return coloration
```

Figure 1. FC partial coloration neighbourhood search for fixed k.

tex has a *domain* of colours that are FC-consistent with the current partial coloration, initialised to the full set of colours $\{1, \ldots, k\}$. The algorithm proceeds by selecting uncoloured vertices using the UVERTEX rule, and colours them using the COLOUR rule. On reaching a dead-end ($D = \{\}$) it uncolours B vertices, each selected by the CVERTEX rule. Termination is not guaranteed but occurs if all vertices are coloured ($U = \{\}$ and $C = \{v_1, \ldots, v_n\}$).

The algorithm can be used to find a near-optimal colouring by applying it iteratively in an obvious way: start with large k (for example, $k = n$) and apply the algorithm; on finding a total coloration using $k' \leqslant k$ colours, restart the algorithm with $k' - 1$ colours; repeat until reaching a target number of colours or a specified time. Performance is improved by starting each iteration with a coloration similar to the previous one: colour assignments are stored between iterations, and until the first dead-end occurs each vertex is assigned its previous colour where possible. It is also possible to speculatively reduce k further in the hope of finding better colourings more quickly. However, this *aspiration* approach does not always speed up search, because inadvertently choosing k less than the chromatic number of the graph runs the risk of spending a long time in fruitless search. Aspiration is not used in current FCNS implementations.

3. Experimental results

FCNS is now evaluated using published results for several other algorithms on the well-known DIMACS [11] [1] benchmarks. They are: Iterated Greedy (IG) of Culberson and Luo [4], distributed IMPASSE of Morgenstern [17], parallel IMPASSE of Lewandowski and Condon [14], Squeaky Wheel Optimization (SWO) of Joslin and Clements [12] and TABU of Glover, Parker and Ryan [8]. The TABU algorithm combines the TABU meta-heuristic with branch-and-bound. SWO operates in two search spaces: a solution space and a prioritisation space. Both searches influence each other: each solution is analysed and used to change the prioritisation, which guides the search strategy used to find the next solution, found by restarting the search.

We use a standard set of benchmarks taken from the DIMACS web site. *Geometric graphs* R$x.y$ and DSJR$x.y$ are generated by randomly placing x vertices in a unit square, then assigning edges between any two vertices with euclidean distance less than $y/10$ between them; a graph denoted by Gc is the complement of the graph G. The names R and DSJR reflect different sources, but are (we believe) the same type of graph. *Random graphs* C$n.p$ and DSJC$n.p$ have n vertices, an edge being assigned between any two vertices with a fixed probability $p/10$. The names C and DSJC again reflect different sources. *Flat graphs* contain colorations that are hidden in such a way as to mislead Brélaz-style heuristics; a graph flatn_c_x contains n vertices and a known hidden (though not necessarily optimal) c-colouring. *Leighton graphs* le450_15x ($x \in \{a, b, c, d\}$) are derived from scheduling, and have 450 edges and known chromatic number 15. Graph colouring is closely related to the timetabling problem and there are

[1] ftp://dimacs.rutgers.edu/pub/challenge/

two *timetabling graphs*; the school1 problem is derived from timetabling data from a real high school with around 500 students; the school1_nsh problem is derived from the same data but ignores study halls. *Register allocation graphs* are used in compilers to assign variables to registers, with the aim of avoiding the placement of two variables in the same register when both may be active; there is one such graph, mulsol.i.1. The *latin square graph* latin_sqr_10 is derived from a standard problem in design theory.

Table 1 reproduces published results for SWO, IG, d-IMP (distributed IMPASSE), p-IMP (parallel IMPASSE) and TABU, and table 2 shows results for FCNS with the Brélaz (FCNS-b) and Nonsingleton (fCNS-n) heuristics. All times are normalised to our machine (a 300 MHz DEC Alphaserver 1000A 5/300 under Unix) using benchmark tim-

Table 1
Previous results for DIMACS benchmarks.

Problem	SWO		IG		d-IMP		p-IMP		TABU	
	k	t	k	t	k	t	k	t	k	t
R125.1	5	0.1	5	1.1	5	<0.24	5	64.6	5	0.2
R125.1c	46	2.7	46	0.6	46	<0.24	46	85	46	0.5
R125.5	36	1.5	36.9	1	36	<0.24	37	33	36	0.4
R250.1	8	0.3	8	3.8	8	<0.24	8	22	8	0.1
R250.1c	64	16.4	64	2.5	64	0.24	64	278	65	24.9
R250.5	65	7.9	68.4	4.5	65	44.1	66	39.9	66	31.7
R1000.1	20	4.3	20.6	46.9	20	4.4	20	49.9	20	0.9
R1000.1c	102	233	98.8	26.4	98	303	103	3940	105	1849
R1000.5	239	309	253	55.3	241	507	246	216	248	1849
DSJR500.1	12	1.1	12	11.3	12	<0.24	12	26.6	12	0.3
DSJR500.1c	85.2	52	85	7.8	85	31.7	85.2	5768	87	1849
DSJR500.5	124	36.9	130	14	123	94.1	128	90.5	126	212
school1	14	4.5	14	5.6	14	<0.24	14	46.3	14	48.7
school1_nsh	14	3.9	14.1	4.8	14	<0.24	14	66.4	26	16.8
mulsol.i.1	49	3.2	49	2.3	49	<0.24	49	27.2	49	0.2
le450_15a	15	3	17.9	9.1	15	<0.24	15	163	16	9.6
le450_15b	15	3.3	17.9	8.7	15	<0.24	15	178.4	15	15.3
le450_15c	21.1	4.3	25.6	7.8	15	30.7	16.6	2230	23	1849
le450_15d	21.2	4.2	25.8	7.2	15	19.5	16.8	2860	23	1849
DSJC125.5	18.3	0.9	18.9	1.3	17	3.4	17	4044	20	82.3
DSJC250.5	31.9	4.5	32.8	3.7	28	144	29.2	4358	35	1849
DSJC500.5	56.3	22	58.6	9.8	49	4355	53	4784	65	1849
DSJC1000.5	102	112	104	36.3	89	22279	100	5334	117	1849
C2000.5	186	562	190	146	165	7571				
C4000.5	342	2659	347	566						
flat300_20_0	25.3	8.8	20.2	2	20	<0.24	20	274	39	1849
flat300_26_0	35.8	6.4	37.1	4.1	26	5.4	32.4	6637	41	1849
flat300_28_0	35.7	6.4	37	5.2	31	1028	33	1914	41	1849
flat1000_50_0	100	110	65.6	78.6	50	<0.24	97	7792		
flat1000_60_0	101	106	103	46.9	60	<0.24	97.8	6288		
flat1000_76_0	101	112	104	42.7	89	5925	99	6498		
latin_sqr_10	112	198	107	32.1	98	2738	109	6520	130	1849

Table 2
FNCS results for DIMACS benchmarks.

Problem	FCNS-b			FCNS-n		
	B	*k*	*t*	*B*	*k*	*t*
R125.1	1	5	0.003	1	5	0.006
R125.1c	1	46	0.026	1	46	0.611
R125.5	1	36	0.109	1	37	5.18
R250.1	1	8	0.001	1	8	0.034
R250.1c	2	64	0.162	1	64	5.18
R250.5	4	65	0.166	1	68	65.0
R1000.1	1	20	0.341	1	20	52.7
R1000.1c	1	98	14.9	1	114	239
R1000.5	7	234	475	1	263	403
DSJR500.1	1	12	0.063	1	12	0.661
DSJR500.1c	1	85	1.107	1	89	49.9
DSJR500.5	5	122	1.62	1	133	27.0
school1	1	14	0.068	1	17	8.26
school1_nsh	1	14	0.489	1	17	4.53
mulsol.i.1	1	49	0.016	1	49	0.248
le450_15a	2	15	2.39	1	17	9.59
le450_15b	2	15	0.974	1	17	8.65
le450_15c	1	21	17.9	1	22	91.0
le450_15d	1	21	24.2	1	22	44.7
DSJC125.5	1	18	14.7	2	18	32.0
DSJC250.5	1	32	33.9	2	32	53.1
DSJC500.5	1	59	2.88	1	54	176
DSJC1000.5	1	107	15.4	1	97	748
C2000.5	1	192	606	1	175	8576
C4000.5	1	357	573	1	325	48114
flat300_20_0	1	35	4.13	2	20	16.3
flat300_26_0	1	37	1.32	1	35	18.3
flat300_28_0	1	37	1.23	1	35	20.1
flat1000_50_0	1	105	11.7	1	95	582
flat1000_60_0	1	106	7.81	1	97	303
flat1000_76_0	1	105	49.0	1	98	312
latin_sqr_10	1	117	49.2	1	106	2463

ings from [11]; the DIMACS benchmark program `dfmax r500.5` takes 46.2 seconds on our machine. The times for parallel IMPASSE were not normalised because of its parallel platform (a 32-processor CM-5). In both tables k is the number of colours used and t is the time taken in seconds. In table 2, B is the value used for the B parameter. Its value was chosen after a few runs to find an appropriate setting. This ad hoc approach is unfortunately necessary with many local search algorithms; TABU has a list length parameter, and some algorithms have more than one parameter (for example, several local search algorithms for the satisfiability problem). The initial number of colours k_0 for FCNS was set to the worst k found by the other algorithms in each case (except where our algorithms were even worse, when higher values were used). FCNS was halted on

reaching the target k, which was selected after a few experimental runs. Times shown for FCNS are mean times taken to reach k from k_0, averaged over 10 runs (more for short times). Experimental details for the other algorithms vary (for details see the cited papers). Briefly, SWO was terminated after 1000 iterations, IG after 1000 iterations without improvement, TABU after an hour or sooner if a lack-of-progress condition was satisfied, distributed IMPASSE used conditions depending on the problem but always halted on reaching a specified target k, and parallel IMPASSE ran for 3 hours then reported the time taken to find the best solution. The use of a time limit instead of a target number of colours explains the occasional fractional values of k.

First we discuss FCNS-b, which is clearly the best algorithm on the geometric graphs. On R1000.5 and DSJR500.5, it finds the best reported colourings, and on most of the others it finds equally good colourings in shorter times. The geometric graphs are randomly generated but closely related to a real problem: frequency allocation [10]. FCNS-b is therefore a promising algorithm for solving such problems, and this is an area for future research. It also performs very well on the school and mulsol graphs, roughly matching the performance of distributed IMPASSE. We also tested Mehrotra's and Trick's DSATUR implementation[2] on the geometric graphs because it is known to perform well on such graphs. On those with edge probability 0.1 it found the same colourings in a slightly shorted time than FCNS-b. On those with edge probability 0.5 it quickly found good colourings but then made no further progress for a long time. For R125.5 it found a 36-colouring in 63.4 seconds, for R250.5 a 66-colouring in 2.9 seconds, for DSJR500.5 a 130-colouring in 17.6 seconds, and for R1000.5 a 246-colouring in 75.8 seconds; no further progress was made after several minutes. FCNS-b clearly scales better than DSATUR, finding better colourings on the larger graphs. However, it is very poor on the random, flat and latin square graphs, and mediocre on the Leighton graphs.

Next we discuss FCNS-n. On the geometric and school graphs it is poor, sometimes the worst algorithm, and (like FCNS-b) mediocre on the Leighton graphs, but on the random, flat and latin square graphs it is beaten only by distributed IMPASSE. This is presumably due to the use by distributed IMPASSE of the XRLF algorithm [10] to generate initial colorations: parallel IMPASSE does not use XRLF and is beaten by FCNS-n on random and flat graphs. However, other algorithms are also better than FCNS-n on random graphs. For example, on $\mathcal{G}_{1000,0.5}$ graphs the best algorithms find colourings in the low- or mid-80s.

To further investigate FCNS-n we applied it to *equipartite* graphs, which have been studied by several researchers. A k-colourable equipartite graph is generated by partitioning its vertices into k subsets, which are as equally-sized as possible, the smallest being no more than 1 vertex smaller than the largest. Edges are assigned with probability p, disallowing edges between vertices in the same subset. This guarantees a k-colouring but does not preclude better colourings. Eiben, van der Hauw and van Hemert [5] apply evolutionary algorithms to 3-colourable equipartite graphs with 200 vertices. They

[2] http://mat.gsia.cmu.edu/COLOR/color.html.

report low success rates on graphs with low density, especially around $p = 0.05$ where a phase transition occurs. Minton et al. [16] also report that the Min-conflicts local search algorithm has difficulties with similar problems, but that a backtracking version of DSATUR solves them easily for $3 \leqslant n \leqslant 180$. Yugami et al. [27] apply local search with constraint propagation to the same problems and obtain improved results. FCNS-n solves these problems easily: with $B = 2$ it finds 3-colourings in approximately 3 seconds. Moving to larger problems, Culberson et al. [4] show that IG can find hidden k-colourings for $\mathcal{G}_{1000,0.5}$ equipartite graphs with $k \leqslant 60$. FCNS-n is also able to do this and can go a little further. The algorithm was quite insensitive to B until $k \approx 55$, after which it became more sensitive. The optimal value for $k = 60$ was approximately $B = 65$. For $k > 60$, the problems rapidly became harder and the optimal value of B fell. It found a hidden 67-colouring after several hours computation with $B = 20$ but failed to find a hidden 68-colouring. So far as we known, 67 is the highest value of k solved for this class of graph. In further experiments, FCNS-n also managed to find the hidden colourings in flat1000_{50,60}_0, by setting $B = 60$ and starting with the target colouring (50 or 60) specified as the initial colouring. However, these results took much trial and error to achieve, so they were not included in our table.

It is perhaps surprising that the nonsingleton heuristic should be successful at all, let alone competitive. In particular, if a vertex has domain size 1 (hence only one possible colour) then Brélaz will select it before a vertex with larger domain (hence more than one possibility), but nonsingleton will delay colouring such vertices as long as possible. To investigate the effect of adding "forced moves" another variant was tried: select a vertex with domain size 1 if one exists, otherwise select one with maximum domain size. However, this variant was inferior to both Brélaz and Nonsingleton. We speculate that Nonsingleton causes FCNS to behave in a similar way to independent set-based algorithms such as MAXIS, by focusing search on low-degree vertices. A better algorithm might be obtained by explicitly designing it to find independent sets, and applying forward checking. Another research direction is the design of new vertex orderings, with the aim of improving FCNS performance on random, Leighton and latin square graphs.

The noise parameter B unfortunately requires tuning to each graph. As with any noise parameter, the effect of setting B too low is stagnation: FCNS will never find a colouring because it becomes trapped in a local minimum. The effect of setting B too high is less serious, simply increasing the time taken to find a solution, but the increase depends on the problem. The performance of FCNS-n seems to be fairly insensitive to the value of B when finding an 18-colouring for DSJC125.5, while on DSJC1000.5 increasing B from 1 to 2 slows it down greatly – or equivalently – prevents it from finding good colourings in the same time. FCNS-b seems less sensitive, but can still be slowed down significantly by too much noise. We experimented with variable noise levels to try to reduce sensitivity to noise, but with inconclusive results. A slightly surprising feature of B is that, on several graphs (for example, R250.5), best results were obtained for FCNS-b and FCNS-n using different values of B. However, perhaps this should not be surprising, because the two algorithms focus on different regions of the graphs and therefore might be expected to encounter local minima of different depths.

4. Discussion

The main difference between FCNS and other stochastic colouring algorithms is that it performs forward checking. It also has an additional advantage over IG and SWO: *incrementality*. IG and SWO are not incremental because restarting is an expensive move, whereas IMPASSE, TABU and FCNS make small, cheap moves in the search space. This is pointed out as a source of inefficiency by Joslin and Clements [12], and they propose hybrids of SWO with local search for future work.

Graph colouring is a binary constraint satisfaction problem (given fixed k), and the use of conflict counts to perform forward checking in local search is easily generalised to other such problems. It can be further generalised to non-binary constraint problems, and this has been done for propositional satisfiability (SAT). Experimental results are very promising: on some large, structured SAT problems it out-performs current systematic and local search algorithms [20]. In fact our colouring and SAT algorithms are instances of a general-purpose approach to combinatorial optimisation and constraint satisfaction, which we call Constrained Local Search (CLS). The aim of CLS is to enhance local search with constraint programming techniques used in systematic search. It has also given good results on other SAT problems [20], maximum clique problems [22], Golomb rulers [22] and a hard optimisation problem (the generation of low-autocorrelation binary sequences) [21]. The general approach is to take an effective backtracking algorithm and replace systematic by randomised backtracking, usually improving its scalability at the expense of completeness.

It might be argued that FCNS (or more generally CLS) is not a local search algorithm, but simply a randomised backtracker. It certainly is a randomised backtracker and has much in common with Dynamic Backtracking (DB), which also allows the removal of early assignments without affecting the assignments made since. This increased flexibility of backtracking was a stated aim in the design of DB, and a later hybrid algorithm called Partial Order Dynamic Backtracking [7] achieved even greater flexibility. Is FCNS simply an inferior version of DB, sacrificing completeness to no good purpose? A counter-example to this view is the random 3-SAT problem, on which DB is slower than depth-first search [1] while CLS scales precisely as local search [21]. Our view is that FCNS stochastically explores a space of FC-consistent partial colorations by local search; the objective function it minimises is the number of uncoloured vertices. However, to some extent the question is academic: even if FCNS is not local search, experimental results show that it captures its essence, successfully solving problems that are beyond the range of systematic backtracking.

There are several other hybrids of local search and constraint techniques. The simplest hybrid is a parallel or distributed implementation of more than one algorithm, as in the IMPASSE algorithms used for colouring. Schaerf's timetabling algorithm [24], extended to constraint satisfaction problems, searches the space of *all* partial assignments (not only the consistent ones) using an objective function that includes a measure of constraint violation. This is a different space again than those searched by current colouring algorithms and FCNS. In graph colouring terms, this space may be called

the *partial colorations* as opposed to the *consistent* partial colorations explored by IMPASSE. Jussien's and Lhomme's path-repair algorithm [13] is described as a generalisation of Schaerf's approach that includes learning (allowing complete versions to be devised) and a TABU list. EFLOP algorithm of Yugami, Ohta and Hara [27] uses constraint propagation to escape local minima, while allowing some constraint violation. However, forward checking is not maintained throughout the search, as it is in FCNS. Partial-order dynamic backtracking of Ginsberg and McAllester [7] is a hybrid of the dynamic backtracking algorithm with a local search algorithm [6], enabling it to follow local gradients in the search space. Pesant and Gendreau [18] apply systematic branch-and-bound search to efficiently explore local search neighbourhoods. The two-phase algorithm of Zhang and Zhang [28] searches a space of partial variable assignments, alternating backtracking search with stochastic local search on the same data structure. It can be tuned to different problems by spending more time in either phase. Yokoo's Weak Commitment Search [26] (WCS) greedily constructs consistent partial assignments. On reaching a dead-end it randomly restarts, and uses learning to maintain completeness. Richards and Richards [23] describe a SAT algorithm called learn-SAT based on WCS. Shaw [25] describes a vehicle routing algorithm called Large Neighbourhood Search. It performs local search and uses backtracking with constraint propagation to test the legality of moves.

Each of these algorithms either permits constraint violation or uses learning, or both. Constraint violation implies, in the view of this author, that constraints are being under-used. This may be a drawback when solving highly structured problems: the best graph colouring results for structured problems are obtained by algorithms such as IMPASSE and FCNS, which do not violate constraints. The use of learning is a drawback when solving large problems. It can be restricted to use only polynomial memory, but combinatorial problems may be very large. The FCNS approach combines constraint handling and local search, making cheap local moves and avoiding memory-intensive learning techniques. We believe that this combination of features makes it ideal for large, highly constrained problems.

References

[1] A.B. Baker, The hazards of fancy backtracking, in: *Proceedings of the Twelfth National Conference on Artificial Intelligence*, Vol. 1 (AAAI Press, 1994) pp. 288–293.

[2] D. Brélaz, New methods to color the vertices of a graph, Communications of the ACM 22(4) (1979) 251–256.

[3] J.C. Culberson, A. Beacham and D. Papp, Hiding our colors, in: *Proceedings of the CP'95 Workshop on Studying and Solving Really Hard Problems*, Cassis, France (1995).

[4] J.C. Culberson and F. Luo, Exploring the k-colorable landscape with iterated greedy, in: [11], pp. 245–284.

[5] A.E. Eiben, J.K. van der Hauw and J.I. van Hemert, Graph coloring with adaptive evolutionary algorithms, Journal of Heuristics 4(1) (1998) 25–46.

[6] M.L. Ginsberg, Dynamic backtracking, Journal of Artificial Intelligence Research 1 (1993) 25–46.

[7] M.L. Ginsberg and D.A. McAllester, GSAT and dynamic backtracking, in: *Proceedings of the Fourth International Conference on Principles of Knowledge Representation and Reasoning* (Morgan Kaufmann, 1994) pp. 226–237.
[8] F. Glover, M. Parker and J. Ryan, Coloring by tabu branch and bound, in: [11], pp. 285–307.
[9] R.M. Haralick and G.L. Elliot, Increasing tree search efficiency for constraint satisfaction problems, Artificial Intelligence 14 (1980) 268–277.
[10] D.S. Johnson, C.R. Aragon, L.A. McGeoch and C. Schevon, Optimization by simulated annealing: an experimental evaluation; part II, graph coloring and number partitioning, Operations Research 3 (1991) 378–406.
[11] D.S. Johnson and M.A. Trick (eds.), *Cliques, Coloring and Satisfiability: Second DIMACS Implementation Challenge*, DIMACS Series in Discrete Mathematics and Theoretical Computer Science, Vol. 26 (American Mathematical Society, 1996).
[12] D.E. Joslin and D.P. Clements, Squeaky wheel optimization, Journal of Artificial Intelligence Research 10 (1999) 353–373.
[13] N. Jussien and O. Lhomme, The path-repair algorithm, in: *Proceedings of the Workshop on Large Scale Combinatorial Optimization and Constraints*, Electronic Notes in Discrete Mathematics 4 (1999).
[14] G. Lewandowski and A. Condon, Experiments with parallel graph coloring heuristics and applications of graph coloring, in: [11], pp. 309–334.
[15] A. Mehrotra and M.A. Trick, A column generation approach to graph colouring, INFORMS Journal on Computing 8 (1996) 344–354.
[16] S. Minton, M.D. Johnston, A.B. Philips and P. Laird, Minimizing conflicts: a heuristic repair method for constraint satisfaction and scheduling problems, Artificial Intelligence 58(1–3) (1992) 161–205.
[17] C. Morgenstern, Distributed coloration neighborhood search, in: [11], pp. 335–357.
[18] G. Pesant and M. Gendreau, A view of local search in constraint programming, in: *Proceedings of the Second International Conference on Principles and Practice of Constraint Programming*, Lecture Notes in Computer Science, Vol. 1118 (Springer-Verlag, 1996) pp. 353–366.
[19] S.D. Prestwich, Using an incomplete version of dynamic backtracking for graph colouring, in: *Proceedings of the Workshop on Large Scale Combinatorial Optimization*, Electronic Notes in Discrete Mathematics 1 (1998).
[20] S.D. Prestwich, Stochastic local search in constrained spaces, in: *Proceedings of the Practical Applications of Constraint Technology and Logic Programming, Practical Applications Company* (2000) pp. 27–39.
[21] S.D. Prestwich, A hybrid search architecture applied to hard random 3-SAT and low-autocorrelation binary sequences, in: *Proceedings of the Sixth International Conference on Principles and Practice of Constraint Programming*, Lecture Notes in Computer Science, Vol. 1894 (Springer-Verlag, 2000) pp. 337–352.
[22] S.D. Prestwich, Trading completeness for scalability: hybrid search for cliques and rulers, in: *Proceedings of the Third International Workshop on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, Ashford, Kent, England (2001) pp. 159–174.
[23] E.T. Richards and B. Richards, Non-systematic search and learning: an empirical study, in: *Proceedings of the Fourth International Conference Principles and Practice of Constraint Programming*, Lecture Notes in Computer Science, Vol. 1520 (Springer-Verlag, 1998) pp. 370–384.
[24] A. Schaerf, Combining local search and look-ahead for scheduling and constraint satisfaction problems, in: *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence* (Morgan Kaufmann, 1997) pp. 1254–1259.
[25] P. Shaw, Using constraint programming and local search methods to solve vehicle routing problems, in: *Proceedings of the Fourth International Conference on Principles and Practice of Constraint Programming*, Lecture Notes in Computer Science, Vol. 1520 (Springer-Verlag, 1998) pp. 417–431.

[26] M. Yokoo, Weak-commitment search for solving constraint satisfaction problems, in: *Proceedings of the Twelfth National Conference on Artificial Intelligence* (AAAI Press, 1994) pp. 313–318.

[27] N. Yugami, Y. Ohta and H. Hara, Improving repair-based constraint satisfaction methods by value propagation, in: *Proceedings of the Twelfth National Conference on Artificial Intelligence*, Vol. 1 (AAAI Press, 1994) pp. 344–349.

[28] J. Zhang and H. Zhang, Combining local search and backtracking techniques for constraint satisfaction, in: *Proceedings of the Thirteenth National Conference on Artificial Intelligence and Eighth Conference on Innovative Applications of Artificial Intelligence* (AAAI Press / MIT Press, 1996) pp. 369–374.