A Multilevel Approach to the Graph Colouring Problem

Chris Walshaw

Computing and Mathematical Sciences, University of Greenwich,
Park Row, Greenwich, London, SE10 9LS, UK.
Email: C.Walshaw@gre.ac.uk; URL: www.gre.ac.uk/~c.walshaw

Mathematics Research Report 01/IM/69

May 9, 2001

Abstract

We motivate, derive and implement a multilevel approach to the graph colouring problem. The resulting algorithm progressively coarsens the problem, initialises a colouring and then employs either Culberson's iterated greedy algorithm or tabu search to refine the solution on each of the coarsened problems in reverse order. Tests on a large suite of problem instances indicate that for low-density graphs (up to around 30% edge density) the multilevel paradigm can either speed up (iterated greedy) or even improve (tabu search) the asymptotic convergence. This augments existing evidence that, although the multilevel framework cannot be considered as a panacea for combinatorial optimisation problems, it can provide a useful addition to the combinatorial optimisation toolkit.

Keywords: Multilevel Refinement; Graph Colouring; Combinatorial Optimisation.

1 Introduction

The Graph Colouring Problem (GCP) can be simply stated as follows: given a graph $G(V, E)$, assign a colour to each vertex in V such that no two adjacent vertices have the same colour and so that the number of colours are minimised. If found, the minimum possible number of colours is known as the chromatic number of the graph G and denoted $\chi(G)$. The GCP is well studied and has many applications including scheduling, timetabling and the solution of sparse linear systems, see e.g. [4, 9, 23, 24]. However it is also known to be one of the most difficult combinatorial optimisation problems, e.g. [18]. Not only is the problem of finding $\chi(G)$ NP-hard, [11], but Lund & Yannakakis have even shown that, for some $\epsilon > 0$, approximate graph colouring within a factor of N^ϵ is also NP-hard, [25].

In this paper we consider a multilevel approach to colouring and look at its potential to aid the search for good colourings in reasonable time. The multilevel paradigm is a simple one, which at its most basic involves recursive coarsening to create a hierarchy of approximations to the original problem. An initial solution is found (sometimes for the original problem, sometimes at the coarsest level) and then iteratively refined at each level. Projection operators can transfer the solution from one level to another.

As a general solution strategy the multilevel procedure has been around for many years and has been applied to many problem areas (for example multigrid techniques can be viewed as a prime example of the multilevel paradigm). Survey papers such as [33] attest to its efficacy. However, with the exception of graph partitioning, multilevel techniques have not been widely applied to combinatorial problems.

An important prerequisite for the multilevel paradigm is a good local search strategy to carry out the refinement at each level. Unfortunately the GCP has not been generally viewed as a prime candidate for local search heuristics. Nonetheless some success has been achieved in this area and, for example, Hertz & de Werra, [15], and Glover *et al.*, [13], have applied tabu search, Johnson *et al.* have looked at simulated annealing, [17], and Culberson *et al.* have investigated the iterated greedy algorithm, [6, 7, 8]. In this paper we look at two such approaches, tabu search and the iterated greedy algorithm and apply them in a multilevel framework. In each case we aim to improve either the asymptotic convergence of the algorithm, or the runtime, or preferably both.

The rest of the paper is organised as follows. In Section 2 we discuss the multilevel paradigm, outline how it has been applied to other combinatorial optimisation problems and look at the requirements for a generic approach. Then in Section 3 we describe the implementation of a multilevel graph colouring scheme and in §3.2 outline the two local search algorithms with which it has been used. In Section 4 we look at the results from extensive testing of the algorithms on a large test suite of problem instances and draw some conclusions. Finally in Section 5 we present a summary of the work and list a number of possible methods for enhancing the techniques developed here.

2 Multilevel Optimisation

Our interest in the multilevel paradigm arises from our work in the field of graph partitioning, e.g. [38], and subsequently graph drawing, [36], and the travelling salesman problem, [35]. Typically a P-way graph partitioning algorithm aims to divide a graph into P disjoint subdomains of equal size and minimise the number of cut edges, an NP-hard problem. In recent years it has been recognised that an effective way of both accelerating graph partitioning algorithms and, more importantly, giving them a 'global' perspective, is to use multilevel techniques. The usual method is to match pairs of adjacent vertices to form *clusters*, use the clusters to define a new graph and recursively iterate this procedure until the graph size falls below some threshold. The coarsest graph is then partitioned (often with a crude algorithm) and the partition is successively refined on all the graphs starting with the coarsest and ending with the original. This sequence of contraction followed by repeated refinement loops is known as multilevel partitioning and has been successfully developed as a strategy for overcoming the localised nature of the Kernighan-Lin (KL), [22], and other partition optimisation algorithms. The multilevel partitioning paradigm was first proposed by Barnard & Simon, [1], as a method of speeding up spectral bisection and improved by Hendrickson & Leland, [14], who generalised it to encompass local refinement algorithms. Several enhancements for carrying out the matching of vertices have been devised by Karypis & Kumar, [20]. The multilevel partitioning strategy is widely used and forms the basis of several public domain partitioning packages including CHACO [14], JOSTLE [38], and METIS [20].

More recently the multilevel paradigm has been applied with significant effect to the travelling salesman problem, [35]. The resulting algorithm progressively coarsens the problem, initialises a tour and then employs the Chained Lin-Kernighan (CLK) algorithm to refine the solution on each of the coarsened problems in reverse order. In experiments on a well established test suite of 79 problem instances multilevel configurations were found that either improved the average tour quality by over 25% as compared to the standard CLK algorithm using the same amount of execution time, or that achieved approximately the same tour quality over 7 times more rapidly. Moreover the multilevel version seems to optimise far better the more clustered instances with which the CLK algorithm has the most difficulties.

2.1 The generic multilevel paradigm

Two important questions about these approaches arise – why do multilevel approaches appear to work, and, is there an abstraction of the paradigm that can be applied to other combinatorial problems?

Considered from the point of view of the multilevel procedure, a series of increasingly smaller & coarser

versions of the original problem are being constructed. It is hoped that each problem P_l retains the important features of its parent P_{l-1} but the (usually) randomised and irregular nature of the coarsening precludes any rigorous analysis of this process.

On the other hand, viewing the multilevel process from the point of view of the optimisation problem and, in particular, the objective function is considerably more enlightening. For a given problem instance the *solution space*, $\mathcal{X}$, is the set of all possible solutions for that instance. The *objective function* or *cost function*, $f : \mathcal{X} \to \mathbb{R}$, assigns a cost to each solution in $\mathcal{X}$(e.g. in the case of colouring $f : \mathcal{X} \to \mathbb{N}$ and expresses the number of colours required for a given solution). Typically the aim of the problem is to find a state $x \in \mathcal{X}$ at a minimum (or maximum) of the objective function. Iterative refinement algorithms usually attempt to do this by moving stepwise through the solution space (which is hence also known as a *search space*) but often can become trapped in local minima of f.

Suppose then for the partitioning problem that two vertices $u, v \in G_{l-1}$ are matched and coalesced into a single vertex $v' \in G_l$. When a partition refinement algorithm is subsequently used on G_l and whenever v' is assigned to a subdomain, both u & v are also both being assigned to that subdomain. In this way the partitioning of G_l is being restricted to consider only those configurations in the solution space in which u & v lie in the ***same*** subdomain. Since many vertex pairs are generally coalesced from all parts of G_{l-1} to form G_l this set of restrictions is in some way a sampling of the solution space and hence the surface of the objective function.

This is an important point (see also [35]). Previously authors have made a case for multilevel partitioning on the basis that the coarsening successively ***approximates*** the problem. In fact it is somewhat better than this; the coarsening ***samples*** the solution space by placing restrictions on which states the refinement algorithm can visit. Furthermore, this methodology is not confined to multilevel partitioning but can be applied to other combinatorial optimisation problems.

We can then hypothesise that, if the coarsening manages to sample the solution space so as to gradually ***smooth*** the objective function, the multilevel representation of the problem combined with an iterative refinement algorithm should work well as an optimisation ***meta-heuristic***. In other words, by coarsening and smoothing the problem, the multilevel component allows the refinement algorithm to find regions of the solution space where the objective function has a low average value (e.g. broad valleys). This does rely on a certain amount of 'continuity' in the objective function but it is not unusual for these sort of problems that changing one or two components of the solution tends not to change the cost very much.

Figure 1 shows an example of how this might work for a search space $\mathcal{X}$ and objective function $f(\mathcal{X})$ which we aim to minimise. On the left hand side the objective function is gradually sampled and smoothed (the sampled points are circled and all intermediate values removed to give the next coarsest representation). The initial solution for the final coarsened space (shown as a black dot in the bottom right hand figure) is then trivial (because there is only one possible state) although the resulting configuration is not an optimal solution to the overall problem. However this state is used as an initial configuration for the next level up and a *steepest descent* refinement policy finds the nearest local minimum (steepest descent refinement will only move to a neighbouring configuration if the value of the objective function is lower there). Recursing this process keeps the best found solution (indicated by the black dot) in the same region of the solution space. Finally this gives a good initial configuration for the original problem and (in this case) the optimal solution can be found. Note that it is possible to pick a different set of sampling points for this example for which the steepest descent policy will fail to find the global minimum, but this only indicates, as might be expected, that the multilevel procedure is somewhat sensitive to the coarsening strategy.

Of course, this motivational example might be considered trivial or unrealistic (in particular an objective function cannot normally be pictured in 2D). However, consider other meta-heuristics such as repeated random starts combined with steepest descent local search, or even simulated annealing, applied to this same objective function; without lucky initial guesses either might require many iterations to solve this simple problem.

It should be stressed that this hypothesis is nothing more than speculation and we cannot prove that this

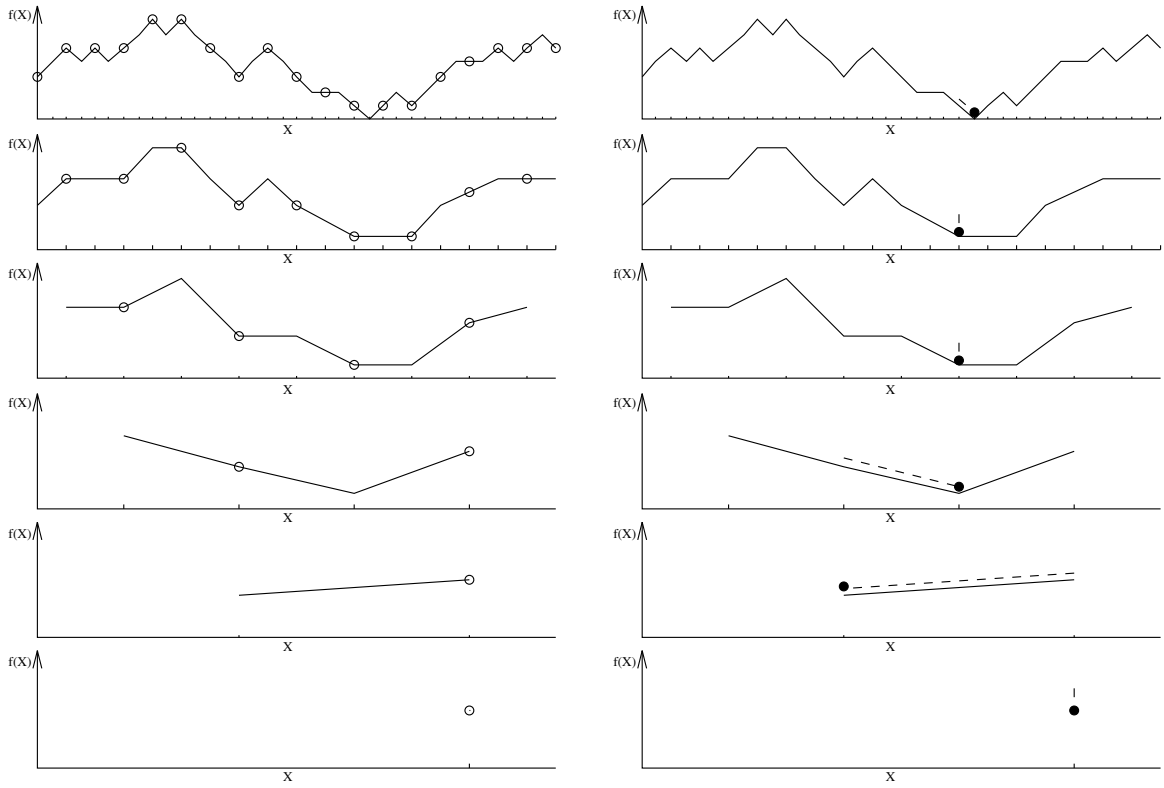


Figure 1: The multilevel scheme in terms of a simple objective function

process underlies the multilevel paradigm. However experimental evidence, here and elsewhere (see e.g. [37]), suggests that the multilevel approach does indeed enhance local search strategies and we suspect that the sampling/smoothing of the objective function contributes to this.

To summarise the paradigm then, multilevel optimisation combines a coarsening strategy together with a refinement algorithm (employed at each level in reverse order) to provide an optimisation meta-heuristic. Figure 2 contains a schematic of this process in pseudo-code. Here P_l refers to the coarsened problem after l coarsening steps, $C_l\{P_l\}$ is a solution of this problem and $C_l^0\{P_l\}$ denotes the initial solution.

2.2 Algorithmic requirements

Assuming that the above analysis does contain some elements of truth, how can we implement a multilevel strategy to test it on a given combinatorial optimisation problem?

First of all let us assume that we know of a refinement algorithm for the problem, which refines in the sense it can reuse an existing solution and (attempt to) improve it. Typically the refinement algorithm will be a local search strategy which can only explore small regions of the solution space neighbouring to the current solution. However the paradigm does not preclude the use of more complex techniques and there is no reason (other than execution time) why it should not be a more sophisticated scheme. Indeed, in the case of graph partitioning, examples of multilevel implementations exist for simulated annealing, [34], tabu search, [2, 34], and even genetic algorithms, [21].

The refinement algorithm must also be able to cope with any additional restrictions placed on it by using a coarsened problem (e.g. in graph partitioning the coarser graphs are always weighted whether or not the original is). If such a refinement algorithm does not exist (e.g. if the only known heuristics for the problem

```
multilevel refinement(input problem instance P_0 , output solution C_0{P_0} )
begin
        l := 1
        while (coarsening)
                P_{l+1} = coarsen( P_l )
                l := l+1
        end
        C_l{P_l} = initialise( P_l )
        while l > 0
                l := l−1
                C_l^0{P_l} = extend( C_{l+1}{P_{l+1}}, P_l )
                C_l{P_l} = refine( C_l^0{P_l}, P_l )
        end
end
```

Figure 2: The multilevel optimisation algorithm

are based on construction rather than refinement) it is not clear that the multilevel paradigm can be applied.

To implement a multilevel algorithm, given a problem and a refinement strategy for it, we then require three additional basic components: a coarsening algorithm, an initialisation algorithm and an extension algorithm (which takes the solution on one problem and extends it to the parent problem). It is difficult to talk in general terms about these requirements, but the existing examples from graph partitioning and the travelling salesman problem (TSP) suggest that the extension algorithm can be a simple and obvious reversal of the coarsening step which preserves the same cost. For example in partitioning a pair of parent vertices are assigned to the same subdomain as their child. The initialisation is also generally a simple canonical mapping (e.g. for P-way partitioning – assign P vertices to P subdomains; for the TSP – construct a tour to visit 2 cities). By canonical we mean that a (non-unique) solution is 'obvious' and that the refinement algorithm cannot possibly improve on the initial solution at the coarsest level (because there are no degrees of freedom).

This just leaves the coarsening algorithm which is then perhaps the key component of a multilevel optimisation implementation. For the existing examples two principles seem to hold:

- Any solution in any of the coarsened spaces should induce a legitimate solution on the original space. Thus at any stage after initialisation the current solution could simply be extended through all the problem levels to achieve a solution of the original problem. Furthermore both solutions (in the coarse space and the original space) should have the same cost with respect to the objective function. This requirement ensures that the coarsening is truly sampling the solution space (rather than approximating and/or distorting it).
- The number of levels need not be determined *a priori* but coarsening should cease when any further coarsening would render the initialisation degenerate.

This still does not tell us *how* to coarsen a given problem. So far most solutions for the partitioning problem have employed gradual and fairly uniform contraction of the graphs. Furthermore it has been shown (for partitioning at least), that it is usually more profitable for the coarsening to respect the objective function in some sense (see e.g. the heavy edge matching strategy in [20]). In this respect it seems likely that the most difficult aspect of finding an effective multilevel algorithm for a given problem and given refinement scheme is the (problem dependent) task of devising the coarsening strategy.

3 A Multilevel Algorithm for the Graph Colouring Problem

In this section we describe the derivation and implementation of a multilevel algorithm for the graph colouring problem. Firstly we describe all the operations necessary for the multilevel scheme to operate and then in §3.2 outline the two local search heuristics that we have investigated in a multilevel framework.

3.1 Multilevel operations

We have implemented the coarsening and uncoarsening procedures in a similar manner to that used in graph partitioning. Thus each coarse graph G_{l+1} is created from its parent graph G_l by matching pairs of vertices and representing each matched pair v_1 & v_2 of parent vertices in G_l with a child vertex in G_{l+1}. Figure 3 shows an example of this with a graph of 7 vertices coarsened down to a (complete) graph of 3 vertices in 2 contraction steps. The dotted lines indicate the vertex matching used to create the child graph at each stage. A colouring is initialised on the coarsest graph, Figure 3(c), a trivial operation since this graph is complete (every vertex is adjacent to every other) and so each vertex is assigned a different colour. During the optimisation, which successively refines the solution on each graph in reverse order, coarsest to finest, a solution of the child graph G_{l+1} is extended to its parent G_l. In fact, for the example in Figure 3, no optimisation would be possible since the original graph contains a 3-clique (e.g. $\{v_1, v_4, v_5\}$) and so $\chi(G) \geq 3$ (since every vertex in a clique must have a different colour).

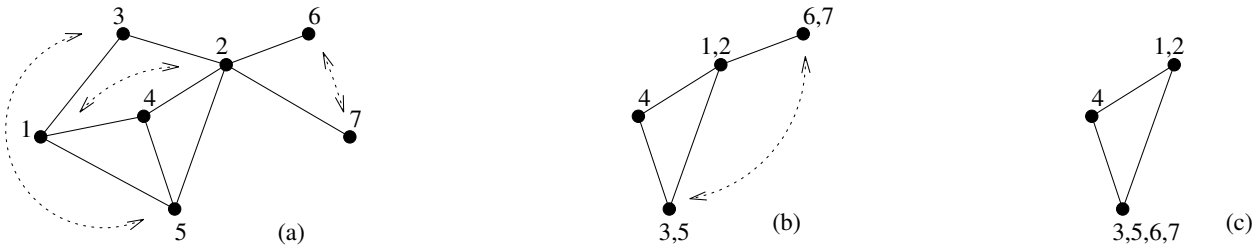


Figure 3: An example of graph contraction by vertex matching

3.1.1 Vertex matching

The matching procedure is also based fairly closely on algorithms used in graph partitioning, e.g. [14, 38], with one important difference. In partitioning it is normal to match ***neighbouring*** vertices on the basis that if a child vertex $w \in G_{l+1}$ is assigned to a set then, when the partition is extended to G_l, the pair of vertices that w represents are then assigned to the same set with the edge between them uncut by the partition. Initially we tried the same procedure for a preliminary implementation of the colouring algorithm. In this version, if a child vertex, $w \in G_{l+1}$, is assigned a colour, c say, then the colouring on graph G_{l+1} can be extended to G_l without any colouring conflicts by assigning the colours $2c$ and $2c+1$ to its parents in G_l. In this way if graph G_l has a colouring with k_l colours then the colouring can be extended to G_0, the original graph, to give a colouring for G_0 with a maximum of $2^l k_l$ colours although we would expect to reduce this by optimisation at each level.

Early investigations with such a scheme proved unsuccessful and so we replaced it with a perhaps more natural scheme of matching vertices that were not adjacent. This works on the basis that if a child vertex w is assigned a colour then the same colour can be assigned to its parents without colouring conflicts. Furthermore we only allow vertices to be matched with neighbours of neighbours rather than any non-adjacent vertex. Thus, in Figure 3(a), v_1 is allowed to match with v_2 but not v_3, v_4, v_5 (because they are adjacent) and not with v_6, v_7 (because they are not neighbours of neighbours).

To express this in set notation, denote the neighbourhood of a set S as $\Gamma(S)$, the set of vertices adjacent to, but not including, vertices in S, i.e. $\Gamma(S) = \{v \in V - S : \text{ there exists } u \in S \,\&\, (u, v) \in E\}$. Then a vertex v is allowed to match with any vertex in $\Gamma(\Gamma(\{v\}))$, or $\Gamma^2(v)$ for short, the set of neighbours of neighbours of v, rather than any vertex in $V - \Gamma(v)$, the set of vertices not adjacent to v. In fact there is no mathematical reason why a matching should be restricted in this way, however, as we shall see below (§4.3), it is easier to prioritise which vertex to match with if the candidate set is $\Gamma^2(v)$ rather than $V - \Gamma(v)$.

The matching algorithm we use is essentially the same as for many of the graph partitioning implementations, e.g. [14, 20, 38]. We pick an ordering of the vertices and they are visited in turn using a linked list. If a vertex v has unmatched candidate vertices (i.e. $\Gamma^2(v)$ is non-empty and contains vertices which are not yet matched) then a candidate vertex u is selected and u & v are matched and removed from the list. If a vertex has no unmatched candidates then it is matched with itself and removed from the list.

This leaves just two 'parameters' to the method: (a) how to choose the initial ordering of the vertices, and (b) prioritising the candidate vertices in order to select one which will best aid the colouring algorithms. We discuss these parameters further in §4.3 below.

3.1.2 Graph contraction

The child graph is constructed by merging matched pairs of vertices and representing them with a single child vertex. Edges which then become duplicated are also merged. Although our algorithms do not use weighted graphs we do not preclude the possibility of using them later (see §5.4) and it is easy to see how a child graph should be weighted. Essentially weights represent the number of vertices and edges that existed in the original graph. Thus with the matching shown in Figure 3(a) to give the graph in Figure 3(b), then using $|.|$ to denote the weight of a vertex or edge, vertex $v'_{1,2}$ has weight $|v_1| + |v_2|$, etc. Similarly for the edges, $|(v'_{1,2}, v'_{3,5})| = |(v_1, v_3)| + |(v_1, v_5)| + |(v_2, v_3)| + |(v_2, v_5)|$, etc.

3.1.3 Termination

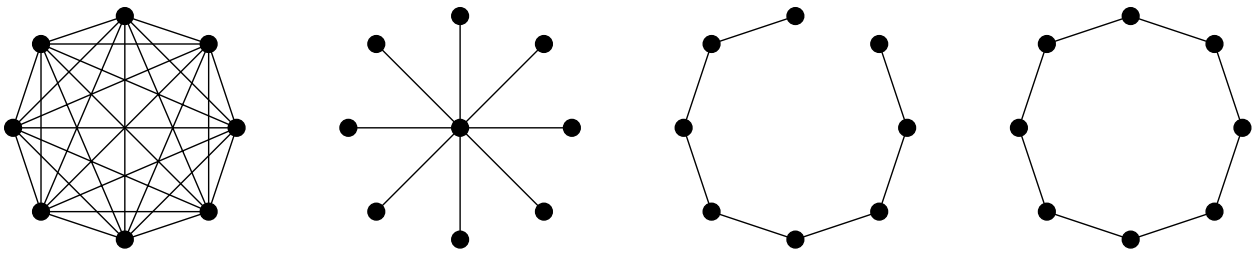


Figure 4: Graph primitives: a complete graph, a star, a chain and a ring

Assume for now that the graph is connected (i.e. by traversing edges we can reach any vertex from any other vertex) and see below, §3.1.6, for when this is not the case. The coarsening process can be terminated when the initialisation is trivial. This certainly occurs when a child graph turns out to be a complete graph (i.e. all vertices are adjacent to each other) and in this case matching is no longer possible anyway, as in Figure 3(c). In fact it is not difficult to see that if the graph is connected but not complete then matching and hence contraction is always possible and that, since contraction always reduces the size of the graph, the process must always result in a complete graph (even if it only contains two vertices and one edge). Indeed the process can be terminated even earlier if we can identify the graph easily and if we know a trivial colouring algorithm for the graph. Figure 4 shows the four graph primitives we have encoded into the termination algorithm, a complete graph, a star, a chain and a ring. Given N, the number of vertices, E, the number of edges and D, the maximum degree of any vertex, these can be easily identified with $O(1)$

operations. Thus if $E = \frac{1}{2}N(N-1)$ the graph is complete, if $E = N-1$ & $D = N-1$ the graph is a star, if $D = 2$ & $E = N-1$ the graph is a chain and if $D = 2$ & $E = N$ the graph is a ring.

3.1.4 Initialisation

The initial colouring is trivial (and optimal for the coarsest graph at least) provided the graph corresponds to one of the four primitives. A complete graph, G, can only be coloured by giving a different colour to each vertex and so $\chi(G) = N$, a star and a chain can be coloured with 2 colours, whilst a ring requires 2 colours if N is even and 3 if N is odd.

3.1.5 Extension

Having derived and possibly refined a k-colouring on a graph level G_l we must extend it to the parent graph G_{l-1}. In fact, as indicated above, this is a trivial operation and, since each pair of parent vertices are never adjacent, simply assigning the same colour to them as their child renders a legitimate k-colouring for G_{l-1}.

3.1.6 Disconnected graphs

Although most refinement strategies can cope with disconnected graphs, they do need special handling for the multilevel procedure. We do not include isolated vertices (i.e. with no neighbours and degree 0) in this category as they are the special case of trivial components and can all be assigned the same colour (since their colouring does not affect the rest of the graph). Here then, disconnected refers to graphs with two or more non-trivial components each of which cannot be reached from another by traversing paths of edges.

If the original graph has C components, the coarsening will eventually result in C disjoint cliques and no further matching will be possible (unless we allow matches between vertices in different components). However it is not possible to determine how many cliques there are using an $O(1)$ operation and so the identification of the graph primitives becomes a more costly procedure. Instead, therefore, we choose to identify the components prior to the multilevel procedure (using a breadth first search) and then do the colouring on a component by component basis. In fact this can also save time because, if more than one component is detected, we sort the components by size and colour them largest first. Suppose then that prior to the colouring of a component C_j, the optimisation has found k_i-colouring for each of the previous components C_i, $i = 1, \ldots, j-1$ and that $k^* = \max_{1 \le i < j} k_i$. During the subsequent colouring of C_j, if a k_j-colouring is found with $k_j \le k^*$ then the optimisation of C_j can be terminated immediately (because, even if the colouring of C_j is suboptimal, the final colouring number of the graph cannot be less than k^*).

3.1.7 Solution based coarsening

If a colouring of the graph already exists prior to optimisation it can be reused during the multilevel procedure. Thus, given a k-colouring of the original problem we can carry out *solution based coarsening* by insisting that, at each level, every vertex v matches with a candidate vertex (i.e. in $\Gamma^2(v)$) of the same colour. Colours are injected up the hierarchy in the same way as they are extended downwards; a child vertex inherits the mutual colouring of its parents. Clearly this does not increase the colours in the graph and when no further coarsening is possible we end up with a graph of k or more vertices together with a k-colouring for it. Thus, provided the refinement algorithms guarantee not to find a worse colouring than the initial one (in fact even if they do find a worse colouring it can be replaced by the initial one) the process can guarantee to find a k'-colouring with $k' \le k$, the initial value. Although we have not made any great use of this technique here, we do discuss its use in two contexts below, §4.4 & §5.2.

3.2 Refinement

In this section we outline two alternative algorithms that we use for the refinement phase of the multilevel procedure. They have been chosen primarily because the source code, written by Culberson, [5], is available for them[1] thus avoiding the need for a re-implementation and also giving a degree of objectivity to the comparison of a multilevel version with the original version. However they are also of interest because each offers a completely different approach to iterative colouring via local search.

3.2.1 The iterated greedy algorithm

The greedy algorithm (or sequential algorithm as it is sometimes known) was one of the earliest heuristics for the graph colouring problem, e.g. [3, 27]. The idea is to visit the graph in a specified order and insert each successive vertex into the minimum *colour class* that does not cause any conflicts with previously coloured vertices. Here each colour class, C_j, is an *independent set* of vertices (i.e. no two vertices in a set are adjacent) assigned the same colour, j. Various suggestions have been proposed for the initial ordering of the vertices (e.g. based on vertex degree [3, 27]). The greedy algorithm is a *constructive approach* (i.e. a solution is constructed from scratch rather than refined).

Constructive algorithms are generally seen as a single-pass approach to finding a solution and are often used as an initialisation procedure for iterative refinement methods. The iterated version of this algorithm, however, relies on a clever observation, [6, 8], about the reordering of an existing greedy colouring. Given any k-colouring of a graph, if the vertices are reordered so that vertices in each colour class are contiguous then it is trivial to prove that using the greedy procedure on this new ordering will result in another colouring with no more than k colours, [6]. In fact, if the previous colouring has been generated by the greedy algorithm, then for a colour class, C_i, every vertex in C_i must be adjacent to a vertex in C_j for $1 \leq j < i$. However the converse does not hold and every vertex in C_j, for some $1 \leq j < i$, need not be adjacent to a vertex in C_i. Therefore, if the colour classes are reordered (whilst maintaining the property that the vertices of each class are contiguous) so that the vertices in C_i precede those in C_j in the ordering, then it is possible that some of the vertices in C_j may be given a different colour and that the greedy algorithm may find a colouring with fewer than k colours.

This neat argument forms the basis of Culberson's iterated greedy algorithm, [6, 7, 8]. At each iteration a reordering of the colour classes is chosen from a variety of possibilities (e.g. reversing the order, random order, or sorted so that the classes are in order of decreasing total degree of each class). Furthermore in Culberson's implementation, the code will randomly pick one of these possibilities according to a weighting supplied by the user. This gives the algorithm many different possibilities to jump out of local minima traps. The implementation also allows the user to specify a search intensity, λ, in the form of the number of failed iterations; i.e. if no improvement in the cost function is seen after λ iterations the algorithm terminates. In this context the cost function is expressed not only in terms of the colouring number k, but also the *colouring sum*, $\Sigma_v \pi(v)$, where $\pi(v)$ is the colour assigned to vertex v. Thus improvements are sought in the value of

$$f(G, \pi) = Nk + \Sigma_{v \in V} \pi(v)$$

for a given colouring π, where N is the number of vertices (used as a scaling factor). Clearly this does not represent the true cost function that we seek to minimise (which is just $f(G, \pi) = k$, the number of colours) and indeed there exist graphs for which the minimum colouring sum is arbitrarily smaller than the colouring sum of any optimal colouring and graphs for which the chromatic number is arbitrarily smaller than the colouring number for any minimum colouring sum, [6]. Nonetheless this measure does reflect progress towards improved colourings for many graphs since it can indicate if colouring classes with a high index are shrinking and hence if they are likely to disappear. It therefore appears to be better than simply using $f(G, \pi) = k$ which gives no indication of whether the current colouring is in the region of a better one.

[1] via `http://www.cs.ualberta.ca/~joe/Coloring/Colorsrc/index.html`

3.2.2 Tabu search

Tabu search is a general technique, proposed by Glover, [12], for finding approximate solutions to combinatorial optimisation problems. Given an existing solution, the search moves stepwise through the solution space and at each iteration steps to the neighbouring solution with the lowest cost (even if that cost is higher than the current one). However to prevent cyclic behaviour, i.e. stepping straight back to the solutions that the algorithm has just left and hence becoming stuck in local minima, a 'tabu' list is maintained containing disallowed moves to states that the algorithm has recently visited. Generally moves only remain tabu during a certain number of iterations and so the tabu list is normally implemented as a fixed length queue where the oldest move is dropped every time a new move is added.

Hertz & de Werra proposed a tabu search algorithm for the graph colouring problem in [15]. Strictly speaking this implementation does not move through the solution space but instead moves through a closely related space to try and find a legitimate colouring. Thus given an existing k-colouring of the graph and a target colour, $k_t < k$, the vertices of the graph are placed in k_t colour classes (with inevitable colouring conflicts). This is a point of the search space and neighbours of this point can be generated by picking any vertex in conflict and moving it to a different colour class. Hertz & de Werra's algorithm works by generating a neighbourhood of a chosen size at random (because for reasonably sized graphs, generating every neighbour would be prohibitively expensive) and stepping to the neighbour with the minimum number of conflicts. If a state is found with no conflicts then a k_t-colouring has been achieved and the search terminates.

Culberson's implementation of this algorithm (which is used for this paper) also adds a couple of additional features (and as we shall see below one is very important). Firstly, once there are only 1 or 2 colouring conflicts, the code switches to a brute force search to try and resolve them. This generally works well when used in the context of a single-level tabu search, however, when used *in extremis*, and in particular on the coarsest problems in the multilevel hierarchy, it can be very expensive. In such graphs there are sometimes many completely connected vertices (i.e. adjacent to every other vertex) for which the brute force search can never resolve conflicts. The problem is easily avoided, however, by filtering out all such vertices prior to the use of tabu search and our implementation does this.

A more important feature occurs once a conflict free state has been found. During the tabu iterations, the vertices in each colour class are stored as linked lists with no explicit mapping of vertices to colours. However, once a state has been found with no conflicts the vertices of each colour class are listed contiguously (in an array) and then given an explicit colouring using the greedy algorithm. At this point the same feature applies as in the iterated greedy procedure and the greedy algorithm acting on such a list may produce a better colouring than the k_t-colouring found by the tabu iterations.

Of course there is a strong possibility that no k_t-colouring will be found and and so additional termination criterion are needed. In Culberson's implementation once again the search intensity, λ, can be specified and the algorithm will terminate if no improvement is seen in the cost function (in this case the number of edge conflicts, [8]) after λ iterations. If failure occurs then the code resets $k_t := k_t + 1$ and makes another colouring attempt. This is repeated until a colouring is successfully found or it has made a user specified number of attempts.

Perhaps the most difficult feature in using Hertz & de Werra's tabu algorithm is selecting (automatically) the target colouring number k_t, even given an initial k_0-colouring. Clearly, whatever k_t is selected we can limit n_a, the number of attempts, to $n_a = k_0 - k_t - 1$ and look for a k-colouring for each $k_t \leq k < k_0$. An obvious brute force choice of k_t therefore is some lower bound, k_l, on the chromatic number of the graph, $\chi(G)$. One such lower bound would be the size of the largest clique in the graph (since every vertex in a clique must have a different colour) but, since finding the maximum clique of a graph is also an NP-hard problem, [11], we must either put a considerable amount of effort into getting a good approximation or put up with a quick but crude estimate. In the latter case the subsequent colouring process may be exorbitantly expensive, especially for graphs with large chromatic number. For example, the best colouring we are aware of for the C4000.5 graph from the test suite (§4.2) has 280 colours, yet in tests a simple greedy maximum clique algorithm only found cliques of size 14 or 15. This would mean that the search is likely to

fail over 250 times and probably (although we do not know $\chi(G)$ for this graph) most of these searches will be looking for a k-colouring with $k < \chi(G)$, i.e. non-existent. Even making a more conservative estimate based on k_0 can be expensive. For example setting $k_t = (k_l + k_0)/2$ could still involve a large number of doomed iterations, especially if k_0 is already close to $\chi(G)$.

As an alternative to this bottom up approach (i.e. start somewhere at or below $\chi(G)$ and work upwards) we have found that a top down approach works very well. Thus given a k_0-colouring, we simply set $k_t = k_0 - 1$. If a k_t-colouring is found, we set $k_t := k_t - 1$ and repeat. The process is iterated until failure occurs.

It might seem that such a procedure could be as expensive as the bottom up process and, particularly if k_0 is far from $\chi(G)$, many iterations might be required. However, experience suggests that if the existing colouring is poor then the tabu search algorithm generally finds a better one very rapidly (because there will only be a few conflicts). More importantly, the greedy algorithm used in Culberson's implementation to explicitly assign the colouring (see above) will then move the solution to a local minimum (local in the context of the greedy algorithm) and may very well find a better colouring (sometimes substantially better if the initial colouring is very poor). In this way the combination of tabu step plus greedy acts very much like the chained local optimisation scheme of Martin & Otto, [26], where simulated annealing is used to take 'big' steps around the solution space and is followed a local search algorithm which moves the solution to a nearby local minimum. A similar principle is applied by Moscato, [30], who suggests the use of genetic algorithms in combination with local search techniques to give what he terms as a *memetic* algorithm.

4 Experimental Results

We have implemented the algorithms described above to give two multilevel optimisation schemes, a multilevel iterated greedy algorithm (MLIG), and multilevel tabu search (MLTS). In this section we shall compare them with the original algorithms, iterated greedy (IG) and tabu search (TS). In each case our aim is not necessarily to determine the best procedure; rather, we wish to investigate, given a local search strategy, whether a multilevel version can either accelerate the convergence rate of the local search or even improve the asymptotic convergence in solution quality.

The tests were carried out on a DEC Alpha machine with a 466 MHz CPU and 1 Gbyte of memory. The code was written in C and compiled with the GNU C compiler (version 2.95.2) and optimisation `gcc -O3`. Unfortunately machine benchmark examples of the form used for papers in [18] were no longer available but an appropriate scaling for the runtimes in Tables 2 & 3 can be calculated by comparing the results in Table 2 with those in [8].

4.1 Methodology

Typically local search algorithms contain a parameter which allows the user to specify how long the search should continue before giving up. At its simplest this can be just a given time interval, but perhaps more commonly it is the number of iterations of some outer loop of the algorithm, either in absolute terms or (as above in §3.2.1 & §3.2.2) in terms of the number of failures to achieve a better solution. We refer to this as the *intensity* of the refinement policy.

To assess a given algorithm, we measure the runtime and solution quality for a chosen group of problem instances and for a variety of intensities. Since all of the algorithms tested here have a degree of randomisation we run each test with several random seed values. For problem instance p, at search intensity λ, with random seed s, this gives a pair, $Q_{\lambda,p,s}$, the solution quality found (i.e. the colouring number) and $T_{\lambda,p,s}\}$, the runtime. For each intensity value and problem instance we average the solution quality and runtime

results over the number of seed values, n_s, to give

$$\overline{Q}_{\lambda,p} = \frac{1}{n_s}\sum_{s=1}^{n_s} Q_{\lambda,p,s} \quad , \quad \overline{T}_{\lambda,p} = \frac{1}{n_s}\sum_{s=1}^{n_s} T_{\lambda,p,s}.$$

We then normalise these values with the best known solution quality for that problem instance and some reference runtime to prevent instances with a large absolute solution quality or larger than average runtime from dominating the results. Finally these normalised values are averaged over all problem instances to give a single data point of averaged normalised solution quality, Q_λ, and runtime, T_λ, for a given intensity λ. By using several intensity values, λ, we can then plot Q_λ against T_λ to give an indication of algorithmic performance over those instances.

The normalisation of solution quality is calculated as $(\overline{Q}_{\lambda,p} - Q^*_p)/Q^*_p$ where Q^*_p is the best known solution for instance p. If the chromatic number of p is known then $Q^*_p = \chi(p)$; if not then we use the best known value found either by our testing or taken from the literature. Of course this is not ideal, and as Leighton points out, [23], given two k-colourings for a graph G with $k = 20$ and $k = 22$ say, the conclusions that would be drawn if $\chi(G) = 20$ might be very different if $\chi(G) = 4$. Nonetheless, the fact that the pseudo-optimal solutions have been established over a wide range of tests and algorithms gives a good indication of the current best possible heuristic solution, even if no heuristic can get close to the optimal solution. Besides, the alternative of limiting the tests to those (small) problems with known chromatic number would be far too restrictive.

The relative normalisation means that we can express the solution quality as a percentage excess over the pseudo-optimal colouring number, i.e. $100 \times (\overline{Q}_{\lambda,p} - Q^*_p)/Q^*_p\%$. It can be argued that normalisation in this manner tends to mean that problem instances with small Q^*_p could dominate the results. For example if $\overline{Q}_{\lambda,p_1} = 5$ and $Q^*_{p_1} = 4$ then the normalised quality is 25% in excess whereas if $\overline{Q}_{\lambda,p_2} = 101$ and $Q^*_{p_2} = 100$ then the excess is only 1% even though in both cases the colouring is just 1 away from optimal. On the other hand using absolute normalisation, $(\overline{Q}_{\lambda,p} - Q^*_p)$, rather than relative means that problem instances with large Q^*_p may dominate. For example if $\overline{Q}_{\lambda,p_1} = 125$ and $Q^*_{p_1} = 100$ whilst $\overline{Q}_{\lambda,p_2} = 30$ and $Q^*_{p_2} = 5$, then in both cases the normalised quality is 25 even though experience suggests that the colouring for p_2 is more likely to be the worse of the two. In fact we have tried absolute normalisation for the following results but generally it does not change the qualitative behaviour and this is especially true in the results classified by density.

The time normalisation is more simple and we just calculate $\overline{T}_{\lambda,p}/T^G_{1,p}$ where $T^G_{1,p}$ is the average runtime for a single pass of the greedy algorithm on an instance p (calculated by averaging 3 runs of the iterated greedy algorithm with different test seeds each of which makes 1,000 calls to greedy).

To summarise then, for a set of problem instances P, we plot averaged normalised solution quality Q_λ against averaged normalised runtime T_λ for a variety of intensities, λ, and where:

$$Q_\lambda = \frac{1}{|P|}\sum_{p\in P} \frac{(\overline{Q}_{\lambda,p} - Q^*_p)}{Q^*_p} \quad , \quad T_\lambda = \frac{1}{|P|}\sum_{p\in P} \frac{\overline{T}_{\lambda,p}}{T^G_{1,p}}.$$

4.2 Test suite

Table 1 lists the entire test suite of 90 problem instances used for testing the algorithms. The suite consists of the examples compiled for the 2nd DIMACS implementation challenge, [18], augmented by further examples added since then[2]. Although a subset of this test suite has been used for the DIMACS challenge, we decided to use the full set of examples available because most of the DIMACS subset are of medium density and, as we shall see later, the multilevel framework does not perform well on such instances.

For each instance we give its size in terms of $|V|$, the number of vertices, and $|E|$, the number of edges, as well as the density, Δ, which is defined as $\Delta = 2|E|/|V|(|V|-1)$ (so that a complete graph with $|V|(|V|-1)/2$

[2]available from `http://mat.gsia.cmu.edu/COLOR/instances.html`

edges has density 1 or 100% density). We also list the maximum and minimum degree of the vertices for each graph. The column headed $\chi(p)$ gives the chromatic number of the graph if known or, if not, an upper bound provided by the best known colouring number found either in our experiments or taken from the literature and in particular the papers [8, 13, 10, 19, 24, 29, 32].

The names of the graphs give an indication of their origin. For example CN.Δ and DSJCN.Δ are random graphs of size N and approximate density Δ, RN.Δ and DSJRN.Δ are random geometric graphs with similar specification, le$N$_χ are Leighton graphs with chromatic number χ as described in [23], flat$N$_χ are flat graphs as described by Culberson in [7], etc. The colouring suite also includes examples drawn from real applications such as course scheduling and register allocation together with some other interesting colouring problems. Further details on the origin of the graphs can be found in e.g. [24][3].

4.3 Parameter settings

An important part of the preliminary testing was to establish a good choice of parameters for the methods under consideration. For iterated greedy and tabu search we used parameters suggested in the papers of Culberson *et al.*, [6, 8], and the manual, [5]. Thus for iterated greedy we used a ratio of 50:50:30 to weight the random choice of algorithm for permuting the colour classes to *largest first* : *reverse order* : *random order*. We weighted the random choice of which order to test for conflicts with existing colour classes (thus prioritising the colouring choice if there are several classes with no conflicts) using the ratio 100:100:50 with respect to *in order* : *largest first* : *random order*. We also used Kempe reductions every 30 iterations (see also [17]). For tabu search, the size of the tabu list was set to 7 except for very small (coarse) graphs with $|V| \leq 20$ in which case it was set to $|V|/3$. We also set the code to generate a maximum of 600 neighbours in the search space, and a minimum of 2 (see [8]). Finally, when the tabu search algorithm is used as a standalone (as opposed to within the multilevel framework) an initial colouring is required. After some experimentation using either a trivial colouring (every vertex initially assigned a different colour) or different greedy variants we found that tabu search appeared to work best when initialised by a greedy colouring with the vertices ordered by decreasing degree (i.e. largest first) and the colour classes examined in order. With these parameters fixed this just leaves the search intensity and random seed value as input.

For the multilevel versions a couple more parameter choices must be made. First of all for an input search intensity, λ, it is intuitively somewhat extravagant to search at the same intensity at every level, especially given that the coarsest levels are very restricted versions of the original problem and that refinement possibilities are less likely. After some experimentation we chose to set λ_l, the search intensity at level l, to $\lambda_l = \lambda/(l+1)$ where the original problem is level 0 and so $\lambda_0 = \lambda$. This appears to work reasonably well and gives very similar results to those obtained by simply setting $\lambda_l = \lambda$, only somewhat more rapidly. However, this parameter could well benefit from further investigation.

The other important parameters for the multilevel scheme relate to the matching (as mentioned in §3.1.1). Again after considerable experimentation we found that initially sorting the vertices by decreasing degree (i.e. largest first) appeared to give the best results. However we also tried sorting by increasing degree, by smallest difference from the mean degree and randomly, all of which gave broadly similar results.

The second choice to make for the matching algorithm is, given a vertex v, how to prioritise the candidate vertices (i.e. unmatched vertices in $\Gamma^2(v)$) in order to select the 'best' one to match with. For two non-adjacent vertices u, v, let $C_{u,v}$ denote the set of common neighbours, i.e. $C_{u,v} = \Gamma(u) \cap \Gamma(v)$, and denote the size of this set as $|C_{u,v}|$. We can then also define the set of distinct neighbours as $D_{u,v} = \Gamma(u) \cup \Gamma(v) - C_{u,v}$ (i.e. those neighbours of u which are not neighbours of v and vice-versa) and the size of this set is just $|D_{u,v}| = |\Gamma(u)| + |\Gamma(v)| - 2|C_{u,v}|$. The priority which then seemed to work best was, for a vertex v, to pick u which maximised $|C_{u,v}|$, the number of common neighbours. In the event that there were several such vertices, the one which minimised $|D_{u,v}|$ was chosen and, in the event of a further tie-break, a random choice was made. Again this method was selected after considerable experimentation using different initial orderings of the vertices and/or using the priorities reversed (i.e. minimise $|D_{u,v}|$ and then maximise $|C_{u,v}|$). These

[3] and from `http://mat.gsia.cmu.edu/COLOR/instances.html`

selection heuristics echo those used in the Recursive Largest First algorithm of Leighton, [23], which aims to minimise the number of edges in the remaining induced uncoloured subgraph (equivalent to maximising $|C_{u,v}|$) and the algorithm of Johnson, [16], which aims to pick a vertex of minimal degree in the remaining induced uncoloured subgraph (equivalent to minimising $|D_{u,v}|$). The difference is that in those algorithms the chosen vertices are successively added to an independent set, whilst for matching we are effectively creating many independent sets each of size two.

4.4 Greedy initialisation

One feature of the results only became apparent after closer inspection of individual examples. For certain graphs the greedy initialisation of the tabu search was able to find extremely good (and sometimes optimal) colourings which the tabu search could not improve on. Usually the multilevel version was able to find equally good colourings but in certain cases it was not. These failures, which to a certain extent were dependent on the random seed and the intensity, occurred most notably on some of the medium and high-density random graphs (in particular R1000.5, R125.1c, R250.1c, R1000.1c & DSJR500.5). Although this is clearly a deficiency in the multilevel algorithm it also skews the results somewhat. In these tests we are interested in comparing a local search algorithm, LS, with its multilevel counterpart, MLLS. Clearly if the greedy initialisation, G, dominates both these results then we end up comparing the MLLS algorithm against G, rather than G+LS, which demonstrates little about the effectiveness of the multilevel strategy. One way around this would be to drop such examples from the test suite. However it is somewhat unscientific to remove instances which do not conform to the behaviour of interest and furthermore it is not completely clear which to drop. Besides we are looking for a method which can deal with all examples well. As a solution therefore we have introduced a single pass of the greedy algorithm, G, (using largest first ordering) prior to the multilevel optimisation, MLLS. At the end of the test, the code then picks the best solution of either G or MLLS. All the results include this simple addition which only adds a relatively small time penalty, especially for high intensity tests, and henceforth when we refer to MLLS results we shall take this to mean the best of G or MLLS.

Of course one possibility would be to use the greedy partition to create a solution based coarsening (see §3.1.7) and then use multilevel optimisation on that hierarchy of graphs. However some preliminary investigations of this technique did not reveal particularly good results. In fact such an approach sometimes seems to inhibit the highest quality colourings from being found by high intensity searches and we suspect that this is because the greedy initialisation pulls the hierarchy of graphs too much towards a 'basin of attraction' in the objective function.

4.5 Main results

After the preliminary investigation we conducted tests on all of the examples in the test suite with all 4 algorithms, TS, MLTS, IG & MLIG and at several intensities $\lambda = 2^m$ for $m = 0, \ldots, 15$ (i.e. ranging between $\lambda = 1$ and $\lambda = 32{,}768$). Typically for an intensity value, λ, a multilevel algorithm MLLS$^\lambda$ will take very approximately the same time to run as the single-level local search at double the intensity, LS$^{2\lambda}$. This is because, for the multilevel algorithm, the contraction plus optimisation on all of the coarsened graphs takes roughly the same time as LS$^\lambda$, the optimisation on the final graph (see [35]) and so the total runtime is approximately $2T(\text{MLLS}^\lambda)$, where $T(\text{A})$ is the runtime of algorithm A. Because here the intensity refers to the number of failed iterations rather than an absolute number, there is no direct correspondence between the running time of LS$^\lambda$ and LS$^{2\lambda}$. However this rule of thumb that $T(\text{MLLS}^\lambda) \approx T(\text{LS}^{2\lambda})$ appears to work reasonably well (at least for tabu search).

Figure 5 shows the convergence behaviour (using the methodology described in §4.1) for the entire test suite. The final points on the local search curves are at intensity $\lambda = 32{,}768$ for TS and $\lambda = 16{,}384$ for IG since at these points the convergence appears to be tailing off. With the runtime factor of two in mind, the final points on the multilevel curves were then set at $\lambda = 16{,}384$ for MLTS and $\lambda = 8{,}192$ for MLIG.

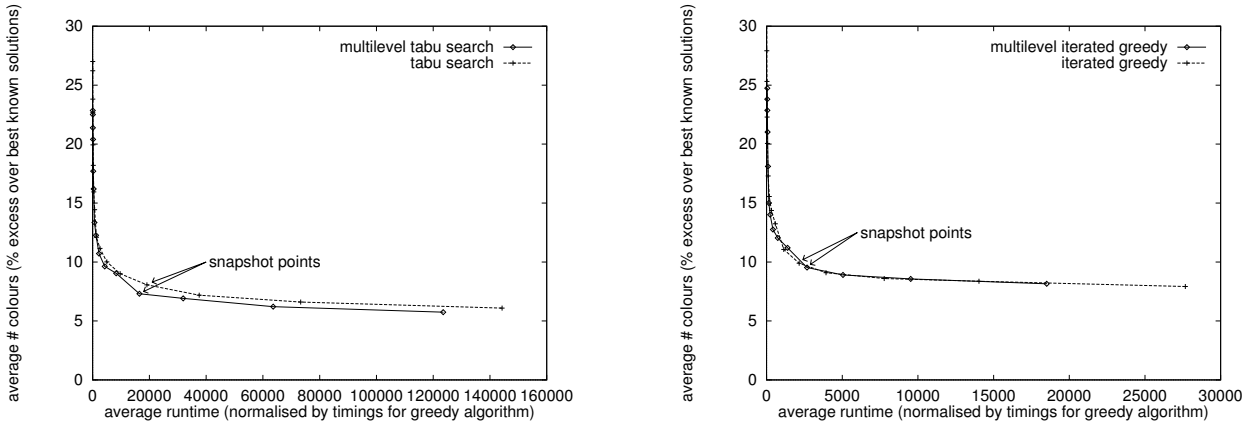


Figure 5: Results on the complete test suite

Clearly these plots summarise a vast amount of data (90 instances, 3 random seeds, 4 algorithms and an average of 14 intensities for each algorithm) and so we present a snapshot of the data by picking one point on each curve (marked with an arrow) and giving individual results in Tables 2 & 3 (MLIG/IG & MLTS/TS respectively). Thus for each instance we show minimum & average colouring numbers and the average runtime, averaged over 3 random seeds. The points we chose were at intensity $\lambda = 1{,}024$ for both IG and MLIG (close to Culberson's suggestion of 1,000, [8]) in Table 2 together with $\lambda = 4{,}096$ for TS and $\lambda = 2{,}048$ for MLTS (Table 3). Despite IG and MLIG having the same search intensity, these points are reasonably close together in average runtime for each pair of algorithms (16,485 for TS & 19,135 for MLTS; 2,156 for IG & 2,674 for MLIG) and thus make for a good comparison. They are also close to the turning point in the gradient for each curve and so give a good indication of maximum return for minimum effort. In fact in most cases the optimisation has already found the pseudo-optimal solution for the problem at this point. However it is very difficult to draw any overall conclusion from this sort of results table.

Returning to Figure 5 then, unfortunately the convergence behaviour is somewhat inconclusive although we can see that the tabu search (TS) variants appear to have better asymptotic convergence (to around 5.8-6.2% in excess of the pseudo-optimal solutions) as compared with the iterated greedy (IG) variants which only reach around 8.0-8.2%. However as we shall see below this is probably very dependent on the choice of examples in the test suite. Note also that the scales on the x-axis for these two plots differ by a factor of about 5 and so IG reaches apparent convergence much faster than TS.

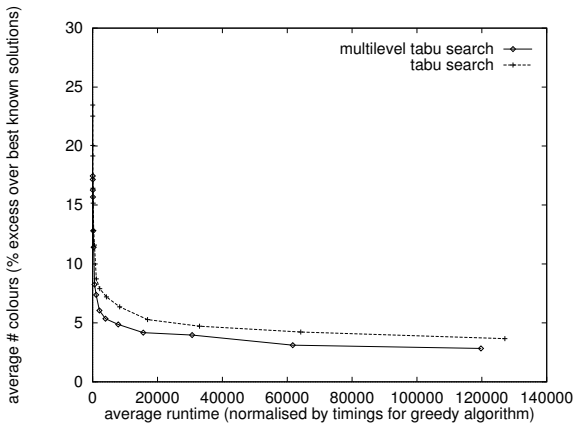
Comparing the multilevel versions with the original algorithms, multilevel tabu search (MLTS) is marginally better than TS, but hardly conclusively. For IG the comparison is not so clear and the multilevel version (MLIG) does not appear to offer any particular advantage. Overall then MLTS appears to provide the best results across the test suite but hardly conclusively.

4.5.1 Density based results

To explore the behaviour further it seems necessary to look at subsets of the test suite to see if more definitive results can be established. Having examined the results in more detail we decided to base the subsets on the density, Δ, of the problem instances. This has the advantage that it is easily calculated (so that, given a new problem instance, an appropriate colouring algorithm can be chosen without recourse to some expensive and possibly complex classification algorithm).

We split the test suite into 3 density classes: low ($0\% \leq \Delta \leq 33\frac{1}{3}\%$), medium ($33\frac{1}{3}\% < \Delta \leq 66\frac{2}{3}\%$) and high ($66\frac{2}{3} < \Delta \leq 100\%$). Table 1 shows the densities; the largest class is low density (with 58 out of 90 instances), followed by medium (23 instances) and then high (with only 9 instances).

It is not immediately clear in which class to place disconnected graphs (or even graphs with isolated ver-



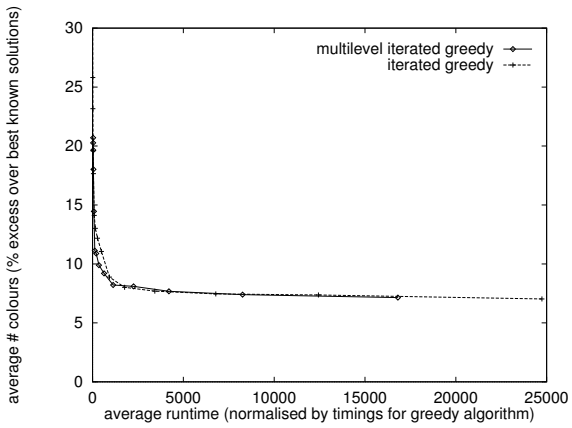
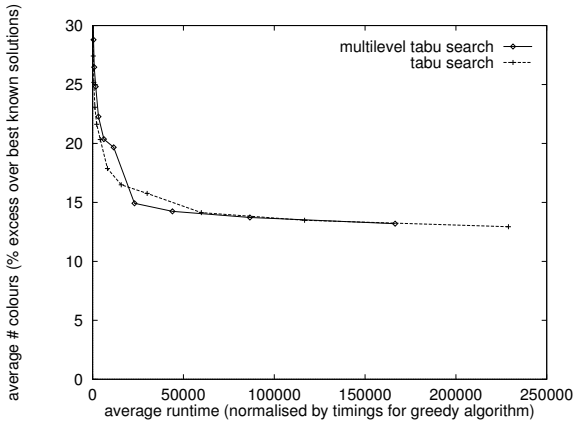


Figure 6: Results on the low-density problem instances



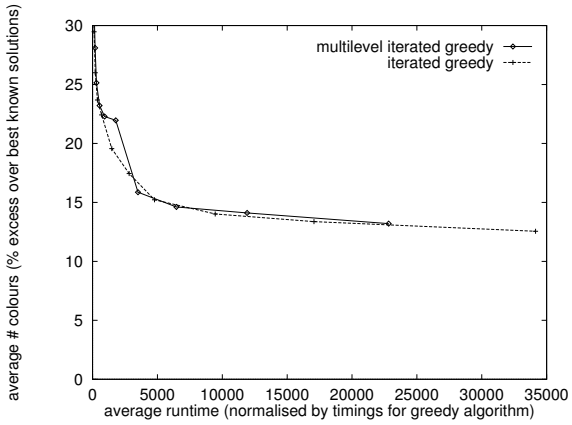
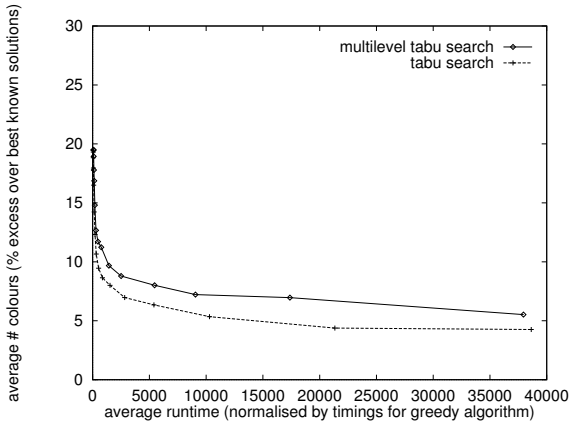


Figure 7: Results on the medium-density problem instances



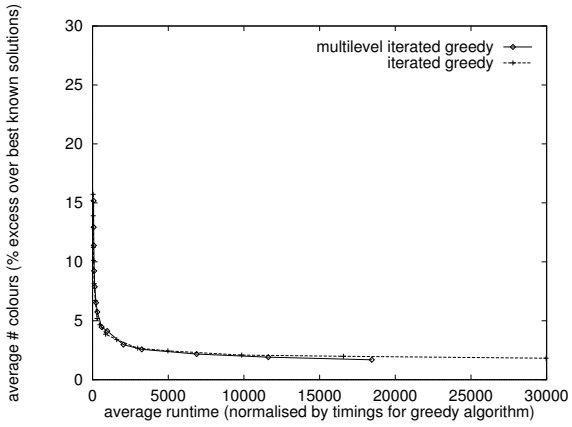


Figure 8: Results on the high-density problem instances

tices) particularly since the multilevel implementation treats each component separately (see §3.1.6). For example a graph with two components, each with $N/2$ vertices, but with one component of low density and one of high, might be classified as medium density. As an even more extreme example, a graph with two complete components each of 2 vertices connected by 1 edge would be classified as low density ($\Delta = 2/6 = 33\frac{1}{3}\%$) even though each component has 100% density. Meanwhile a graph of any density Δ can be turned into a graph with density $\Delta < \epsilon$, for any $\epsilon > 0$, simply by adding sufficient isolated vertices. In fact all the disconnected examples in the test suite (R125.1, homer, huck, miles250, school1_nsh, school1) generally contain one large low-density component with a few small and sometimes high-density components. However, because of the way that the multilevel implementation is sometimes able to ignore the smaller components (see §3.1.6), the smaller denser components in the suite were, without exception, bypassed in the tests and so we include these instances in the low-density subset.

Figures 6-8 show the convergence behaviour on these subclasses and here the results start to become a little more distinctive. They are still inconclusive on the medium-density test cases (Figure 7) although interestingly here both TS and IG variants all seem to be reaching approximately the same asymptotic convergence.

For the high-density examples in Figure 8 the results actually appear to reverse the findings for the whole test suite (Figure 5). Here, for tabu search, the multilevel framework appears to actually hinder colouring and MLTS has poorer convergence than TS. However the performance of both IG and MLIG surpasses that of TS and although the curves overlay each other closely MLIG has very marginally better asymptotic convergence.

It is only for the low-density test cases, Figure 6, that the multilevel versions really start to come into their own. Thus for iterated greedy, although MLIG and IG appear to have approximately the same asymptotic convergence, the multilevel version converges more rapidly. TS meanwhile provides better asymptotic convergence and indeed MLTS is slightly although distinctly better still.

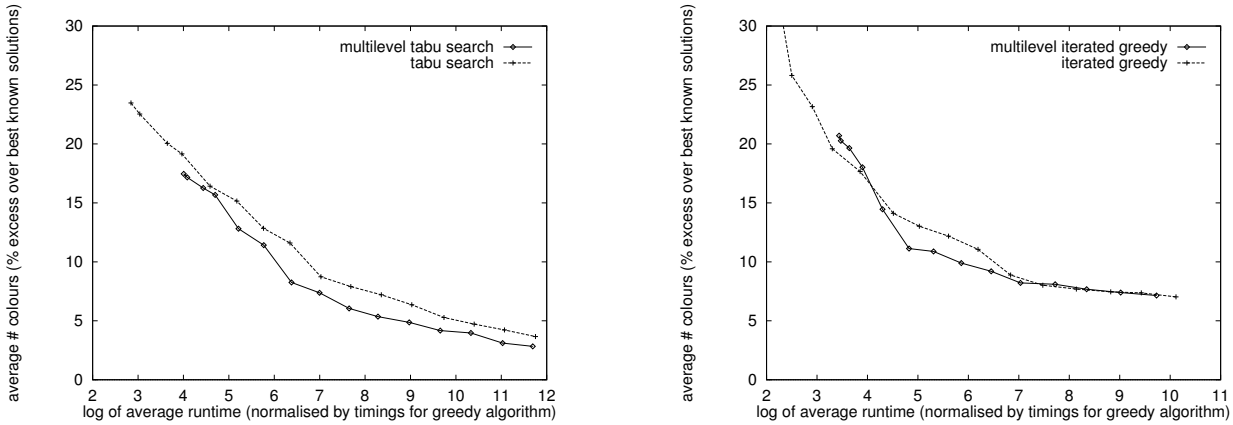


Figure 9: Results on the low-density problem instances using a logarithmic scale

Finally Figure 9 shows the same results as Figure 6 only with the normalised runtime plotted on a logarithmic scale. This gives a clearer picture of the convergence (although less demonstrative) and on the left we see that the MLTS algorithm dominates TS throughout the run. On the right hand side the IG algorithm is initially somewhat faster than MLIG at low search intensities. In fact, this initial runtime overhead is common in multilevel implementations (see e.g. [37]), and MLIG soon shows better performance than IG. Finally, as the convergence rate slows down the curves coincide.

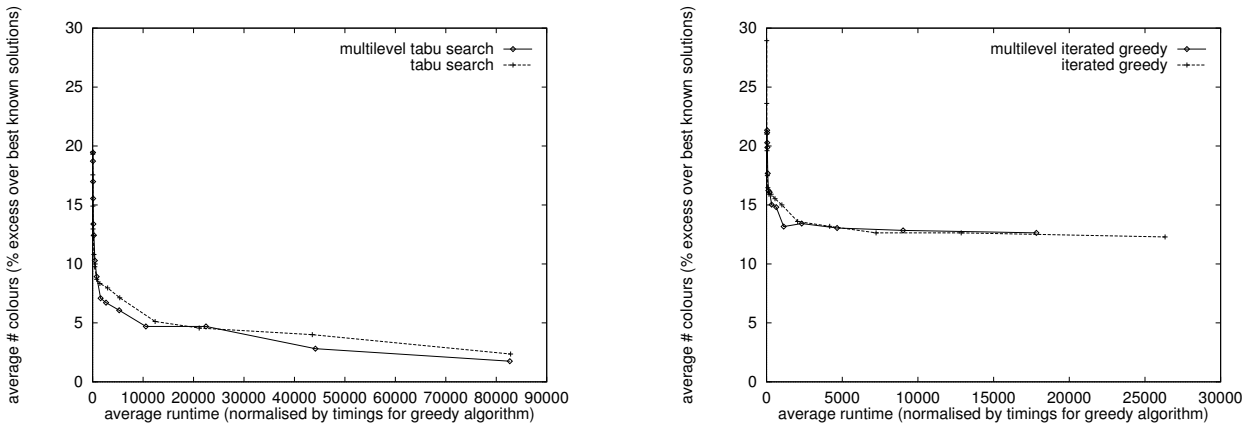


Figure 10: Results on the low-density random problem instances

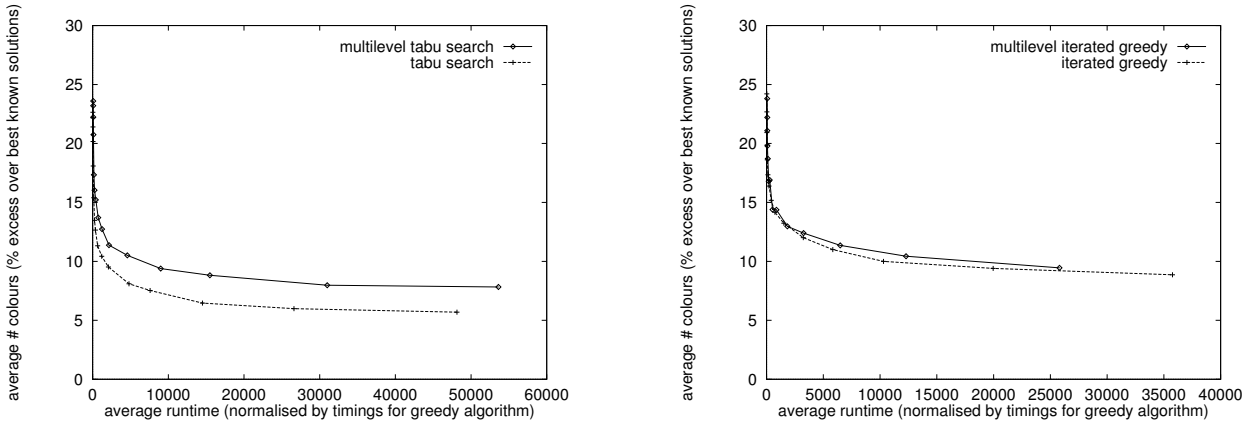


Figure 11: Results on the medium-density random problem instances

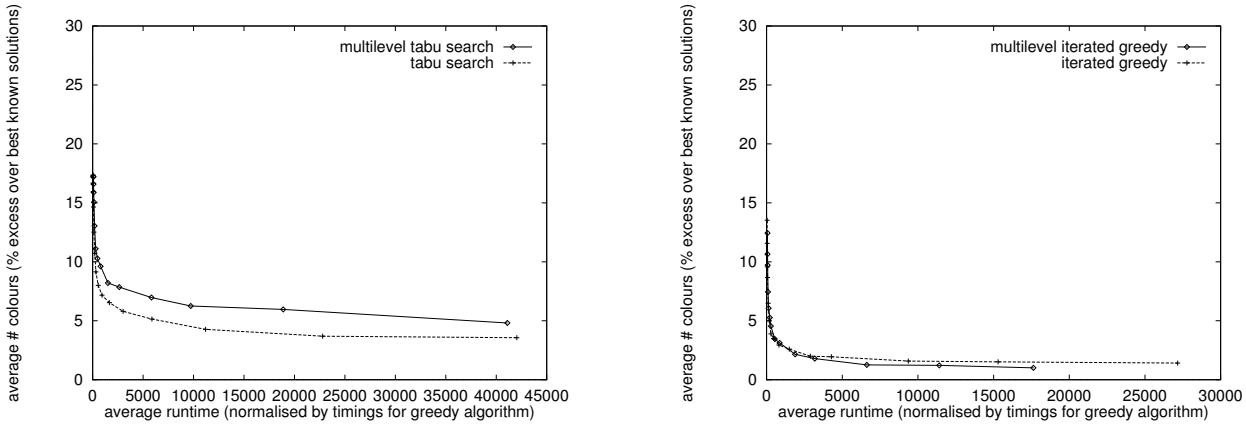


Figure 12: Results on the high-density random problem instances

4.5.2 Density based results on a reduced test set

It could be argued that the density based results actually arise because the subgroups contain different families of problem instances. For example, all the Leighton graphs are of low density, whilst all the flat graphs are in the medium-density class, and so the convergence behaviour might be solely attributable to the fact that the multilevel procedure is better suited to Leighton graphs. To investigate this possible explanation we restricted the subgroups to (randomly generated) graphs for which we could include the same mix of instances in each subgroup. Thus each test set consisted of the R125.d, R250.d, R1000.d, DSJC125.d, DSJC250.d, DSJC500.d, DSCJ1000.d & DSJR500.d examples with $d = 1$ for the low-density reduced test set, $d = 5$ for the medium and $d = 9$ or $d =$ 1c for the high-density set.

The results for these restricted test sets are shown in Figures 10-12. Generally they confirm the results established above with the one exception that, for the medium-density test cases, the algorithm of choice is no longer inconclusive and TS is the clear winner. However, again for the low-density instances the multilevel framework generally enhances the convergence whilst for the high-density problems IG and MLIG outperform the TS variant and have little to choose between them although MLIG again has marginally better asymptotic convergence.

4.5.3 Conclusions

The broad conclusions that we draw from these results are that for medium-density problems the multilevel framework does not appear to offer any improvement to the convergence of tabu search (TS) or the iterated greedy (IG) algorithm. Indeed it may even hinder their convergence. Further, at least for the examples considered here, TS and IG offer similar asymptotic convergence, with TS being slightly better whilst IG is considerably faster. For the high-density problems IG and MLIG appear to offer the best asymptotic convergence and although the multilevel framework appears to enhance this slightly the difference is very marginal. However for the low-density problem instances, the multilevel versions do appear to either speed up (MLIG) or improve (MLTS) the convergence and for such instances MLTS is the clear winner.

The reasons behind this sort of convergence behaviour are not immediately obvious but we can certainly speculate. The fact that tabu search can concentrate on resolving colouring conflicts in specific parts of the graph may account for both TS and MLTS performing well on low-density instances where the conflicts may be confined to a small part of the graph. This may also account for the fact that tabu search does not perform so well on high-density examples as it unable to take a global view of which conflicts it should attempt to resolve. As a contrast, by permuting the colour classes repeatedly, the iterated greedy algorithm attempts to resolve different conflicts at each step.

With regard to the multilevel paradigm, it is somewhat disappointing that its ability to enhance conclusively the convergence behaviour of the local search algorithms is apparently restricted to low-density examples. However this does seem to be in line with a general trend established in other problem areas. For example, in the case of the travelling salesman problem, the multilevel approach appears to be better at aiding examples where the points are gathered into several clusters so that the density is highly variable across the problem instance with large sparse regions, rather then those instances for which the points are uniformly distributed, [35].

Furthermore, with the colouring results derived above in mind, we revisited the graph partitioning problem and tried the same sort of testing using multilevel and single-level versions of a Kernighan-Lin (KL) algorithm to partition the colouring test suite listed in Table 1. Typically multilevel partitioning schemes have been very successfully used for large sparse (very low-density) examples, usually computational meshes, e.g. [14, 20, 38]. It was therefore somewhat of a surprise to see the same sort of density-dependent results repeated for multilevel partitioning. Thus for the low-density examples the multilevel KL algorithm achieved better convergence than KL as expected but, in a reverse of multilevel partitioning performance trends established to date, appeared to be outperformed by KL on the medium-density examples. Details of these experiments can be found in [37].

Perhaps a more disturbing feature of the colouring results is the fact that multilevel refinement actually appears to hinder the tabu search on medium and high-density examples. For a given intensity, the optimisation on the final level of MLTS is exactly the same as the optimisation for TS albeit with a different initial colouring. The worst behaviour we might expect then is a similar asymptotic convergence, but with MLTS taking much longer (since it has the additional cost of graph contraction plus the cost of refinement on all the coarsened levels). However, it is not even as good as that and we suspect that this may occur because poor matching succeeds in pulling the solution into a basin of attraction in the search space (see also §5.3).

5 Summary and Further Work

We have applied the multilevel framework to the graph colouring problem and in particular two refinement strategies, tabu search and iterated greedy. Tests on a large suite of problem instances indicate that for low-density test cases (up to around 30% edge density) the multilevel paradigm can either speed up (iterated greedy) or even improve (tabu search) the asymptotic convergence although the results are somewhat disappointing for medium and high-density examples. This augments existing evidence (e.g. [37]) that, although the multilevel framework cannot be considered as a panacea for combinatorial optimisation problems, it can provide a useful addition to the combinatorial optimisation toolkit. Furthermore Leighton says, with reference to scheduling, that 'for most large-scale practical applications, the edge density of the graphs to be colored is generally small', [23], and so multilevel colouring may be more applicable than the general case would suggest.

An obvious subject for further work would be the use of different refinement strategies such as simulated annealing or even genetic/evolutionary algorithms. Beyond this suggestions, some more specific topics for the possible enhancement of the multilevel colouring algorithms are listed below.

5.1 Sparsification

Perhaps the most disappointing feature of the multilevel colouring algorithm, at least in the manifestation described here, was the inability to aid the solution of medium-density, and to a certain extent, high-density graphs. Clearly of great interest would be any technique for overcoming this difficultly and one way to achieve this might be through sparsification. For example, we could certainly reduce the problem size by picking one or more independent sets S_i from G and removing the vertices in S_i plus all edges incident on vertices in S_i. Whether or not this would result in a sparsification of the problem would depend on the relative number of vertices and edges removed but it seems quite promising as a technique. It also fits in with the hybrid scheme successfully used by Hertz & DeWerra, [15], and by Fleurent & Ferland, [10], who shrink the number of graph vertices by removing a number of maximal independent sets prior to using tabu search-based colouring on the remainder of the graph.

Unfortunately, having removed the independent sets, their vertices are no longer available for the multilevel and local search algorithms to work with. However it is not difficult to imagine an iterated procedure which removes one or more independent sets, S_i, uses multilevel techniques on the remainder of the graph, replaces S_i and then repeats with a different choice of S_i at each iteration. Indeed after the first iteration the procedure is easier since the colour classes that have been found already comprise independent sets and there is no requirement to explicitly search for them.

5.2 Iterated multilevel

The above procedure is in fact a special case of what we refer to as an iterated multilevel algorithm. Recall from the introduction that multigrid solution is one of the best known multilevel techniques and within this

field it is usual for algorithms to traverse both up and down the hierarchy of problems whilst refining the solution. In an echo of this we can iterate the multilevel procedure by using repeated coarsening/refinement loops. However we can enhance the procedure somewhat by using an existing colouring, found in a previous iteration, to create a solution based coarsening (see §3.1.7) and construct a new hierarchy of graphs. Indeed we are required to do so if we do not wish to throw away the existing colouring. Recall from §3.1.7 that such a process guarantees not to find a worse colouring than the initial one. However, if the matching is done randomly (within the restriction that matching must be between vertices of the same colour) each iteration is very likely to give a different hierarchy of graphs to previous iterations and hence give the possibility for the refinement algorithm to visit different solutions in the search space. Although we have not yet made any serious tests of this procedure for colouring, a similar technique is frequently used in graph partitioning for dynamic load-balancing, e.g. [31], and has also been used to find very high quality partitions, [37].

5.3 Advanced matching

Another approach which might improve the results for medium/high-density problems would be the use of better (although possibly more expensive) matching algorithms. It seems possible that medium-density test cases are more challenging for the matching because of the large number of possible vertices to match with and the difficulty of choosing between them. One way of addressing this might be with a dynamic list which reorders the set of unmatched vertices after every match is made (as opposed to the static list which is in use at present). This echoes the sort of techniques used often used in well known colouring schemes such as RLF, [23].

A second possibility might be the sort of greedy matching scheme proposed by Monien *et al.* for graph partitioning, [28]. In the current scheme (§3.1.1) a vertex v_0 picks a preferred candidate v_1 and they are matched. However, v_1 may have available matches which are much more advantageous than v_0. Hence, in the scheme of Monien *et al.*, rather than a match being made at this point, v_1 then picks its preferred match and the process is repeated until a pair of vertices is generated that mutually select each other as a match. Tie-breaking of matches by random selection is forced to be commutative to prevent infinite loops.

5.4 Weighting

A final possibility for further investigation would be to use the weighting implicit in the coarsened levels of the graph (see §3.1.2). In fact for the graph partitioning problem it is essential to use at least the edge weights as they represent the number of edges in the original problem and hence partitioning the coarsened graphs without the weights does not correctly represent the problem. However in graph colouring no such imperative exists and an unweighted coarsened graph does correctly represent the problem, albeit in a restricted form. Nonetheless, both matching and refinement processes frequently have to make decisions about prioritising vertices for certain operations and it may be that using weights can aid this choice. Thus if we seek to maximise the size of a set (e.g. an independent set) it would be better to pick the heaviest weighted vertex from a set of candidates with all other priorities equal. We have not tested this sort of weighting but we believe it might have some effect, although probably not a major one, on the performance of the multilevel algorithm.

Acknowledgements. The author wishes to gratefully acknowledge Joe Culberson for making available the source code for his iterated greedy and tabu search algorithms. The author also wishes to thank Michael Trick making most of the problem instances available and David Joslin & David Clements for supplying the rest of them.

References

[1] S. T. Barnard and H. D. Simon. A Fast Multilevel Implementation of Recursive Spectral Bisection for Partitioning Unstructured Problems. *Concurrency: Practice & Experience*, 6(2):101–117, 1994.

[2] R. Battiti, A. Bertossi, and A. Cappelletti. Multilevel Reactive Tabu Search for Graph Partitioning. Preprint UTM 554, Dip. Mat., Univ. Trento, Italy, 1999.

[3] N. Christofides. *Graph Theory, an Algorithmic Approach*. Academic Press, London, 1975.

[4] T. F. Coleman and J. J. Moré. Estimation of sparse Jacobian matrices and graph coloring problems. *SIAM J. Numer. Anal.*, 20(1):187–209, 1983.

[5] J. Culberson. Graph Coloring Programs Manual. `http://www.cs.ualberta.ca/~joe/Coloring/Colorsrc/manual.html`.

[6] J. C. Culberson. Iterated Greedy Graph Coloring and the Difficulty Landscape. Tech. Rep. TR 92-07, Dept. Comp. Sci., Univ. Alberta, Edmonton, Alberta T6G 2H1, Canada, 1992.

[7] J. C. Culberson, A. Beacham, and D. Papp. Hiding our colors. In *CP'95 Workshop on Studying & Solving Really Hard Problems*, pages 31–42, Cassis, France, September 1995.

[8] J. C. Culberson and F. Luo. Exploring the k-colorable Landscape with Iterated Greedy. In D. S. Johnson and M. A. Trick, editors, *Cliques, Coloring, and Satisfiability*, volume 26 of *DIMACS*, pages 245–284. AMS, Providence RI, 1996.

[9] D. de Werra. An Introduction to Timetabling. *Eur. J. Oper. Res.*, 19:151–162, 1985.

[10] C. Fleurent and J. A. Ferland. Object-Oriented Implementation of Heuristic Search Methods for Graph Coloring, Maximum Clique and Satisfiability. In D. S. Johnson and M. A. Trick, editors, *Cliques, Coloring, and Satisfiability*, volume 26 of *DIMACS*, pages 619–652. AMS, Providence RI, 1996.

[11] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman, San Francisco, 1979.

[12] F. Glover. Future Paths for Integer Programming and Links to Artificial Intelligence. *Comput. Oper. Res.*, 13:533–549, 1986.

[13] F. Glover, M. Parker, and Jennifer Ryan. Coloring by tabu branch and bound. In D. S. Johnson and M. A. Trick, editors, *Cliques, Coloring, and Satisfiability*, volume 26 of *DIMACS*, pages 285–307. AMS, Providence RI, 1996.

[14] B. Hendrickson and R. Leland. A Multilevel Algorithm for Partitioning Graphs. In S. Karin, editor, *Proc. Supercomputing '95, San Diego*. ACM Press, New York, NY 10036, 1995.

[15] A. Hertz and D. de Werra. Using Tabu Search Techniques for Graph Coloring. *Computing*, 39:345–351, 1987.

[16] D. S. Johnson. Approximation Algorithms for Combinatorial Problems. *J. Comput. Syst. Sci.*, 9:256–278, 1974.

[17] D. S. Johnson, C. R. Aragon, L. A. McGeoch, and C. Schevon. Optimization by Simulated Annealing: Part II, Graph Coloring and Number Partitioning. *Oper. Res.*, 39(3):378–406, 1991.

[18] D. S. Johnson and M. A. Trick, editors. *Cliques, Coloring, and Satisfiability*, volume 26 of *DIMACS*. AMS, Providence RI, 1996.

[19] D. E. Joslin and D. P. Clements. "Squeaky Wheel" Optimization. *J. Artificial Intelligence Res.*, 10:353–373, 1999.

[20] G. Karypis and V. Kumar. A Fast and High Quality Multilevel Scheme for Partitioning Irregular Graphs. *SIAM J. Sci. Comput.*, 20(1):359–392, 1998.

[21] A. Kaveh and H. A. Rahimi Bondarabady. A Hybrid Graph-Genetic Method for Domain Decomposition. In B. H. V. Topping, editor, *Computational Engineering using Metaphors from Nature*, pages 127–134. Civil-Comp Press, Edinburgh, 2000. (Proc. Engrg. Comput. Technology, Leuven, Belgium, 2000).

[22] B. W. Kernighan and S. Lin. An Efficient Heuristic for Partitioning Graphs. *Bell Syst. Tech. J.*, 49:291–308, 1970.

[23] F. T. Leighton. A Graph Colouring Algorithm for Large Scheduling Problems. *J. Res. National Bureau Standards*, 84:489–503, 1979.

[24] G. Lewandowski and A. Condon. Experiments with Parallel Graph Coloring and Applications of Graph Coloring. In D. S. Johnson and M. A. Trick, editors, *Cliques, Coloring, and Satisfiability*, volume 26 of *DIMACS*, pages 309–334. AMS, Providence RI, 1996.

[25] C. Lund and M. Yannakakis. On the Hardness of Approximating Minimization Problems. *J. ACM*, 41(5):960–981, 1994.

[26] O. C. Martin and S. W. Otto. Combining Simulated Annealing with Local Search Heuristics. Tech. Rep. CSE-94-016, Oregon Grad. Inst. Sci. Tech., September 1993.

[27] D. W. Matula, G. Marble, and J. D. Isaacson. Graph Coloring Algorithms. In R. C. Read, editor, *Graph Theory and Computing*, pages 109–122. Academic Press, New York, 1972.

[28] B. Monien, R. Preis, and R. Diekmann. Quality matching and local improvement for multilevel graph-partitioning. *Parallel Comput.*, 26(12):1605–1634, 2000.

[29] C. Morgenstern. Distributed Coloration Neighborhood Search. In D. S. Johnson and M. A. Trick, editors, *Cliques, Coloring, and Satisfiability*, volume 26 of *DIMACS*, pages 335–357. AMS, Providence RI, 1996.

[30] P. Moscato. On evolution, search, optimization, genetic algorithms and martial arts: Towards memetic algorithms. Technical Report C3P 826, CalTech, Pasadena, CA, 1989.

[31] K. Schloegel, G. Karypis, and V. Kumar. Multilevel Diffusion Schemes for Repartitioning of Adaptive Meshes. *J. Parallel Distrib. Comput.*, 47(2):109–124, 1997.

[32] E. C. Sewell. An Improved Algorithm for Exact Graph Coloring. In D. S. Johnson and M. A. Trick, editors, *Cliques, Coloring, and Satisfiability*, volume 26 of *DIMACS*, pages 359–373. AMS, Providence RI, 1996.

[33] S.-H. Teng. Coarsening, sampling, and smoothing: Elements of the multilevel method. In M. T. Heath *et al.*, editor, *Algorithms for Parallel Processing*, volume 105 of *IMA Volumes in Mathematics and its Applications*, pages 247–276. Springer-Verlag, New York, 1999.

[34] D. Vanderstraeten, C. Farhat, P. S. Chen, R. Keunings, and O. Zone. A Retrofit Based Methodology for the Fast Generation and Optimization of Large-Scale Mesh Partitions: Beyond the Minimum Interface Size Criterion. *Comput. Methods Appl. Mech. Engrg.*, 133:25–45, 1996.

[35] C. Walshaw. A Multilevel Approach to the Travelling Salesman Problem. Accepted for *Oper. Res.*, (originally published as Univ. Greenwich Tech. Rep. 00/IM/63), 2000.

[36] C. Walshaw. A Multilevel Algorithm for Force-Directed Graph Drawing. In J. Marks, editor, *Graph Drawing, 8th Intl. Symp. GD 2000*, volume 1984 of *LNCS*, pages 171–182. Springer, Berlin, 2001.

[37] C. Walshaw. Multilevel Combinatorial Optimisation. To appear in Proc. Metaheuristics Intl. Conf. (MIC 2001), Porto, Portugal, 2001.

[38] C. Walshaw and M. Cross. Mesh Partitioning: a Multilevel Balancing and Refinement Algorithm. *SIAM J. Sci. Comput.*, 22(1):63–80, 2000. (originally published as Univ. Greenwich Tech. Rep. 98/IM/35).

Table 1: The test suite of problem instances

problem	size			degree			problem	size			degree		
instance p	$\|V\|$	$\|E\|$	$\Delta\%$	max	min	$\chi(p)$	instance p	$\|V\|$	$\|E\|$	$\Delta\%$	max	min	$\chi(p)$
C2000.5	2000	999836	50.02	1074	919	$\leq$153	le450_5a	450	5714	5.66	42	13	5
C4000.5	4000	4000268	50.02	2123	1895	$\leq$280	le450_5b	450	5734	5.68	42	12	5
DSJC125.1	125	736	9.50	23	5	$\leq$5	le450_5c	450	9803	9.70	66	27	5
DSJC250.1	250	3218	10.34	38	13	$\leq$8	le450_5d	450	9757	9.66	68	29	5
DSJC500.1	500	12458	9.99	68	34	$\leq$12	le450_15a	450	8168	8.09	99	2	15
DSJC1000.1	1000	49629	9.94	127	68	$\leq$22	le450_15b	450	8169	8.09	94	1	15
DSJC125.5	125	3891	50.21	75	51	$\leq$17	le450_15c	450	16680	16.51	139	18	15
DSJC250.5	250	15668	50.34	147	101	$\leq$28	le450_15d	450	16750	16.58	138	18	15
DSJC500.5	500	62624	50.20	286	220	$\leq$49	le450_25a	450	8260	8.18	128	2	25
DSJC1000.5	1000	249826	50.02	551	447	$\leq$84	le450_25b	450	8263	8.18	111	2	25
DSJC125.9	125	6961	89.82	120	103	$\leq$44	le450_25c	450	17343	17.17	179	7	25
DSJC250.9	250	27897	89.63	234	207	$\leq$72	le450_25d	450	17425	17.25	157	11	25
DSJC500.9	500	112437	90.13	471	430	$\leq$129	miles250	128	387	4.76	16	0	8
DSJC1000.9	1000	449449	89.98	924	870	$\leq$234	miles500	128	1170	14.39	38	3	20
R125.1	125	209	2.70	8	0	5	miles750	128	2113	26.00	64	6	31
R250.1	250	867	2.79	13	1	8	miles1000	128	3216	39.57	86	13	42
R1000.1	1000	14378	2.88	49	10	20	miles1500	128	5198	63.95	106	28	73
R125.5	125	3838	49.52	99	23	36	mulsol.i.1	197	3925	20.33	121	0	49
R250.5	250	14849	47.71	191	53	65	mulsol.i.2	188	3885	22.10	156	0	31
R1000.5	1000	238267	47.70	781	193	$\leq$238	mulsol.i.3	184	3916	23.26	157	0	31
R125.1c	125	7501	96.79	124	113	46	mulsol.i.4	185	3946	23.18	158	0	31
R250.1c	250	30227	97.11	249	234	64	mulsol.i.5	186	3973	23.09	159	0	31
R1000.1c	1000	485090	97.12	991	943	$\leq$98	myciel3	11	20	36.36	5	3	4
DSJR500.1	500	3555	2.85	25	4	12	myciel4	23	71	28.06	11	4	5
DSJR500.5	500	58862	47.18	388	103	$\leq$123	myciel5	47	236	21.83	23	5	6
DSJR500.1c	500	121275	97.21	497	473	$\leq$85	myciel6	95	755	16.91	47	6	7
anna	138	493	5.22	71	1	11	myciel7	191	2360	13.01	95	7	8
david	87	406	10.85	82	1	11	queen5_5	25	160	53.33	16	12	5
homer	561	1628	1.04	99	0	13	queen6_6	36	290	46.03	19	15	7
huck	74	301	11.14	53	1	11	queen7_7	49	476	40.48	24	18	7
jean	80	254	8.04	36	0	10	queen8_8	64	728	36.11	27	21	9
flat300_20_0	300	21375	47.66	160	127	20	queen8_12	96	1368	30.00	32	25	12
flat300_26_0	300	21633	48.23	158	123	26	queen9_9	81	1056	32.59	32	24	10
flat300_28_0	300	21695	48.37	162	130	28	queen10_10	100	1470	29.70	35	27	$\leq$11
flat1000_50_0	1000	245000	49.05	520	459	50	queen11_11	121	1980	27.27	40	30	11
flat1000_60_0	1000	245830	49.22	524	457	60	queen12_12	144	2596	25.21	43	33	$\leq$13
flat1000_76_0	1000	246708	49.39	532	455	76	queen13_13	169	3328	23.44	48	36	13
fpsol2.i.1	496	11654	9.49	252	0	65	queen14_14	196	4186	21.90	51	39	$\leq$15
fpsol2.i.2	451	8691	8.56	346	0	30	queen15_15	225	5180	20.56	56	42	$\leq$17
fpsol2.i.3	425	8688	9.64	346	0	30	queen16_16	256	6320	19.36	59	45	$\leq$18
games120	120	638	8.94	13	7	9	school1	385	19095	25.83	282	1	14
inithx.i.1	864	18707	5.02	502	0	54	school1_nsh	352	14612	23.65	232	1	14
inithx.i.2	645	13979	6.73	541	0	31	zeroin.i.1	211	4100	18.51	111	0	49
inithx.i.3	621	13969	7.26	542	0	31	zeroin.i.2	211	3541	15.98	140	0	30
latin_square_10	900	307350	75.97	683	683	$\leq$98	zeroin.i.3	206	3540	16.77	140	0	30

Table 2: Colouring solution quality (minimum, Q_m, and average, $\overline{Q}$) and average runtime in seconds, $\overline{T}$, for single-level & multilevel iterated greedy (IG & MLIG) over 3 random seeds

problem	MLIG ($\lambda = 1,024$)			IG (λ = 1,024)			problem	MLIG (λ = 1,024)			IG (λ = 1,024)		
instance p	Q_m	$\overline{Q}$	$\overline{T}$	Q_m	$\overline{Q}$	$\overline{T}$	instance p	Q_m	$\overline{Q}$	$\overline{T}$	Q_m	$\overline{Q}$	$\overline{T}$
C2000.5	193	194.7	197.87	193	194.3	139.41	le450_5a	5	5.0	3.10	5	5.0	4.64
C4000.5	354	354.3	1162.86	355	356.0	571.66	le450_5b	5	5.0	3.66	5	5.0	4.44
DSJC125.1	6	6.0	0.61	6	6.0	0.64	le450_5c	5	5.0	2.42	5	5.0	2.09
DSJC250.1	10	10.0	2.19	10	10.0	2.10	le450_5d	5	5.0	2.58	5	5.0	1.92
DSJC500.1	16	16.0	8.52	16	16.3	4.82	le450_15a	18	18.0	6.00	18	18.0	4.08
DSJC1000.1	27	27.7	18.56	27	27.0	23.90	le450_15b	18	18.0	4.56	18	18.0	4.01
DSJC125.5	19	19.0	1.63	19	19.3	0.78	le450_15c	25	25.3	8.75	25	25.3	5.34
DSJC250.5	33	33.7	3.08	33	33.7	1.88	le450_15d	25	25.3	7.12	25	25.7	6.22
DSJC500.5	58	59.0	13.21	59	59.0	10.02	le450_25a	25	25.0	5.97	25	25.0	3.61
DSJC1000.5	107	107.7	45.77	105	105.0	69.62	le450_25b	25	25.0	7.46	25	25.0	4.23
DSJC125.9	44	45.0	1.11	44	45.0	1.27	le450_25c	30	30.7	9.01	30	30.3	7.19
DSJC250.9	74	74.7	6.80	74	75.3	5.05	le450_25d	30	30.7	6.98	30	30.0	7.63
DSJC500.9	133	133.7	20.95	133	133.7	21.23	miles250	8	8.0	0.49	8	8.0	0.50
DSJC1000.9	241	244.3	111.33	242	244.3	115.44	miles500	20	20.0	1.07	20	20.0	0.85
R125.1	5	5.0	0.15	5	5.0	0.37	miles750	31	31.0	1.01	31	31.0	0.87
R250.1	8	8.0	2.40	8	8.0	1.35	miles1000	42	42.0	1.48	42	42.0	1.27
R1000.1	20	20.7	24.83	21	21.0	15.48	miles1500	73	73.0	1.48	73	73.0	1.22
R125.5	36	36.7	0.98	36	36.3	1.53	mulsol.i.1	49	49.0	1.77	49	49.0	1.69
R250.5	68	68.3	4.21	68	68.0	5.03	mulsol.i.2	31	31.0	1.07	31	31.0	0.77
R1000.5	253	253.3	89.09	252	253.0	96.26	mulsol.i.3	31	31.0	1.06	31	31.0	0.81
R125.1c	46	46.0	0.87	46	46.0	0.63	mulsol.i.4	31	31.0	1.08	31	31.0	0.81
R250.1c	64	64.0	3.01	64	64.0	2.75	mulsol.i.5	31	31.0	1.06	31	31.0	0.84
R1000.1c	98	98.3	29.04	98	99.0	18.86	myciel3	4	4.0	0.02	4	4.0	0.02
DSJR500.1	12	12.0	6.63	12	12.0	6.94	myciel4	5	5.0	0.06	5	5.0	0.04
DSJR500.5	129	129.3	16.60	128	129.0	16.90	myciel5	6	6.0	0.17	6	6.0	0.10
DSJR500.1c	85	85.0	7.65	85	85.0	7.44	myciel6	7	7.0	0.35	7	7.0	0.38
anna	11	11.0	0.74	11	11.0	0.65	myciel7	8	8.0	1.22	8	8.0	0.84
david	11	11.0	0.49	11	11.0	0.27	queen5_5	5	5.0	0.05	5	5.0	0.04
homer	13	13.0	6.22	13	13.0	9.20	queen6_6	7	7.7	0.11	7	7.7	0.09
huck	11	11.0	0.25	11	11.0	0.18	queen7_7	7	7.0	0.13	7	7.3	0.11
jean	10	10.0	0.36	10	10.0	0.27	queen8_8	10	10.0	0.24	10	10.0	0.21
flat300_20_0	20	20.0	2.42	20	20.0	1.61	queen8_12	13	13.0	0.44	13	13.0	0.35
flat300_26_0	37	37.7	5.37	37	37.7	3.30	queen9_9	11	11.0	0.44	11	11.0	0.23
flat300_28_0	36	37.0	5.77	37	37.3	3.97	queen10_10	12	12.7	0.66	12	12.3	0.53
flat1000_50_0	50	50.0	102.60	50	67.0	68.67	queen11_11	13	13.7	0.82	14	14.0	0.59
flat1000_60_0	104	104.7	60.41	103	104.7	51.41	queen12_12	15	15.0	1.54	15	15.0	0.69
flat1000_76_0	106	106.3	47.67	105	105.7	42.33	queen13_13	16	16.0	1.78	16	16.0	1.32
fpsol2.i.1	65	65.0	6.19	65	65.0	6.67	queen14_14	18	18.0	1.52	17	17.7	1.19
fpsol2.i.2	30	30.0	4.64	30	30.0	3.15	queen15_15	19	19.0	2.75	18	18.7	1.32
fpsol2.i.3	30	30.0	4.24	30	30.0	3.30	queen16_16	20	20.0	3.27	20	20.0	2.26
games120	9	9.0	0.61	9	9.0	0.53	school1	14	14.0	3.50	14	14.0	1.83
inithx.i.1	54	54.0	16.93	54	54.0	14.95	school1_nsh	14	14.0	4.15	14	14.0	2.18
inithx.i.2	31	31.0	10.10	31	31.0	8.58	zeroin.i.1	49	49.0	1.76	49	49.0	1.75
inithx.i.3	31	31.0	11.59	31	31.0	7.58	zeroin.i.2	30	30.0	1.83	30	30.0	1.23
latin_square_10	106	106.7	32.33	105	106.0	31.41	zeroin.i.3	30	30.0	1.43	30	30.0	1.07

Table 3: Colouring solution quality (minimum, Q_m, and average, $\overline{Q}$) and average runtime in seconds, $\overline{T}$, for single-level & multilevel tabu search (TS & MLTS) over 3 random seeds

problem	MLTS ($\lambda = 2{,}048$)			TS ($\lambda = 4{,}096$)			problem	MLTS ($\lambda = 2{,}048$)			TS ($\lambda = 4{,}096$)		
instance p	Q_m	$\overline{Q}$	$\overline{T}$	Q_m	$\overline{Q}$	$\overline{T}$	instance p	Q_m	$\overline{Q}$	$\overline{T}$	Q_m	$\overline{Q}$	$\overline{T}$
C2000.5	179	179.3	50.93	178	178.0	32.07	le450_5a	5	5.3	12.82	6	6.3	17.51
C4000.5	331	334.0	169.31	332	334.0	72.42	le450_5b	5	5.7	11.20	6	6.7	27.70
DSJC125.1	5	5.3	8.09	5	5.0	13.46	le450_5c	5	5.0	13.75	5	5.3	14.93
DSJC250.1	9	9.0	11.05	9	9.0	8.96	le450_5d	5	5.0	15.18	5	6.0	14.28
DSJC500.1	13	13.7	18.25	13	13.7	20.29	le450_15a	16	16.0	14.96	16	16.0	14.74
DSJC1000.1	23	23.0	27.49	23	23.0	20.55	le450_15b	16	16.0	14.96	16	16.0	13.12
DSJC125.5	18	18.0	7.27	18	18.0	7.65	le450_15c	22	22.0	16.81	21	21.7	20.86
DSJC250.5	30	30.7	9.21	30	30.7	7.42	le450_15d	22	22.3	13.30	22	22.7	12.77
DSJC500.5	54	54.3	15.68	54	54.0	17.55	le450_25a	25	25.0	11.92	25	25.0	4.88
DSJC1000.5	97	97.7	26.02	97	97.7	21.16	le450_25b	25	25.0	8.50	25	25.0	5.37
DSJC125.9	44	44.7	5.84	44	44.3	5.60	le450_25c	28	28.0	17.08	27	27.0	17.72
DSJC250.9	75	75.7	5.90	74	74.3	9.29	le450_25d	28	28.3	13.01	27	27.7	14.38
DSJC500.9	136	137.0	12.39	133	134.3	18.13	miles250	8	8.0	5.59	8	8.0	5.41
DSJC1000.9	253	254.0	24.74	249	251.0	20.45	miles500	20	20.0	7.47	20	20.0	6.91
R125.1	5	5.0	5.89	5	5.0	6.00	miles750	31	31.0	13.73	31	31.0	11.37
R250.1	8	8.0	7.40	8	8.0	6.02	miles1000	42	42.3	13.25	42	42.0	13.93
R1000.1	20	20.0	12.18	22	22.0	20.24	miles1500	73	73.0	25.06	73	73.0	49.37
R125.5	38	38.0	7.10	37	37.0	4.18	mulsol.i.1	49	49.0	21.94	49	49.0	28.95
R250.5	70	71.0	17.63	68	68.7	19.13	mulsol.i.2	31	31.0	5.22	31	31.0	3.31
R1000.5	259	259.0	156.39	251	251.3	119.00	mulsol.i.3	31	31.0	5.05	31	31.0	3.30
R125.1c	47	47.7	5.58	46	46.0	8.72	mulsol.i.4	31	31.0	5.14	31	31.0	3.31
R250.1c	67	67.7	11.12	65	65.0	10.10	mulsol.i.5	31	31.0	14.09	31	31.0	18.25
R1000.1c	107	113.3	41.16	114	114.0	17.54	myciel3	4	4.0	3.06	4	4.0	5.70
DSJR500.1	12	12.0	9.37	12	12.0	7.19	myciel4	5	5.0	3.93	5	5.0	4.57
DSJR500.5	134	134.0	49.29	127	128.3	21.67	myciel5	6	6.0	4.33	6	6.0	4.77
DSJR500.1c	90	93.0	24.05	90	91.7	10.31	myciel6	7	7.0	4.96	7	7.0	4.48
anna	11	11.0	6.74	11	11.0	6.27	myciel7	8	8.0	5.98	8	8.0	5.19
david	11	11.0	5.10	11	11.0	4.80	queen5_5	5	5.0	3.42	5	5.0	3.81
homer	13	13.0	14.45	13	13.0	17.38	queen6_6	7	7.0	3.51	7	7.0	3.68
huck	11	11.0	5.33	11	11.0	5.81	queen7_7	7	7.0	4.12	7	7.0	4.09
jean	10	10.0	5.73	10	10.0	5.24	queen8_8	9	9.0	4.88	9	9.0	4.55
flat300_20_0	20	20.0	13.26	20	28.0	13.24	queen8_12	12	12.0	3.71	12	12.0	3.72
flat300_26_0	34	34.0	10.72	34	34.0	10.43	queen9_9	10	10.0	5.06	10	10.0	7.59
flat300_28_0	34	34.0	10.44	33	34.0	12.03	queen10_10	11	11.3	7.70	11	11.7	7.14
flat1000_50_0	95	96.0	22.31	94	95.0	26.72	queen11_11	12	12.7	5.58	12	12.7	8.51
flat1000_60_0	94	95.7	28.91	94	94.7	31.44	queen12_12	14	14.0	6.45	14	14.0	6.02
flat1000_76_0	95	96.7	25.87	96	96.3	19.41	queen13_13	15	15.0	6.79	15	15.0	6.53
fpsol2.i.1	65	65.0	27.16	65	65.0	44.90	queen14_14	16	16.0	6.91	16	16.0	7.18
fpsol2.i.2	30	30.0	18.33	30	30.0	20.49	queen15_15	17	17.0	9.10	17	17.0	10.08
fpsol2.i.3	30	30.0	21.90	30	30.0	19.94	queen16_16	18	18.0	9.42	18	18.0	10.72
games120	9	9.0	4.93	9	9.0	4.18	school1	14	14.0	11.93	14	14.0	9.20
inithx.i.1	54	54.0	29.81	54	54.0	45.49	school1_nsh	14	14.3	12.35	14	14.0	6.95
inithx.i.2	31	31.0	7.10	31	31.0	33.66	zeroin.i.1	49	49.0	14.18	49	49.0	28.62
inithx.i.3	31	31.0	6.23	31	31.0	32.62	zeroin.i.2	30	30.0	19.04	30	30.0	14.31
latin_square_10	113	114.0	20.92	112	113.7	13.84	zeroin.i.3	30	30.0	18.80	30	30.0	15.03