Annals of Operations Research 131, 325–372, 2004

Multilevel Refinement for Combinatorial Optimisation Problems

CHRIS WALSHAW * C.Walshaw@gre.ac.uk
Computing and Mathematical Sciences, University of Greenwich, Old Royal Naval College, Greenwich, London, SE10 9LS, UK

Abstract. We consider the multilevel paradigm and its potential to aid the solution of combinatorial optimisation problems. The multilevel paradigm is a simple one, which involves recursive coarsening to create a hierarchy of approximations to the original problem. An initial solution is found (sometimes for the original problem, sometimes the coarsest) and then iteratively refined at each level. As a general solution strategy, the multilevel paradigm has been in use for many years and has been applied to many problem areas (most notably in the form of multigrid techniques). However, with the exception of the graph partitioning problem, multilevel techniques have not been widely applied to combinatorial optimisation problems. In this paper we address the issue of multilevel refinement for such problems and, with the aid of examples and results in graph partitioning, graph colouring and the travelling salesman problem, make a case for its use as a metaheuristic. The results provide compelling evidence that, although the multilevel framework cannot be considered as a panacea for combinatorial problems, it can provide an extremely useful addition to the combinatorial optimisation toolkit. We also give a possible explanation for the underlying process and extract some generic guidelines for its future use on other combinatorial problems.

Keywords: multilevel refinement, combinatorial optimisation, metaheuristic, graph partitioning, travelling salesman, graph colouring

1. Introduction

In this paper we consider the multilevel paradigm and its potential to aid the solution of combinatorial optimisation problems. 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 paradigm has been in use 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). Overview papers such as (Brandt, 1988; Teng, 1999) attest to its efficacy. However, with the exception of the graph partitioning problem, multilevel techniques have not been widely applied to combinatorial optimisation problems and in this paper we consider their potential benefits. In particular we are interested in the broad class of discrete systems, with a finite but usually exponential number of states, in which the requirement is to find the minimum

* URL: http://staffweb.cms.gre.ac.uk/~c.walshaw

(or maximum) of a cost function (a function that gives a value in $\mathbb{N}$, or sometimes $\mathbb{R}$, to every state). There are many such problems which are known to be NP-hard (Garey and Johnson, 1979) (for example the graph partitioning problem, the travelling salesman problem, the quadratic assignment problem, etc.). In other words a true minimum for the cost function cannot be found in polynomial time and so a heuristic, which will not be able to guarantee finding an optimum solution, must be used. The problem is therefore relaxed to finding a 'good' solution in 'reasonable' time.

Our interest in the multilevel paradigm arose with work in the field of graph partitioning, e.g., (Walshaw and Cross, 2000; Walshaw et al., 1999). It was clear from some of these examples that the multilevel framework was sometimes able to impart a 'global' quality to local search heuristics (see section 2.2 below and (Walshaw, 2001d, section 2.3)). More recently the paradigm was applied to force-directed (FD) graph drawing (which is not a combinatorial problem but shares some of the characteristics). Although FD methods are generally limited to sparse problems, the multilevel framework was able to extend the size of graphs to which they can be applied by several orders of magnitude, again through the 'global' improvement given by the multilevel scheme (Walshaw, 2001a). With the realisation that this global quality might be put to good use for other discrete problems, and a possible explanation of the underlying process (at least for combinatorial optimisation problems), multilevel algorithms were then derived for the travelling salesman problem (Walshaw, 2002), and for graph colouring (Walshaw, 2001b).

In this paper, we aim to draw together some of this work, address the issue of multilevel combinatorial refinement and, with the aid of examples in graph partitioning, graph colouring and the travelling salesman problem, make a case for its use as a metaheuristic. In particular we hope to achieve three objectives:

- Firstly to demonstrate that, for the problems on which it has been tested, the multilevel paradigm when used in combination with a local search strategy can often either accelerate the convergence rate of the local search or even improve the asymptotic convergence in solution quality. Indeed sometimes it appears to even impart a more 'global' quality to the final solution. The evidence here is confined to results from three example problem areas, but within these areas it is compelling. Related work on other problems, although patchy, substantiates this claim.
- Secondly, and more importantly, we shall attempt to explain the underlying process and hypothesise about why this process allows the multilevel strategy to enhance the local search algorithms.
- Finally we shall extract some of the generic principles underlying existing multilevel algorithms and suggest how they might be applied to other combinatorial optimisation problems.

The ultimate aim is not to show that any one specific multilevel scheme dominates for a given problem, but that, if the reader has a favourite local search algorithm for some problem, then it could be well worth considering the implementation of a multilevel version.

1.1. Overview

The rest of the paper is organised as follows. In sections 2–4 we discuss the evidence for the strengths of the multilevel paradigm by describing three existing multilevel implementations and presenting some sample results. Thus in section 2 we describe the widespread use of multilevel techniques as applied to the graph partitioning problem (GPP). The multilevel paradigm has been employed in this field since 1993 and is one of the key ideas that has enabled such high quality solutions to be found so rapidly. In section 3 we then discuss the recent application of the multilevel strategy to the travelling salesman problem (TSP). The TSP is perhaps the most widely studied combinatorial optimisation problem in existence and yet it has been dominated by a single heuristic, the Lin–Kernighan algorithm, and an iterated version of the same for nearly 30 years. Nonetheless the multilevel strategy when used in combination with this heuristic was able to significantly improve on its results. Next, in section 4, we discuss the application, also recent, of the multilevel paradigm to the graph colouring problem (GCP). For this very challenging problem, often cited as one of the most difficult combinatorial problems, the multilevel techniques were not universally successful but for graphs of sparse and low-density were able to improve the convergence behaviour of two different local search algorithms. These results also give an indication of where the multilevel techniques might fail. In section 5 we then suggest the process that underlies the multilevel coarsening and speculate as to the properties that it imparts to the local refinement and which allow it to enhance the results. We also extract some guiding principles that might aid the application of the strategy to other problems. A discussion of related work, including applications of multilevel algorithms to other problems, can be found in section 5.5. Finally we summarise the findings in section 6 and suggest some future research.

Note that this paper brings together results and ideas from a number of sources. In particular the TSP results are drawn from (Walshaw, 2002, 2001c), whilst some of the GCP results appear in (Walshaw, 2001b). In addition, some of the generic multilevel ideas discussed and extended here can be found in (Walshaw, 2002, 2001b) as motivation for multilevel approaches to the TSP and GCP. However the GPP results, and the issues raised about multilevel partitioning for denser instances, are new.

1.2. Notation and definitions

For all three example problems we use graph-based terminology and so it makes sense to define some common notation here. Let $G = G(V, E)$ be an undirected graph of vertices V, with edges E. We use $|.|$ to denote the number of elements in a set, e.g., $|V|$ is the number of vertices, and if the graph has weights (either for the vertices and/or the edges) $\|.\|$ denotes the summed weight of a set of vertices or edges (and as a shorthand $\|x\| = \|\{x\}\|$ denotes the weight of a single object). For a set of vertices $S \subset V$, we use $\Gamma(S)$ to denote the *neighbourhood* of S, the set of vertices adjacent to, but not including, vertices in S, i.e. $\Gamma(S) = \{v \in V - S$: there exists $(u, v) \in E$ with $u \in S\}$ and as a shorthand we write $\Gamma(v) = \Gamma(\{v\})$ to denote the neighbourhood of a single vertex v.

A *complete* graph is one for which every vertex is adjacent to every other (and so $\Gamma(v) = V - \{v\}$ for all $v \in V$). We then define the *edge density*, Δ, as $\Delta = 2|E|/|V|(|V| - 1)$ so that a complete graph with $|V|(|V| - 1)/2$ edges has density 1.0 or 100% density. A subset of vertices $S \subset V$ forms a *connected component* if each vertex in S can be reached from every other vertex in S by traversing paths of edges and if $\Gamma(S)$ is empty ($\Gamma(S) = \emptyset$). As a special case, an *isolated* vertex is one which has no neighbours, i.e. $\Gamma(v) = \emptyset$, and hence forms a *trivial component*. The graph $G(V, E)$ is *connected* if V is a connected component; a *disconnected* graph, therefore, is one with two or more components and a *proper disconnected* graph is one two or more non-trivial components.

1.3. Experimental methodology

Below we outline results in 3 problem areas: graph partitioning, graph colouring and the travelling salesman problem. In each case we wish to demonstrate that given a (single-level) local search strategy for the problem (or potentially even a more complex scheme such as a genetic algorithm) then a multilevel version can improve the convergence of the local search (either the convergence rate or even the asymptotic convergence in solution quality). In that sense our aim is not necessarily to show that a multilevel implementation is better than any other strategy in that field, although we pick local search algorithms which are either dominant over or at least competitive with other solution methods for the problem in question.

Typically such 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 time interval, but perhaps more commonly it is the number of iterations of some outer loop of the algorithm, either in absolute terms or in terms of the number of failures to achieve a better solution (e.g., sections 2.1.2 and 4.1.2). We refer to this as the *intensity*, λ, of the search.

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, 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

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

We then normalise these values with reference solution quality and runtime values 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 either the best known (and often optimal) solution for instance p, or a lower bound on the optimal solution quality. The actual normalisation values used are discussed further in the relevant results sections (sections 2.2, 3.2 and 4.2). The time normalisation is more simple and we just calculate $\overline{T}_{\lambda,p}/\overline{T}^{\mathcal{A}}_p$ where $\overline{T}^{\mathcal{A}}_p$ is the average runtime on an instance p for some well known reference algorithm, A. Note that in choosing such a reference algorithm we aimed to use schemes which were linear in the problem size, e.g., $O(|V| + |E|)$ for the graph based problems. We also tried normalising with the problem size, $N = |V|$, and qualitatively the behaviour was almost identical, however this does not express so well the densities differences for the graph based problems.

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:

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

As well as demonstrating typical performance behaviour, the plots also give a useful guide to picking a good default value for the intensity. Thus setting $\lambda = \lambda^*$ at, or close to, the turning point in the gradient of a typical convergence curve provided by the experimentation below should give the best return in solution quality for the least runtime cost. Note that typically then for these sorts of methods it would be a good idea to use intensity values $\lambda \geqslant \lambda^*$ to get maximum benefit from the optimisation and so more importance should be attached to the right hand (slow decay) end of the convergence curves.

The tests discussed here were all carried out on a DEC Alpha machine with a 466 MHz CPU and 1 Gbyte of memory. Typically, although not universally, the runtime for a particular local search strategy at intensity λ is about the same as the runtime for a multilevel version at intensity $\lambda/2$ (see section 5.3) and this factor of two prompts the choice of intensity values during the testing. For each instance and each intensity value we ran 3 tests with different random seed values. In each case the runtime measurement includes reading in the problem, output of the solution and any initialisation required including an initial solution construction algorithm for the single-level local search schemes. Raw data for the testing can be found online.[1]

2. The graph partitioning problem

The k-way graph partitioning problem (GPP) can be stated as follows: given a graph $G(V, E)$, possibly with weighted vertices and/or edges, partition the vertices into k disjoint sets such that each set contains the same vertex weight and such that the *cut-weight*, the total weight of edges cut by the partition, is minimised. The GPP has a number of applications including circuit partitioning for optimal placement of electronic components on printed circuit boards and the partitioning of unstructured meshes for parallel process-

ing (mesh partitioning). It is well known that this problem is NP-complete (Garey and Johnson, 1979), so in recent years much attention has been focused on developing suitable heuristics. It is quite common in applications such as mesh partitioning to slightly relax the balancing constraint in order to improve the partition quality.

Typically many researchers have approached this problem by studying its restriction to 2 subsets, the graph bisection problem. This can then be easily extended to the full problem by recursion, i.e. the graph is bisected into two sub-problems which are then themselves bisected to give 4 sub-problems and so on. This technique is known as recursive bisection and has been used with a variety of bisection algorithms, e.g., (Simon, 1991). It is still widely used and is able to give guarantees on satisfying the balancing constraint, although the resulting partition quality may be limited (Simon and Teng, 1997). However with the advent of robust k-way partitioning algorithms, which arguably can be parallelised more easily and are perhaps better suited to dynamic load-balancing, there is now a considerable body of research on methods which solve the full problem in one go. In fact multilevel approaches have been applied to both recursive bisection, e.g., (Battiti, Bertossi, and Cappelletti, 1999; Bui and Jones, 1993; Karypis and Kumar, 1998a; Pellegrini and Roman, 1996) and k-way algorithms, e.g., (Hendrickson and Leland, 1995a; Karypis and Kumar, 1998b; Walshaw and Cross, 2000) and we test both approaches below, section 2.2.

2.1. Multilevel graph partitioning

The GPP was the first combinatorial optimisation problem to which the multilevel paradigm was applied and there is now a considerable volume of literature about multilevel partitioning algorithms. Initially used as an effective way of speeding up partitioning schemes, it was soon recognised as, more importantly, giving them a more 'global' perspective (Karypis and Kumar, 1998a), and has been successfully developed as a strategy for overcoming the localised nature of the Kernighan–Lin (KL) (Kernighan and Lin, 1970), and other optimisation algorithms. Typically such multilevel implementations match and coalesce pairs of adjacent vertices to define a new graph and recursively apply this procedure until the graph size falls below some threshold. The coarsest graph is then partitioned (possibly with a crude algorithm) and the partition is successively refined on all the graphs starting with the coarsest and ending with the original. At each change of levels, the final partition of the coarser graph is used to give the initial partition for the next level down. The use of multilevel combinatorial refinement for partitioning was first proposed by both Hendrickson and Leland (1995a) and Bui and Jones (1993), inspired by Barnard and Simon (1994), who used a multilevel numerical algorithm to speed up spectral partitioning.

Figure 1 shows an example of a multilevel partitioning scheme in action. On the top row (left to right) the graph is coarsened down to 4 vertices which are (trivially) partitioned into 4 sets (bottom right). The solution is then successively extended and refined (right to left; each graph shows the final partition for that level). Although at each level the refinement is only local in nature, a high quality partition is still achieved.

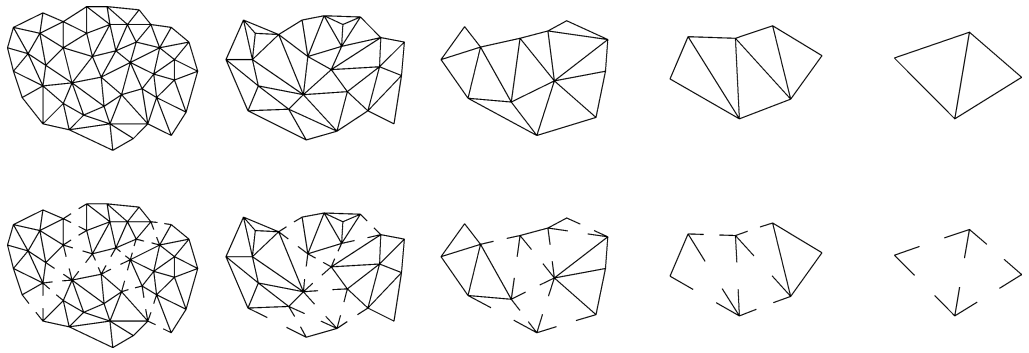

Figure 1. An example of multilevel partitioning.

2.1.1. Multilevel framework

Graph contraction. A common method to create a coarser graph $G_{l+1}(V_{l+1}, E_{l+1})$ from $G_l(V_l, E_l)$ is the edge contraction algorithm proposed by Hendrickson and Leland (1995a). The idea is to find a maximal independent subset of graph edges, or a *matching* of vertices, and then collapse them. The set is independent if no two edges in the set are incident on the same vertex (so no two edges in the set are adjacent), and maximal if no more edges can be added to the set without breaking the independence criterion. Having found such a set, each selected edge is collapsed and the vertices, $u_1, u_2 \in V_l$ say, at either end of it are merged to form a new vertex $v \in V_{l+1}$ with weight $\|v\| = \|u_1\| + \|u_2\|$. Edges which have not been collapsed are inherited by the child graph, G_{l+1}, and, where they become duplicated, are merged with their weight summed. This occurs if, for example, the edges (u_1, u_3) and (u_2, u_3) exist when edge (u_1, u_2) is collapsed. Because of the inheritance properties of this algorithm, it is easy to see that the total vertex weight remains the same, $\|V_{l+1}\| = \|V_l\|$, and the total edge weight is reduced by the sum of the collapsed edge weights.

A simple way to construct a maximal independent subset of edges is to create a randomly ordered list of the vertices and visit them in turn, matching each unmatched vertex with an unmatched neighbour (or with itself if no unmatched neighbours exist). Matched vertices are removed from the list. If there are several unmatched neighbours the choice of which to match with can be random, but it has been shown by Karypis and Kumar (1998a), that it can be beneficial to the optimisation to collapse the most heavily weighted edges.

This simple but rapid algorithm (which has $O(|E|)$ complexity) is guaranteed to construct a *maximal matching* (because no more edges can be added without breaking the independence criterion). However it will not necessarily construct a *maximum matching*, a matching with the largest possible size (which is bounded above by $|V|/2$ – i.e. if every vertex is matched with another then there will be $|V|/2$ edges in the set). A maximum matching is much more costly to construct (certainly greater than $O(|E|)$ in complexity) and algorithms for this purpose are the subject of a considerable body of literature, e.g., (Cook and Rohe, 1999). In practice the fact that a suboptimal matching is found appears not to matter (although see section 6.3.2 for further discussion).

The initial partition. The hierarchy of graphs is constructed recursively until the number of vertices in the coarsest graph is smaller than some threshold and then an initial partition is found for the coarsest graph. At its simplest, the contraction can be terminated when the number of vertices in the coarsest graph is the same as the number of subsets required, k, and then vertex i is assigned to subset S_i. However, since the vertices of the coarsest graph are not generally homogeneous in weight, this does require some mechanism for ensuring that the final partition is balanced, i.e. each subset has (approximately) the same vertex weight. Various methods have been proposed for achieving this, commonly either by terminating the contraction so that the coarsest graph G_L still retains enough vertices, $|V_L|$, to achieve a balanced initial partition (i.e. so that typically $|V_L| \gg k$) (Hendrickson and Leland, 1995a; Karypis and Kumar, 1998a), or by incorporating load-balancing techniques alongside the refinement algorithm, e.g., (Walshaw and Cross, 2000), where it is also shown that relaxing the balance constraint on the coarser levels and tightening it up gradually can enhance the resulting partition quality.

Partition extension. Having optimised the partition on a graph G_l, the partition must be extended onto its parent G_{l-1}. The extension algorithm is trivial; if a vertex $v \in V_l$ is in subset S_i then the matched pair of vertices that it represents, $v_1, v_2 \in V_{l-1}$, are also assigned to S_i.

2.1.2. Refinement: the Kernighan–Lin algorithm

At each level, the partition from the previous level is extended to give an initial partition and then refined. Various refinement schemes have been successfully used including *greedy* refinement, a steepest descent approach, which is allowed a small imbalance in the partition (typically 3–5%) and transfers border vertices from one subset to another if either (a) the move improves the cost without exceeding the allowed imbalance; or (b) the move improves the balance without changing the cost. Although this scheme cannot guarantee perfect balancing, it has been applied to very good effect (Karypis and Kumar, 1998b), and is extremely fast.

A more sophisticated class of method is based on the Kernighan–Lin (KL) bisection optimisation algorithm (Kernighan and Lin, 1970), which includes limited hill-climbing to enable it to escape from local minima. Recent implementations almost universally use the linear time complexity improvements (e.g., bucket sorting of vertices) introduced to partitioning by Fiduccia and Mattheyses (1982). We outline the KL refinement algorithm to illustrate the process, however in principle any iterative refinement scheme can be used and examples of successful multilevel implementations exist for simulated annealing (Vanderstraeten et al., 1996), tabu search (Battiti, Bertossi, and Cappelletti, 1999; Vanderstraeten et al., 1996), genetic algorithms (Kaveh and Rahimi-Bondarabady, 2000), cooperative search (Toulouse, Thulasiraman, and Glover, 1999), and even ant colony optimisation (Langham and Grant, 1999).

A typical KL-type algorithm will have inner and outer iterative loops with the outer loop terminating when no vertex transfers take place during an inner loop. It is initialised by calculating the *gain* – the potential improvement in the cost function (the

cut-weight) – for all border vertices. The inner loop proceeds by examining candidate vertices, highest gain first, and if the candidate vertex is found to be acceptable (i.e. it does not overly upset the load-balance), it is transferred. Its neighbours have their gains updated and, if not already tested in the current iteration of the outer loop, join the set of candidate vertices.

The KL hill-climbing strategy allows the transfer of vertices between subsets to be accepted even if it degrades the partition quality and later, based on the subsequent evolution of the partition, the transfers are either rejected or confirmed. During each pass through the inner loop, a record of the best partition achieved by transferring vertices within that loop is maintained together with a list of vertices which have been transferred since that value was attained. If, during subsequent transfers, a better partition is found then the transfer is confirmed and the list is reset.

This inner loop terminates when a specified number of candidate vertices have been examined without improvement in the cost function. This number (i.e. the maximum number of continuous failed iterations of the inner loop) provides the user specified intensity of the search, λ (see section 1.3). Note that if $\lambda = 0$ then the refinement is purely greedy in nature (as mentioned above). Once the inner loop is terminated, any vertices remaining in the list (vertices whose transfer has not been confirmed) are transferred back to the subsets they came from when the best cost was attained.

The KL algorithm has been extended to k-way partitioning in different ways by several authors (e.g., Hendrickson and Leland, 1995a; Karypis and Kumar, 1998b; Walshaw and Cross, 2000). For example Hendrickson and Leland achieve this by calculating the gain for moving each border vertex to any of the $k - 1$ other sets (Hendrickson and Leland, 1995a). Unfortunately this can add a significant time and memory penalty (Hendrickson and Leland, 1995b), and so in the k-way version used below, Walshaw and Cross just calculate the gain for each set to which the border vertex is adjacent and only store the highest gain value (Walshaw and Cross, 2000).

Weighted graphs. Even though the original graph may not be weighted, the coarsened graphs will all have weights attached to both vertices and edges because of the contraction process. Furthermore the original problem is not correctly represented unless the weights are used to evaluate partitions of the coarsened graphs and so the refinement algorithm must take them into account. In fact it is not difficult to incorporate the edge weights into the gain function. For the vertex weights it is not always so straightforward to modify the load-balancing mechanisms (particularly in the original version of KL where pairs of vertices are swapped), but the relaxation to allow imbalanced partitions does facilitate their use, e.g., (Walshaw and Cross, 2000).

2.1.3. Iterated multilevel partitioning

If a partition of the graph already exists prior to optimisation it can be reused during the multilevel procedure. Thus, given a k-way partition of the original problem we can carry out *solution-based coarsening* by insisting that, at each level, every vertex v matches with a neighbouring vertex in the same set. When no further coarsening is possible this

will result in a partition of the coarsest graph with the same cost as the initial partition of the original. Provided the refinement algorithms guarantee not to find a worse partition than the initial one (in fact even if they do find a worse partition it can be replaced by the initial one) the multilevel refinement can then guarantee to find a new partition that is no worse than the initial one.

This technique can be used to find very high quality partitions, albeit at some expense, by repeatedly coarsening and uncoarsening to iterate the procedure. At each iteration the current solution can be used to create a solution-based coarsening and construct a new hierarchy of graphs and as we have seen the process guarantees not to find a worse solution than the initial one. However, if the matching includes a random factor, each iteration is very likely to give a different hierarchy of graphs to previous iterations and hence allow the refinement algorithm to visit different solutions in the search space.

We refer to this process as an *iterated multilevel algorithm* (see also (Toulouse, Thulasiraman, and Glover, 1999) for a variation of this technique); it is analogous to the use of V-cycles in multigrid. Note that it requires the user to specify a second intensity parameter, γ, namely the number of failed outer iterations (i.e. the number of times the algorithm coarsens and uncoarsens the graph without finding a better solution). Clearly the optimal choices of λ and γ are somewhat interdependent and it would require considerable experimentation to determine the best combination, if such a characterisation is even possible; for now we choose $\lambda = \lambda^*$ (see section 1.3). We can then vary γ to test the iterated multilevel algorithm and give some sample results for this scheme below.

2.2. Experimental results

To illustrate the potential gains offered by multilevel schemes we conducted a number of experiments. These are not intended to be exhaustive but merely give an indication of typical performance behaviour.

We use the Kernighan–Lin (KL) scheme to demonstrate the performance of the multilevel algorithm because arguably the multilevel KL scheme represents the state of the art in graph partitioning – see (Schloegel, Karypis, and Kumar, 2004). As further evidence we would cite the fact that all of the five public-domain graph partitioning packages use multilevel refinement and four (Chaco (Hendrickson and Leland, 1995a), Jostle (Walshaw and Cross, 2000), Metis (Karypis and Kumar, 1998b), and Scotch (Pellegrini and Roman, 1996)) use a multilevel Kernighan–Lin variant as the default setting (often referred to as Fiduccia–Mattheyses after the linear-time complexity improvements in (Fiduccia and Mattheyses, 1982), see section 2.1.2 above). Meanwhile the fifth package, Party (Monien, Preis, and Diekmann, 2000), uses a multilevel version of a refinement scheme known as *helpful sets*, itself an extension of the KL algorithm which swaps several vertices at a time (Diekmann et al., 1996). Furthermore there is evidence that these packages tend produce reasonably similar results, e.g., (Monien, Preis, and Diekmann, 2000; Pellegrini and Roman, 1996; Soper, Walshaw, and Cross, 2000; Walshaw and

Cross, 2000), with no particular version dominating the others in terms of both runtime *and* solution quality, although Metis is usually the fastest. Even in the case of other more general local search schemes such as simulated annealing or tabu search there is good evidence that the multilevel approach can enhance the partitioning results, e.g., (Battiti, Bertossi, and Cappelletti, 1999; Kaveh and Rahimi-Bondarabady, 2000; Langham and Grant, 1999; Toulouse, Thulasiraman, and Glover, 1999; Vanderstraeten et al., 1996), and so although there is no recent experimental survey which compares a range of algorithms it seems hard to deny that the state of the art in graph partitioning involves a multilevel algorithm in some way. It is true that there are also geometric partitioning schemes which might also make a claim, e.g., (Gilbert, Miller, and Teng, 1998), but these are really suitable for mesh-partitioning as they require coordinates for the vertices plus some quality guarantees on the mesh.

We use two test suites, one of which is a collection of 16 sparse, mostly mesh-based graphs drawn from a number of real-life applications and collected together online[2] with some larger examples for benchmarking partitioning algorithms (Soper, Walshaw, and Cross, 2000). The other test suite consists of 90 instances compiled to test graph colouring algorithms for the 2nd DIMACS implementation challenge (Johnson and Trick, 1996), augmented by further examples added since then[3] and including a number of randomly generated examples. Although perhaps not representative of partitioning applications, some interesting results came to light with this suite in (Walshaw, 2001b) whilst testing multilevel colouring algorithms and we were interested in investigating this sort of behaviour further for partitioning. This colouring test suite is further subdivided, as in (Walshaw, 2001b), into 3 density classes; low ($0\% \leqslant \Delta \leqslant 33\frac{1}{3}\%$) with 58 out of 90 instances, medium ($33\frac{1}{3}\% < \Delta \leqslant 66\frac{2}{3}\%$) with 23 instances and high ($66\frac{2}{3} < \Delta \leqslant 100\%$) with just 9 instances.

Note that although the distinction between sparse and low-density graphs is not always clear, typically by sparse we mean families of graphs for which the number of edges $|E|$ is $O(|V|)$ and so the density decreases with increasing $|V|$. Meanwhile by low-density we tend to mean families of graphs which have $O(|V|^2)$ edges but for which the density, $\Delta \ll 100\%$, remains constant with increasing $|V|$ (for example the colouring suite contains a series of randomly generated graphs of different sizes but fixed density of 0.1).

The tests compare both the k-way and recursive bisection based Kernighan–Lin algorithms against multilevel versions at a range of intensities (see section 1.3). The solution quality normalisation is against the best known solutions found to date. These are in the public-domain[4] and were found through extensive testing by a range of algorithms, most notably a combined evolutionary/multilevel scheme (Soper, Walshaw, and Cross, 2000), which can take weeks to run for large instances, but also including the multilevel partitioning packages, Chaco, Jostle, and Metis, as well as recursive spectral bisection (Barnard and Simon, 1994), and (where the graph has vertex coordinate information) recursive coordinate bisection (Simon, 1991). We normalise the runtime using the greedy partition construction algorithm (Farhat, 1988), because this simple but well-known scheme is $O(|V| + |E|)$ in complexity thus capturing the problem size well. All

tests were required to look for 16-way partitions and were allowed an imbalance of 3%; this is indicative only and more thorough testing, at least for sparse graphs, can be found elsewhere.

2.2.1. Direct k-way partitioning

The first set of test results compares a k-way Kernighan–Lin algorithm (kKL) against a multilevel version (MLkKL). The initial partition for the kKL results was provide by the greedy partition construction algorithm (Farhat, 1988), also used for runtime normalisation. All of the algorithms used are available within the framework of Jostle, a partitioning tool developed at the University of Greenwich and freely available for academic and research purposes under a licensing agreement.[5]

The intensity parameter was the number of failed iterations of the KL inner loop (see section 2.1.2) and for the multilevel versions λ_l, the search intensity at level l, is set to $\lambda_l = \lambda/(l+1)$ where the original problem is level 0 and so $\lambda_0 = \lambda$. Note that if the intensity were to control the outer loop we would expect, for kKL at least, that the curve should decrease monotonically. This is because for $\lambda_1 < \lambda_2$, a test at intensity λ_2 would typically just extend the optimisation from the best solution found at intensity λ_1. Since the optimisation can always return to the best solution found for λ_1, the solution found for λ_2 should never be worse than that found for λ_1. However, because here the intensity controls the inner loop, this monotonicity is lost and the quality does sometimes degrade slightly as the intensity increases, e.g., figure 2(a).

Using the methodology described in section 1.3 we conducted tests to compare $k\text{KL}^\lambda$ (with $\lambda = 0$ and $\lambda = 2^m$ with $m = 0, \ldots, 14$) against ML$k\text{KL}^\lambda$ (with $\lambda = 0$ and $\lambda = 2^m$ with $m = 0, \ldots, 13$). Figure 2(a) shows the results for the sparse graphs and the dramatic quality improvement imparted by the multilevel framework is immediately clear. Even for purely greedy refinement (i.e. the extreme left-hand point on either curve where $\lambda = 0$ – see section 2.1.2 above) the MLkKL solution quality is far better than kKL and it is results like these that have helped to promote multilevel partitioning algorithms to the status they enjoy today.

Figures 2(b)–(d) show the partitioning results for the colouring test suite. Here we test $k\text{KL}^\lambda$ (with $\lambda = 0$ and $\lambda = 2^m$ with $m = 0, \ldots, 10$) against ML$k\text{KL}^\lambda$ (with $\lambda = 0$ and $\lambda = 2^m$ with $m = 0, \ldots, 9$). Figure 2(b) more or less confirms the conclusions for the sparse results and although the curves are closer together, MLkKL is the clear winner. For the medium and high-density examples however, it is a surprise (especially considering the widely accepted success of multilevel partitioning) to find that these conclusions are no longer valid. For the high-density instances, figure 2(d), MLkKL is still the leading algorithm, although only marginally. However for the medium-density results, figure 2(c), MLkKL fails to achieve the same performance as kKL and the multilevel framework appears to actually hinder the optimisation. This could simply be due to the fact that one of the algorithms is tuned badly for this particular class but since multilevel algorithms are widely accepted for partitioning this should raise a note of caution to practitioners. For now this is left as an observation but we discuss it further in section 6.1.

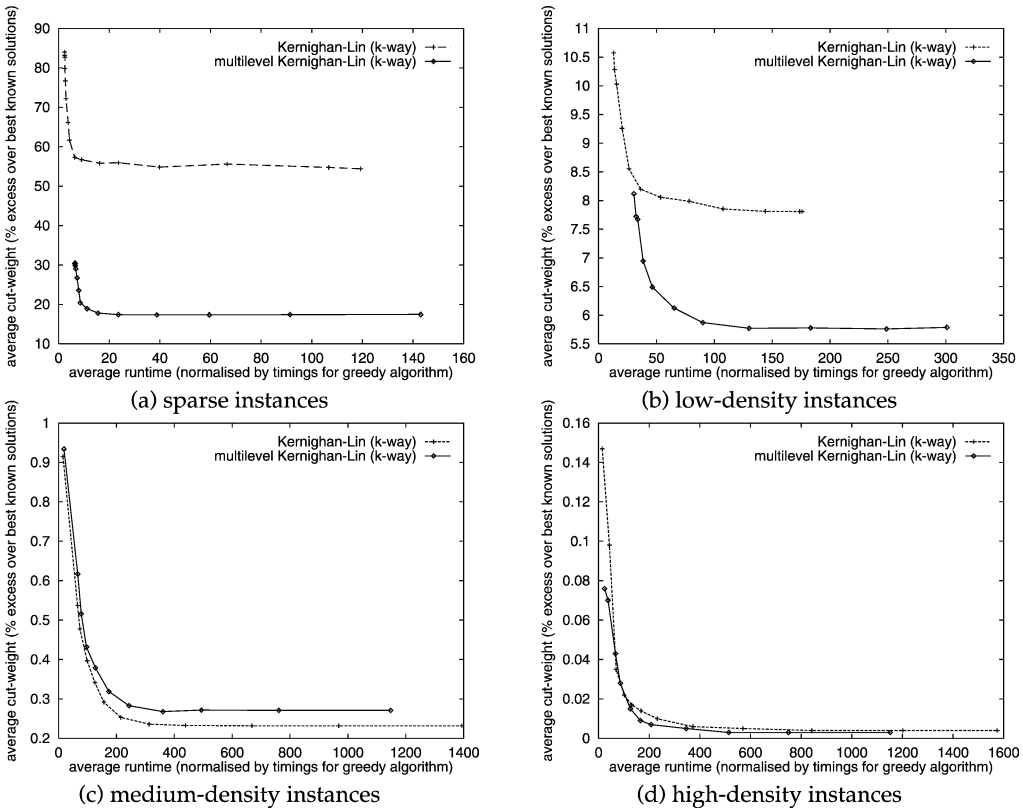

Figure 2. Plots of convergence behaviour for the GPP.

To measure the variation of algorithmic performance across the test suite, we computed, σ, the standard deviation (in this case the deviation from the percentage excess over best known solutions), for each data point in figure 2. The absolute values of the σ values are difficult to summarise (and space precludes plots of them), but a comparison between the two methods, MLkKL and kKL, revealed that the values were around 25–100% worse for kKL across the colouring suite and around 8 times worse for the sparse instances. Interestingly this was true even for the medium density instances where kKL produces better results on average. The implications are that MLkKL has a much lower variation of results across the test suite and we take this to mean that the multilevel framework 'stabilises' kKL in some way.

Although we do not present raw data here, it can be found online.[6] However within each class the runtimes scale reasonably well with problem size and for example if $\lambda = \lambda^* = 64$ (see section 1.3), MLkKL can achieve a good quality partition for a typical sparse graph, 4elt, with $|V| = 15{,}606$ and $|E| = 45{,}878$ in around 0.3 seconds. Meanwhile for the colouring suite the random low, medium and high-density graphs, DSJC1000.1, DSJC1000.5 and DSJC1000.9, each with $|V| = 1{,}000$ and $|E| = 49{,}629$, $|E| = 249{,}826$ and $|E| = 449{,}449$, can be partitioned in around 9, 120 and 240 sec-

onds, respectively. The apparent discrepancy between the 4elt and DSJC1000.1 runtimes (with very similar numbers of edges) arises from the fact that higher densities and hence higher connectivities significantly complicate the partitioning problem.

Finally note that the scale on the vertical axis is very different for each plot in figure 2. This is because, as the density increases, the proportion of edges which must be cut, even for an optimal partition, increases whilst the number of edges which may or may not be cut, depending on the solution quality, decreases.

2.2.2. Recursive bisection partitioning

To augment the evidence above we decided to apply a different implementation and configuration of the KL algorithm to the same test suite. We chose the Chaco software,[7] developed by Hendrickson and Leland, because it is well established and because, of the five packages mentioned above, apart from Jostle it is the only one which gives explicit control over an intensity parameter. For the tests we then chose a recursive bisection version of the KL algorithm (i.e. via recursive application of 2-way partitioning) and which we refer to as 2KL. Although Chaco offers quadrisection and octasection in addition, the authors note that because their implementation calculates $k - 1$ gains for each border vertex, these options can significantly increase the time and memory complexity (Hendrickson and Leland, 1995b). Recursive bisection also broadens the scope of the testing.

The intensity parameter, λ, is the number of failed iterations of the outer loop (each pass of the outer loop may produce different results because of randomisation in the ordering within the bucket sorting structure). The tests then compare 2KL^{λ} (with $\lambda = 0$ and $\lambda = 2^m$ with $m = 0, \ldots, 10$) against ML2KL^{λ} (with $\lambda = 0$ and $\lambda = 2^m$ with $m = 0, \ldots, 9$). The initial partitions for single-level version were computed by linear assignment (the first $\lceil |V|/k \rceil$ vertices are assigned to set 1, etc.) and the multilevel version collapsed the graph down to 16 vertices.

The results are shown in figure 3 alongside the k-way results. Note that because of large variations in runtime between the two approaches we plot the log of average normalised runtime. Thus figure 3 includes the same information as figure 2 but the horizontal scaling is very different and arguably less visually representative of typical behaviour with its long shallow decay curves. The plots broadly confirm the conclusions from the previous section that multilevel enhancement of KL can be spectacularly successful on sparse and low-density problems. Indeed ML2KL (and 2KL) significantly improve on the kKL results for the low-density instances (although it is not clear why this should be the case). This illustrates very well the primary conclusion of this paper – although no one algorithm is dominant across all problem instances and intensities, nonetheless, given a good local search algorithm, a multilevel version of that scheme may be able to improve on the results over a significant proportion of its application range and hence the multilevel paradigm is worth considering as an addition to the combinatorial optimisation toolkit.

A number of other observations can be made about the 2KL results. Firstly in contrast to kKL, ML2KL still dominates 2KL for the medium density instances, although

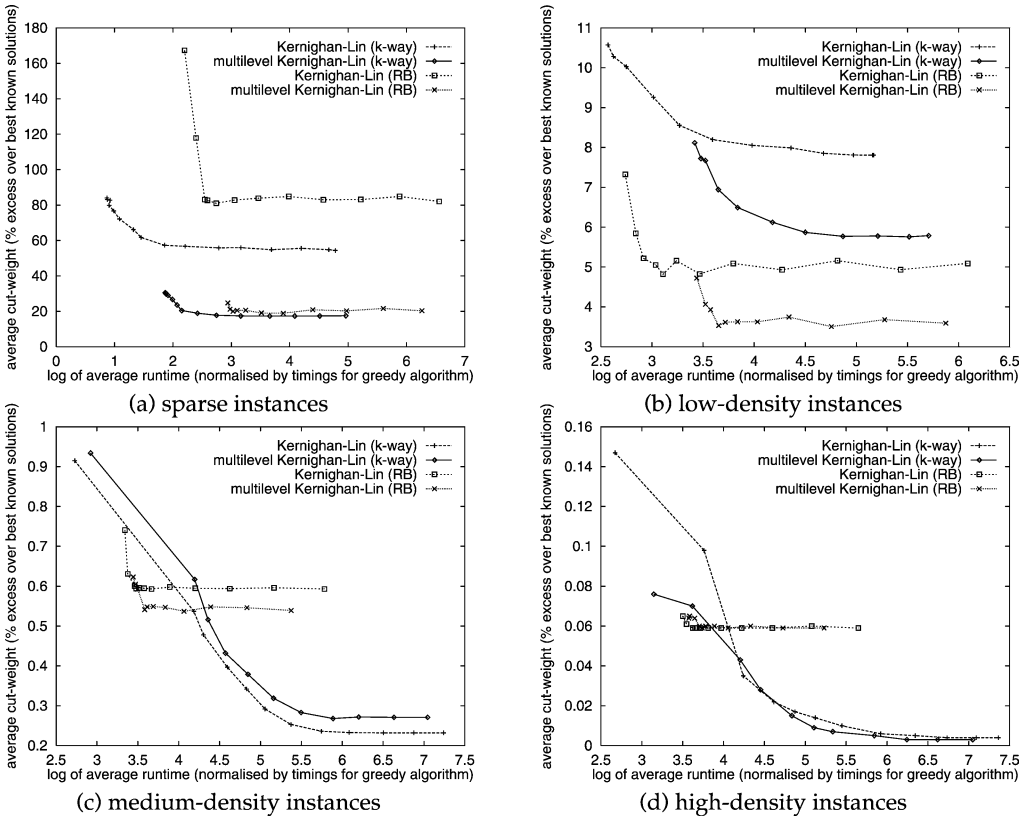

Figure 3. Plots of convergence behaviour for the GPP with logarithmic runtime scaling.

the asymptotic convergence is considerably worse than both kKL and MLkKL. Secondly the 2KL and ML2KL results do not decrease monotonically as λ increase; this is also true for kKL, as mentioned above, but the results seem to be markedly more unstable here. We believe that this is due to the underlying optimisation structure inherent in recursive bisection schemes. Thus a decision about improving one particular bisection has no regard to subsequent bisections of the subproblems and indeed may have deleterious effects. This has been investigated in detail by Simon and Teng (1997), who show that even optimal bisections at each recursion may lead to a final k-way partition that is far from optimal. We believe that this also accounts for the fact that increasing the intensity beyond about $\lambda = 64$ fails to improve the results significantly and hence the curves have a long unstable tail (as compared with the smooth slow decay of the kKL curves).

The recursive bisection results showed very similar variation of performance to the k-way results (at least as measured by the standard deviation) with much higher deviation from the average for 2KL as compared with ML2KL. Again we take this to indicate that the multilevel framework stabilises the performance of 2KL in some way.

Once again raw data can be found online,[8] but to give some example of runtime for $\lambda = 64$, ML2KL can partition a typical sparse graph, 4elt, in around 4 seconds

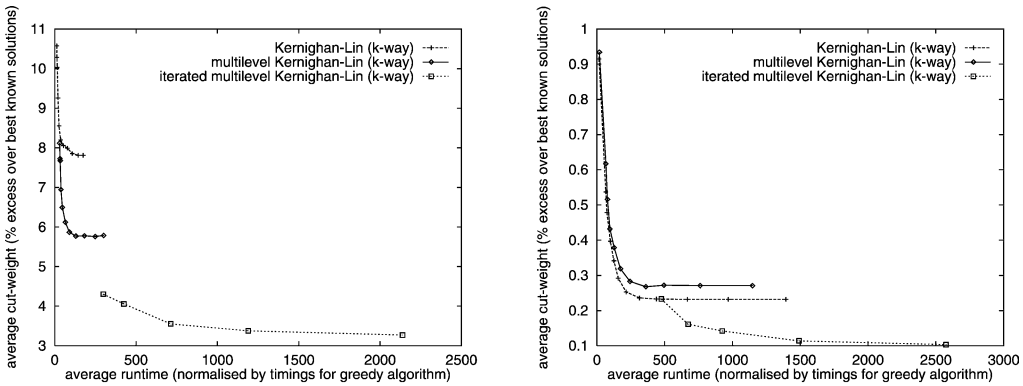

Figure 4. Plots of convergence behaviour including iterated multilevel partitioning results.

and the random low, medium and high-density graphs, DSJC1000.1, DSJC1000.5 and DSJC1000.9, in around 2, 11 and 26 seconds, respectively.

2.2.3. Iterated multilevel results

Figure 4 illustrates the results for the iterated multilevel algorithm (IMLkKL) described in section 2.1.3 alongside the MLkKL and kKL results for low and medium-density subclasses of the colouring suite. These plots contain the same information about MLkKL and kKL as figures 2(b) and 2(c) only it is more compressed here because of the long IMLkKL runtimes. For the IMLkKL results the inner intensity parameter was fixed at $\lambda = \lambda^* = 64$ (see section 2.1.3) whilst the iterated intensity parameter, γ, is varied with $\gamma = 2^l$ and $l = 0, \ldots, 4$.

We do not show results for the sparse and high-density instances because they are not so interesting; for the sparse suite IMLkKL more or less continues the MLkKL curve in figure 2(a) with a few percentage points improvement and very shallow decay whilst for the high-density instances IMLkKL does not appear to offer much improvement at all. However for the low and medium-density subclasses, in figures 4(a) and 4(b), respectively, the asymptotic performance offered by IMLkKL is impressive and worthy of further and more thorough investigation. In both cases IMLkKL improves on MLkKL (and ML2KL) and, for the medium-density instances, even appears to overcome the shortcomings of MLkKL and exceeds the kKL results.

3. The travelling salesman problem

The travelling salesman problem (TSP) can be stated as follows: given a collection of 'cities,' find the shortest tour which visits all of them and returns to its starting point. Typically the cities are given coordinates in the 2D plane and then the tour length is measured by the sum of Euclidean distances between each pair on the tour. However, in

the more general form, the problem description simply requires a metric which specifies the distance between every pair of cities.

The TSP has been shown to be NP-hard (Garey and Johnson, 1979), but has a number of features which make it stand out amongst combinatorial optimisation problems. Firstly, and perhaps because of the fact that the problem is so intuitive and easy to state, it has almost certainly been more widely studied than any other combinatorial problem. For example Johnson and McGeoch (1997), survey a wide range of approaches which run the gamut from local search, through simulated annealing, tabu search and genetic algorithms to neural nets. Remarkably, and despite all this interest, the local search algorithm proposed by Lin and Kernighan (1973), still remains at the heart of the most successful approaches. In fact Johnson and McGeoch describe the Lin–Kernighan (LK) algorithm as the world champion heuristic for the TSP from 1973 to 1989. Further, this was only conclusively superseded by chained or iterated versions of LK (CLK/ILK) originally proposed by Martin, Otto, and Felten (1991).

Even until recently, in spite of all the work on exotic and complex combinatorial techniques, Johnson and McGeoch (1997), concluded in 1997 that an iterated or chained Lin–Kernighan (ILK/CLK) scheme provides the highest quality tours for a reasonable cost and that CLK/ILK variants are 'the most cost effective way to improve on Lin–Kernighan, at least until one reaches stratospheric running times.' In fact, more recently an interesting LK variant has been developed by Helsgaun (2000), which, at least in its multi-trial version, can significantly improve on CLK/ILK results. However this scheme, which we shall refer to here as the Lin–Kernighan–Helsgaun (LKH) algorithm, suffers from running times which are quadratic in N, the problem size, and hence is unsuited to large instances.

Another unusual feature of the TSP is that, for problems which have not yet been solved to optimality (typically with 10,000 or more cities), an extremely good lower bound can be found for the optimal tour length. This bound, known as the Held–Karp Lower Bound (HKLB), was developed in 1970 by Held and Karp (1970), and usually comes extremely close to known optimal tour lengths (often within 1%). Thus to measure the quality of an algorithm for a given set of problem instances (some or all of which have unknown solutions), we can simply calculate the average percentage excess of tours produced by the algorithm over the HKLB for each instance.

It is sometimes convenient to use graph notation (see section 1.2) for the TSP in which case we refer to the cities as vertices and the problem can be specified as a complete graph with weighted edges, i.e. there is an edge between every pair of cities and the weight of the edge specifies the distance between them.

3.1. A multilevel algorithm for the travelling salesman problem

Recently the multilevel paradigm has been applied to the TSP (Johnson and McGeoch, 2002; Walshaw, 2002, 2001c). Although the scheme is perhaps less intuitive than multilevel partitioning, clearly the LK algorithm or one of its variants should make a good refinement method. However, with no graph as such, how can the problem be coarsened?

In fact from (Walshaw, 2002) it seems that the crucial point in devising a coarsening algorithm is the requirement that the solution to each coarsened problem must contain a solution of the original problem. One way of achieving this is for the coarsening to successively fix edges into the tour. For example, given a TSP instance P of size N, if we fix an edge between cities c_a and c_b then we create a smaller problem P' of size $N-1$ (because there are $N-1$ edges to be found) where we insist that the final tour of P' must somewhere contain the fixed edge (c_a, c_b). Having found a tour T' for P' we can then return to P and look for better tours using T' as the initial tour. In fact we can fix many distinct edges in one coarsening step, again by vertex matching, and hence reduce the size of the problem considerably at every level.

Figure 5 shows an example of this. The top row demonstrates the coarsening process where dotted lines represent matchings of vertices (and hence new fixed edges) whilst solid lines represent fixed edges that have been created in previous coarsening steps. Notice in particular that after the second coarsening step chains of fixed edges are reduced down to a single edge with a vertex at either end and any vertices internal to such a chain are removed. The coarsening terminates when the problem is reduced to one fixed edge and two vertices and at this point the tour is initialised. The initialisation is trivial and merely consists of completing the cycle by adding an edge between the two remaining vertices. The procedure then commences the extend/refine loop (bottom row, right to left). Again solid lines represent fixed edges whilst dotted lines represent free edges which may be changed by the refinement. The extension itself is straightforward; we simply expand all fixed edges created in the corresponding coarsening step and add the free edges to give an initial tour for the refinement process. The refinement algorithm then attempts to improve on the tour (without changing any of the fixed edges) although notice that for the first refinement step no improvement is possible. The final tour is shown at the bottom left of the figure; note in particular that fixing any edge during coarsening does not force it to be in the final tour since for the final refinement step all edges are free to be changed. However, fixing an edge early on in the coarsening does give it fewer possibilities for being flipped.

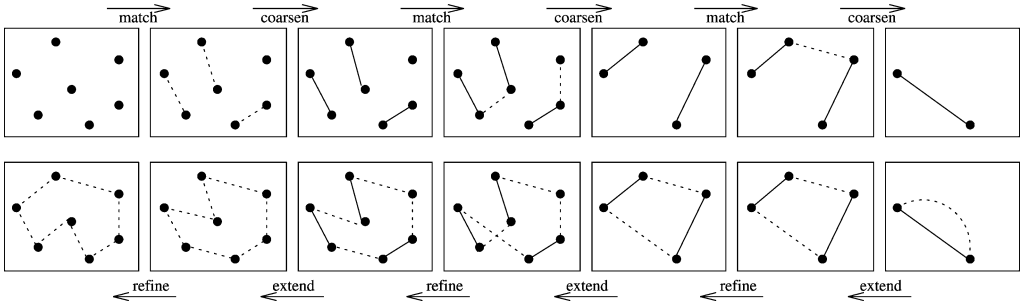

Figure 5. An example of a multilevel TSP algorithm at work.

3.1.1. Multilevel framework

Matching and coarsening. The implementation of this coarsening process in which vertices are matched and edges fixed between them is fully described in (Walshaw, 2002). In fact it is more convenient for the data structure to use edge objects and so in practice a matching of edges is created at each level (in much the same way that a matching of vertices is created for the multilevel partitioning algorithm). Initially each edge is of zero length and has the same vertex at either end; however after the first coarsening most edges will have different vertices at either end.

The aim during the matching process should be to fix those edges that are most likely to appear in a high quality tour thus allowing the refinement to concentrate on the others. Indeed, if by some good fortune, the matching *only* selected optimal edges then the optimal tour would have been found by the end of the matching process and the refinement would have no possible improvements. However, in the absence of information about the optimal tour, vertices are matched with their nearest unmatched neighbours.

The implementation of this process uses a superimposed grid of spacing h to avoid $O(N)$ searches to find the nearest neighbours. An important consequence is that at each level vertices are only allowed to match with neighbours within a distance h, although at each level h is increased (to keep the average number of vertices per grid cell constant). This prevents long-range matching on the lower levels of the coarsening and appears to have an important effect on the results (see section 3.2).

Initialisation. The coarsening ceases when further contraction would cause a degenerate problem, in this case when there remain only two vertices with a fixed edge between them. This is guaranteed to occur because each coarsening level will match at least one pair of vertices and so the problem size will shrink. Initialisation is then trivial and consists of adding an edge between the two vertices to complete the tour.

3.1.2. Refinement: the Lin–Kernighan algorithm and variants

The multilevel TSP algorithm described in (Walshaw, 2002) uses the chained Lin–Kernighan (CLK) algorithm for the refinement step of the multilevel procedure, probably the most effective local search technique for iteratively optimising a TSP tour. However in a recent extension of the scheme, a version which uses the Lin–Kernighan–Helsgaun algorithm has also been developed (Walshaw, 2001c).

Typically TSP tour optimisation takes place by 'flipping' edges. Thus, if the tour contains the edges (v_1, w_1) and (w_2, v_2) in that order, then these two edges can always be flipped to create (v_1, w_2) and (w_1, v_2). This sort of step forms the basis of the 2-opt algorithm (Croes, 1958), which is a steepest descent approach, repeatedly flipping pairs of edges if they improve the tour quality until it reaches a local minimum of the objective function and no more such flips exist. In a similar vein, the 3-opt algorithm of Lin (1965), exchanges 3 edges at a time. The Lin–Kernighan (LK) algorithm (Lin and Kernighan, 1973), also referred to as variable-opt, however incorporates a limited amount of hill-climbing by searching for a sequence of exchanges, some of which may individually increase the tour length, but which combine to form a shorter tour. A vast

amount has been written about the LK algorithm, including ideas to improve its performance together with much on its efficient implementation; for an excellent overview of techniques see the surveys of Johnson and McGeoch (2002, 1997).

The basic LK algorithm employs a good deal of randomisation and for many years the accepted method of finding the shortest tours was simply to run it repeatedly with different random seed values and pick the best (a technique which also had the advantage that it could be run in parallel on more than one machine at once). Martin, Otto, and Felten's important contribution to the field (Martin, Otto, and Felten, 1991), came with the observation that, instead of restarting the procedure from scratch every time, it was more efficient to perturb the final tour of one LK search and use this as the starting point for the next. In their original approach, Martin *et al.* referred to their algorithm as chained local optimisation and used it as a form of accelerated simulated annealing. Thus they would perturb or 'kick' a tour and use LK to find a nearby local minimum. If the new tour was not as good as the champion tour at that point, the algorithm would decide whether or not to keep it as a starting point for the next perturbation by using a simulated annealing cooling schedule. Subsequent implementations however generally discard any new tour which does not improve on the current champion and always perturb the champion, e.g., (Applegate et al., 1999; Johnson and McGeoch, 1997). The version used here takes this approach and is known as the chained Lin–Kernighan (CLK) algorithm.

We also experiment with a new and highly effective variant of the LK algorithm developed by Helsgaun (2000). This scheme employs a number of important innovations including sequential 5-opt moves and the use of sensitivity analysis to direct the search. It has been shown to compute solutions extremely close to the optimal (where known) but suffers from the drawback of runtimes which are quadratic in N, the problem size, and so may not be suitable for large problem instances (e.g., if $N > 10{,}000$ the runtime for N trials can be measured in days whilst for $N = 30{,}000$ it runs to weeks). The algorithm is most effective in its multi-trial version where the optimisation is run multiple times, each with a different initial tour, the construction of which is biased by the edges in the existing champion tour. This is slightly different from the kicks used in the CLK algorithm where each initial tour is constructed by perturbing the existing champion tour and appears to be a very effective modification.

Fixed edges. The only additional requirement to the refinement scheme necessary for the implementation of the multilevel version is to ensure that none of the fixed edges are exchanged. For CLK this was enforced by altering the subroutine which calculated edge lengths between a given pair of cities to return a large negative value whenever it was asked to evaluate the length of a fixed edge; for LKH this functionality was already built into the software.

3.2. Experimental results

The multilevel CLK algorithm is tested in detail in (Walshaw, 2002) and we summarise the results here. We use the chained Lin–Kernighan algorithm of Applegate et al. (1999), because this is representative of the state of the art in finding tours for the TSP (Johnson

and McGeoch, 2002). This has been demonstrated recently in the 8th DIMACS implementation challenge,[9] a comprehensive and exemplary survey of TSP heuristics aimed at creating 'a reproducible picture of the state of the art in the area of TSP heuristics (their effectiveness, their robustness, their scalability, etc.),' and in which researchers were invited to submit their own results on a given test set of problems (Johnson and McGeoch, 2002). This study indicates that the chained/iterated Lin–Kernighan variants are ahead of the field, at least in terms of algorithms which are subquadratic in complexity. Furthermore, (Johnson and McGeoch, 2002) shows that the implementation of Applegate et al. is amongst the leading variants (at least it is not dominated by any other algorithm). However if better results are required and runtime is not a prime consideration, another contender, although with $O(|V|^2)$ complexity and hence suitable for small/medium-sized instances only, is the Lin–Kernighan–Helsgaun (LKH) variant and we include results for this algorithm in section 3.2.2.

The experiments were carried out on a test suite of 80 TSP problem instances, a large subset of the 90 instances compiled for the DIMACS challenge. We excluded 2 instances too large for our test platform, i.e. with 3 million or more vertices and 8 specified by a distance matrix because the matching algorithm requires coordinate information (although this is not an inherent attribute of the multilevel framework and it should be possible to develop suitable matching techniques). The instances used are then in three groups:

(I) All 33 symmetric and geometric instances of 1,000 or more vertices from `TSPLIB`,[10] a collection of sample TSP instances, including some from real-life applications, compiled by Reinelt (1991).

(II) 24 randomly generated instances with uniformly distributed vertices. These range in size from 1,000 to 1,000,000 vertices, going up in size gradations of $\sqrt{10}$ and were constructed by Johnson, Bentley, and McGeoch specifically to study asymptotic behaviour in tour finding heuristics.

(III) 23 randomly generated instances with randomly clustered vertices. These range in size from 1,000 to 100,000 and have the same origin and purpose as (II) although clustered examples such as these are generally considered to be more difficult to solve.

The CLK software is contained in an optimisation package written by Applegate et al. (1999), and known as `concorde`[11] whilst the LKH software is available from its author, Helsgaun.[12] The intensity parameter, λ (see section 1.3) was chosen as the number of outer iterations or (chainings or trials) expressed as a fraction of N the problem size, i.e. $\lambda = xN$ for some factor x. For the multilevel versions, the intensity parameter at each level, λ_l, was then set to $\lambda_l = xN_l$ where N_l is the problem size (the number of free edges) at level l.

We use the runtime of the LK algorithm to normalise the timing results because it is such a well-known TSP heuristic. We then use the Held–Karp lower bound to normalise the solution quality because it provides a very good estimate of optimal tour length (see

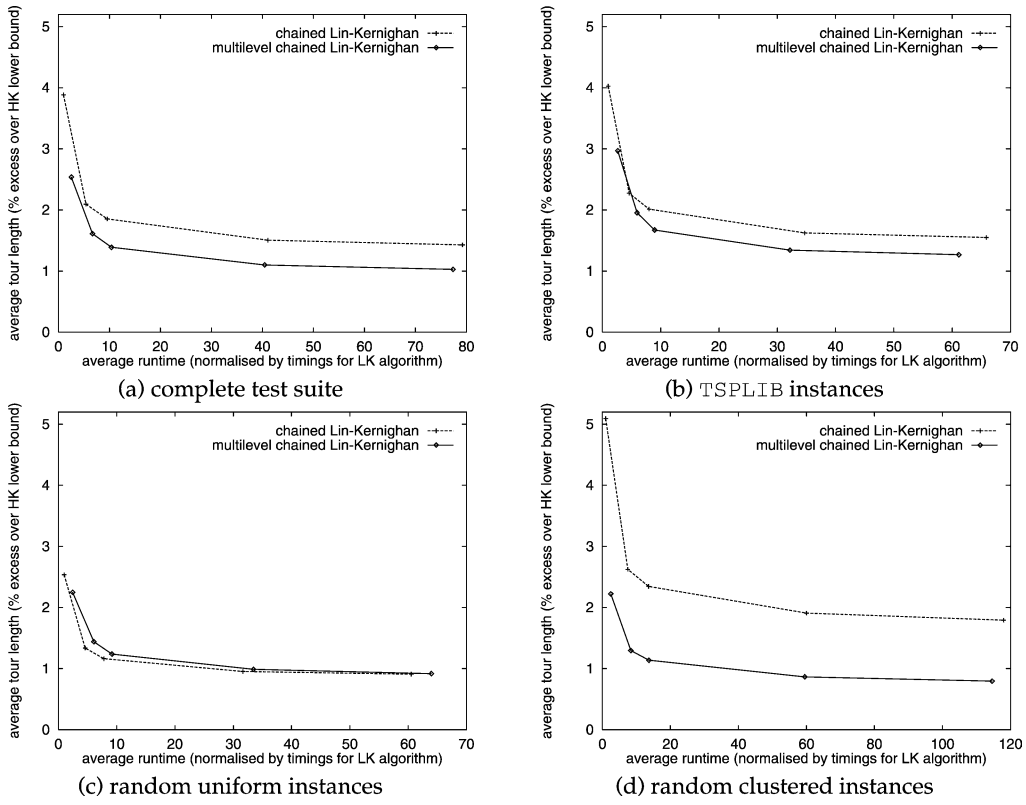

Figure 6. Plots of convergence behaviour for the TSP.

above). It is a standard way of expressing TSP solution quality and in particular is used for the 8th DIMACS implementation challenge (Johnson and McGeoch, 2002). As an estimate it is not entirely uniform across the problem classes; for example, for the 29 out of 33 TSPLIB instances with known optimal solutions the average excess over the HKLB is 1.00%, whilst for the 15 out of 24 random uniform instances this is 0.73% and for the 15 out of 23 random clustered instances it is 0.56% (giving an overall average of 0.82% for the 59 out of 80 instances with known optimum solutions). We have not drawn these lines on the plots in figures 6 and 7 because they do not apply to the larger instances in each class; however, if the figures were the same across the entire suite then the multilevel results look even more impressive.

3.2.1. The chained Lin–Kernighan algorithm

In figure 6 the chained Lin–Kernighan algorithm ($C^{\lambda}LK$) is compared with a multilevel version ($MLC^{\lambda}LK$) at several values of the intensity parameter, λ. In particular, the results compare $MLC^{\lambda}LK$ with $\lambda = 0, N/20, N/10, N/2, N$ against $C^{\lambda}LK$ with $\lambda = 0, N/10, N/5, N, 2N$. These intensity values were chosen on the basis that $MLC^{\lambda/2}LK$ has approximately the same runtime as $C^{\lambda}LK$ (justified below in section 5.3)

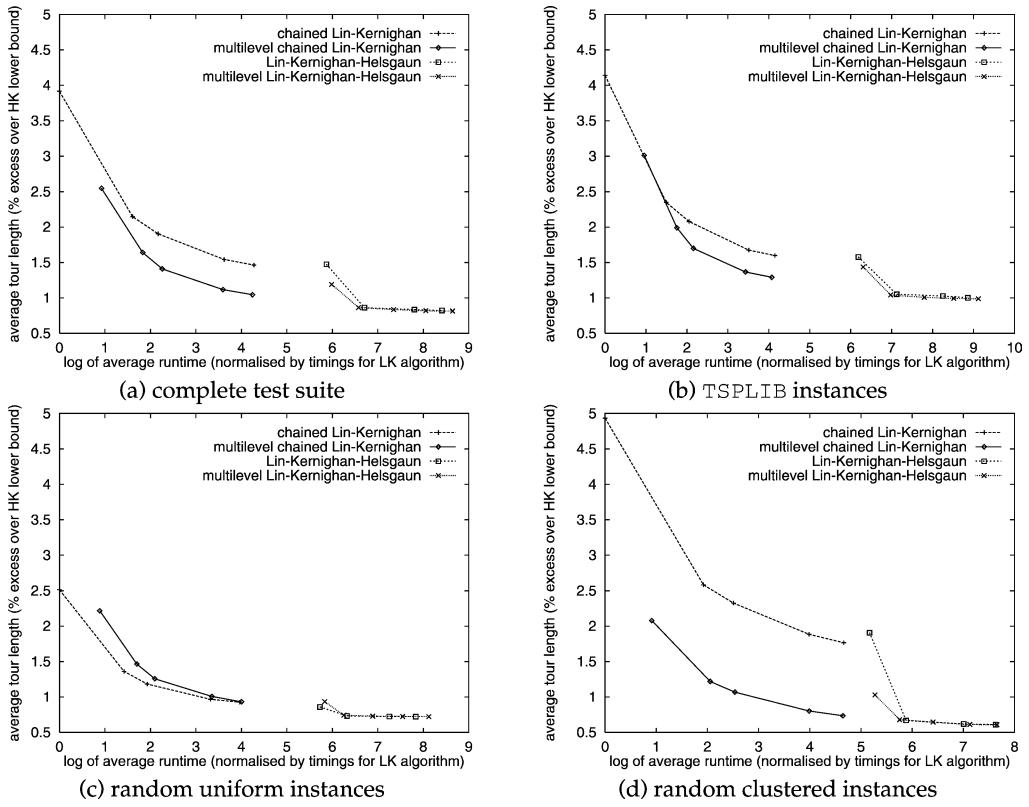

Figure 7. Plots of convergence behaviour for the TSP with logarithmic runtime scaling.

and because they are typical values for TSP experiments, e.g., (Johnson and McGeoch, 2002, 1997). In figure 6(a) this is illustrated graphically for the entire test suite by plotting the average percentage excess over the HKLB against average normalised runtime (i.e. the methodology described in section 1.3). The left-hand point on each curve corresponds to $\lambda = 0$ or in other words the standard LK algorithm and a multilevel version of it. The runtime factor of 2 is illustrated by the fact that each point of the MLC$^{\lambda/2}$LK curve is almost directly below underneath that for the C$^{\lambda}$LK curve. Most importantly figure 6(a) clearly shows the clear improvement in the asymptotic convergence of the solution quality. The multilevel technique brings the apparent asymptotic convergence down from around 1.4% for CLK to around 1.0% over the Held–Karp lower bound. Indeed because this is a *lower* bound on optimal solution quality the actual level for optimal solutions lies somewhat above 0% making this a more impressive reduction. Finally note that although the difference of just fractions of percentage points sounds insignificant, in fact schemes such as CLK are well known to be extremely effective and so it is not uncommon to make comparisons based on tenths or even hundredths of a percentage point, e.g., (Applegate et al., 1999; Helsgaun, 2000; Johnson and McGeoch, 2002, 1997).

The results in figure 6(a) are based on averaged results over all 80 test instances and so to explore them further we split the test suite into its three subclasses, TSPLIB instances (class I), random instance with uniform distribution (class II) and random instances with clustered distribution (class III), figures 6(b)–(d). This subdivision is highly illuminating. The TSPLIB instances, figure 6(b), with their wide range of examples, some from real-life applications, mirror fairly closely the results over the entire test suite with the multilevel version achieving considerably better asymptotic convergence. For the uniformly distributed random instances, figure 6(c), however the picture is completely different. The multilevel performance is actually slightly worse initially, presumably because of the additional runtime overhead, and although the asymptotic convergence looks to be about the same, it is clear that the multilevel process does not really contribute to the quality. Finally for the randomly clustered examples, figure 6(d), those which the CLK algorithm finds more difficult to optimise, the ability of the multilevel framework to aid the convergence is at its most dramatic. (In fact, looking at the raw data in (Walshaw, 2002), this is even more striking for some of the real life instances from TSPLIB.) CLK variants typically have great difficulty with these examples, e.g., (Applegate et al., 1999; Johnson and McGeoch, 1997), and indeed Neto has suggested modifications to the CLK algorithm to make it *cluster-aware* (Neto, 1999), although testing during the DIMACS challenge suggested that these modifications did not really work (Johnson and McGeoch, 2002). Nonetheless the multilevel variants find very good solutions and it seems likely that this is because the multilevel algorithm is good at regarding clusters as a single entity, or *mega-city*. In a high quality tour, typically a cluster will only have one inbound edge and one outbound. The algorithm can thus concentrate on getting these longer edges correct when it has a much simpler coarse representation of the problem and then sort out the tour details within the cluster later on.

As with the GPP, we measured the standard deviation, σ, to assess the performance variation and found a similar trend. Thus the CLK values for σ much worse than the multilevel version, in this case around 3–8 times larger (and again this is true even for the random uniform instances, where the average quality is very similar). This seems to indicate, as with the GPP, that the multilevel scheme is stabilising the performance of the local search, in this case CLK, in some way.

Raw runtime data for these tests can be found in (Walshaw, 2002) and online,[13] but as an example for the uniformly distributed instances, the runtime for MLC^{N}LK, which although not linear with problem size is certainly subquadratic, is around 9 seconds for instances with $N = 1{,}000$, 221 seconds for $N = 10{,}000$, 59 minutes for $N = 100{,}000$ and 16.5 hours for $N = 1{,}000{,}000$.

Overall then, at least for the instances tested, the multilevel framework seems to offer considerable benefits. On the more clustered examples it is able to dramatically enhance the CLK algorithm whilst for those instances with a uniform distribution, although it does not improve the performance, neither does it appear to hinder the optimisation significantly.

3.2.2. The Lin–Kernighan–Helsgaun algorithm

A multilevel version of the LKH algorithm is tested in (Walshaw, 2001c) and we present convergence plots in figure 7. However because the LKH algorithm exhibits runtimes which are quadratic in N, the problem size, it is impractical as a general solution strategy for large problems and we therefore reduced the test suite to include only those instances with $N < 20{,}000$. We also include the CLK results in this plot and, because of the extreme differences in runtime, plot the log of average normalised runtime. Thus figure 7 includes the same information as figure 6 (although on a reduced test suite) but the horizontal scaling is very different (and hence arguably less visually representative of typical behaviour with its long shallow decay curves).

Of course the fact that LKH^N does achieve extremely good (and often optimal) results for medium sized problems and this means that the capacity for improvement by the addition of multilevel techniques is severely limited (it is obviously impossible to improve on an optimal solution). Nonetheless, MLLKH did achieve convergence improvements over the LKH results and this is certainly evident for the single trial tests (the extreme left hand point of each LKH curve) in figures 7(a), (b), (d). It is also just discernible in figure 7(b) where MLLKH does show a slight improvement over LKH for the entire range of intensities. Otherwise the algorithms look to have very similar convergence behaviour although to pick a single statistic, for the 67 instances tested, the N trial version, LKH^N, found optimal solutions for 33 out of 59 instances with known optimal solutions whilst the multilevel version, MLLKH^N, found 39 out of 59 optimal solutions. Furthermore, because of the quadratic runtime and as predicted in (Walshaw, 2002, section 4.5) MLLKH^N only added around an average 18% time penalty over LKH^N rather than the 'factor of two' derived below (section 5.3) for linear refinement schemes. More interestingly, as for CLK, the convergence improvements are concentrated in the `TSPLIB` and randomly clustered subclasses.

The differences in the standard deviation, σ, of the results is similar to that seen for CLK only much less extreme. Thus LKH has typical σ values only around 20–30% worse than that of MLLKH (except for the single trial variants, LKH^1 and MLLKH^1 where it is 3–4 times worse). Again this seems to suggest that the multilevel framework stabilises the performance of LKH somewhat, but less so than for CLK.

Raw runtime data can be found in (Walshaw, 2001c), but as an example for the uniformly distributed instances, the runtime for MLLKH^N is around 105 seconds for instances with $N = 1{,}000$ and 9.5 hours for $N = 10{,}000$.

4. The graph colouring problem

The graph colouring problem (GCP) can be 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 is minimised. The GCP is well studied and has many applications including scheduling, timetabling and the solution of sparse linear systems, see, e.g., (Leighton, 1979; Lewandowski and Condon, 1996). However it is also often cited as one of the most difficult combinatorial optimisation problems, e.g.,

(Johnson and Trick, 1996). In fact, if we use $\chi(G)$ to denote the minimum number of colours required to colour a graph $G - \chi(G)$ is known as the *chromatic number* of G – then not only is the problem of finding $\chi(G)$ NP-hard (Garey and Johnson, 1979), but Lund and Yannakakis have even shown that, for some $\epsilon > 0$, approximate graph colouring within a factor of N^{ϵ} is also NP-hard (Lund and Yannakakis, 1994).

4.1. A multilevel algorithm for the graph colouring problem

Recently a multilevel algorithm has been introduced for the GCP (Walshaw, 2001b). As we have seen above multilevel algorithms require a good local search algorithm to refine the solution at each level and, although the GCP has not been generally viewed as a prime candidate for local search heuristics, nonetheless some success has been achieved in this area. For example, Hertz and de Werra (1987), and Glover, Parker, and Ryan (1996), have applied tabu search, Johnson et al. have looked at simulated annealing (Johnson et al., 1991), and Culberson, Beacham, and Papp have investigated the iterated greedy algorithm (Culberson, Beacham, and Papp, 1995; Culberson and Luo, 1996). In (Walshaw, 2001b) multilevel versions of two such approaches, tabu search and the iterated greedy algorithm, are investigated; here we summarise and discuss those results.

4.1.1. Multilevel framework

Since the problem is graph-based and the solution set-based, the coarsening and uncoarsening procedures can be implemented 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 of parent vertices in G_l with a child vertex in G_{l+1}. Figure 8 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.

Vertex matching. The matching procedure is also based on algorithms used in graph partitioning, e.g., (Hendrickson and Leland, 1995a; Walshaw and Cross, 2000), with one very important difference; rather than matching *neighbouring* vertices, matches are made between those that are *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 vertices are only allowed to match with neighbours of neighbours

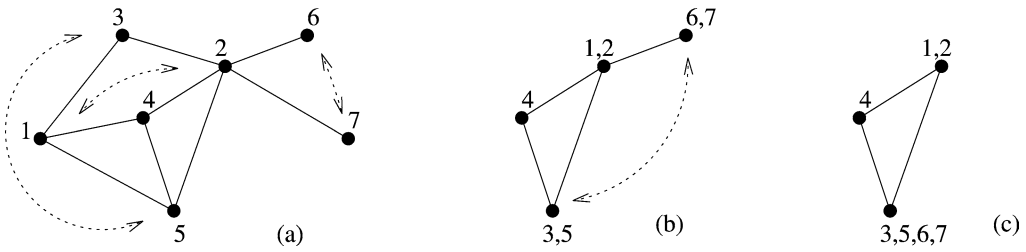

Figure 8. An example of graph contraction for the colouring problem.

rather than any non-adjacent vertex. Thus in figure 8(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). In set notation (using the definitions in section 1.2) a vertex v is allowed to match with any vertex in $\Gamma(\Gamma(v))$, or $\Gamma^2(v)$ for short, rather than any vertex in $V - \Gamma(v)$.

The matching algorithm used in (Walshaw, 2001b) is essentially the same as for many of the graph partitioning implementations, e.g., (Hendrickson and Leland, 1995a; Karypis and Kumar, 1998a; Walshaw and Cross, 2000). An ordering of the vertices is chosen 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 and 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. After considerable experimentation it appeared that for (a) initially sorting the vertices by decreasing degree (i.e. largest first) gave the best results. Meanwhile for (b) the priority which then seemed to work best was, for a vertex v, to pick u which maximised the number of neighbours common to both u and v. In the event that there were several such vertices, the one which minimised the number of distinct neighbours (i.e. those adjacent to either u or v but not both) was chosen and, in the event of a further tie-break, a random choice was made. These priorities also match choices made for other well known graph colouring algorithms (Walshaw, 2001b).

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 weights can be incorporated here, unlike partitioning they are not essential to represent the problem correctly and in (Walshaw, 2001b) they are suggested only as a possible future enhancement to the method (e.g., for aiding otherwise random decisions during refinement).

Termination. The coarsening process is 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 8(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 and some other possibilities are identified in (Walshaw, 2001b). Disconnected graphs (see section 1.2) are then best coloured by multilevel refinement on a component by component basis. Apart from making the termination criteria much simpler, this can also save time (Walshaw, 2001b).

Initialisation. The initial colouring is trivial (and optimal for the coarsest graph at least) if the final graph is complete since for such a graph, $G_l(V_l, E_l)$, every vertex must have a different colour and so $\chi(G_l) = |V_l|$.

Extension. As indicated above, the extension of a k-colouring for graph G_l to its parent G_{l-1} 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}.

4.1.2. Refinement: tabu search and the iterated greedy algorithm

Tabu search. Tabu search is a general technique, proposed by Glover (1986), 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 solutions 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 and de Werra proposed a tabu search algorithm for the graph colouring problem in (Hertz and de Werra, 1987). 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 and de Werra's algorithm works by generating a random neighbourhood 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.

Of course there is a strong possibility that no k_t-colouring will be found and so additional termination criterion are needed. In Culberson's implementation 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 (Culberson and Luo, 1996)) after λ iterations. In (Walshaw, 2001b) this is combined with a top down refinement search. Thus given a k_0-colouring, the target colour is set to $k_t := k_0 - 1$; every time a k_t-colouring is found, the target colour is set to $k_t := k_t - 1$ and the process is iterated until failure occurs.

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., (Christofides, 1975; Matula, Marble, and Isaacson, 1972). 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 (Christofides, 1975; Matula, Marble, and Isaacson, 1972)). 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 (Culberson and Luo, 1996), 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. 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 $1 \leqslant j < i$. However the converse does not hold and every vertex in C_j, for some $1 \leqslant j < i$, 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 (Culberson, Beacham, and Papp, 1995; Culberson and Luo, 1996). 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). 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.

4.2. Experimental results

We use tabu search and the iterated greedy algorithm as representative colouring schemes to illustrate the effectiveness of the multilevel approach. It is difficult to assess the state of the art in graph colouring as the most recent survey of heuristics dates back to the 2nd DIMACS implementation challenge of 1993. As with the 8th implementation challenge for the TSP (see section 3.2) this was aimed at characterising heuristics in the areas of the maximum clique, graph colouring and maximum satisfiability problems (Johnson and Trick, 1996). Unfortunately, there were few entrants in the colouring section and the editors suggest that this is 'due to its difficulty.' It is therefore hard to know what algorithm(s) represents the state of the art in colouring, although careful study of the papers in (Johnson and Trick, 1996), and in particular (Fleurent and Ferland, 1996),

does seem to indicate that tabu search is at least a contender. In fact it seems very likely that no one method is universally dominant and that the best methods are hand-tuned hybrids. For example some of the best results are found in (Fleurent and Ferland, 1996), where Fleurent and Ferland switch between tabu search and a clique finding algorithm used on the complement of the graph.

In (Walshaw, 2001b) the above procedures are tested on a large suite of 90 instances which consists of the examples compiled for the 2nd DIMACS implementation challenge (Johnson and Trick, 1996), augmented by further examples added since then.[14] As well as summarising the results from (Walshaw, 2001b) here we have extended them by including an additional set of experiments on the sparse test suite used for the GPP in section 2.2 (mirroring the testing of the partitioning algorithms on the colouring suite). Again the test methodology in section 1.3 was employed and to normalise the solution quality the results we use the best known solutions found either during the testing or taken from the literature (Johnson and Trick, 1996; Joslin and Clements, 1999). In fact optimal solutions are known for 66 out of the 90 instances in the colouring suite although the exact algorithms used to compute them do not scale to give feasible runtimes for the larger test cases, e.g., (Sewell, 1996). We used the greedy algorithm for runtime normalisation; as with partitioning (although an entirely different algorithm) this scheme is $O(|V|+|E|)$ in complexity thus capturing the problem size well. The intensity parameter was the number of failures to find a better solution and for the multilevel versions λ_l, the search intensity at level l, is set to $\lambda_l = \lambda/(l+1)$ where the original problem is level 0 and so $\lambda_0 = \lambda$.

4.2.1. Tabu search

Plots of convergence behaviour for the colouring test suites are shown in figure 9. Note that for each plot the limits of the y-axis have been chosen so that every data point is included but the performance behaviour for the left hand end of each curve, where it is obscured by the y-axis, is only visible in figure 10 (although this region of the curve is perhaps of less interest to practitioners – see section 1.3). Figure 9(a) shows the colouring results for the sparse test suite where we test TS^λ (with $\lambda = 2^m$ and $m = 0, \ldots, 10$) against MLTS^λ (with $\lambda = 2^m$ and $m = 0, \ldots, 9$) and, as can clearly be seen, MLTS appears to both converge faster and has better asymptotic convergence, although not dramatically so. Note that for this set of results the runtime factor of 2 (see section 5.3) does not hold true (i.e. it is not the case that the runtime for $\text{MLTS}^{\lambda/2}$ is approximately the same as that for TS^λ). Indeed for intensity $\lambda = 512$, the multilevel version (the final point on the MLTS curve) is much faster than the single-level version (the penultimate point on the TS curve) even though ostensibly MLTS should have more work to do. This is because MLTS manages to carry out most of the optimisation on higher levels, leaving the final level with an easier problem to optimise. However this observation should not be given too much weight since the current implementation of the colouring algorithms is not optimised for sparse graphs and the edges are stored in an adjacency matrix containing $|V|^2$ entries.

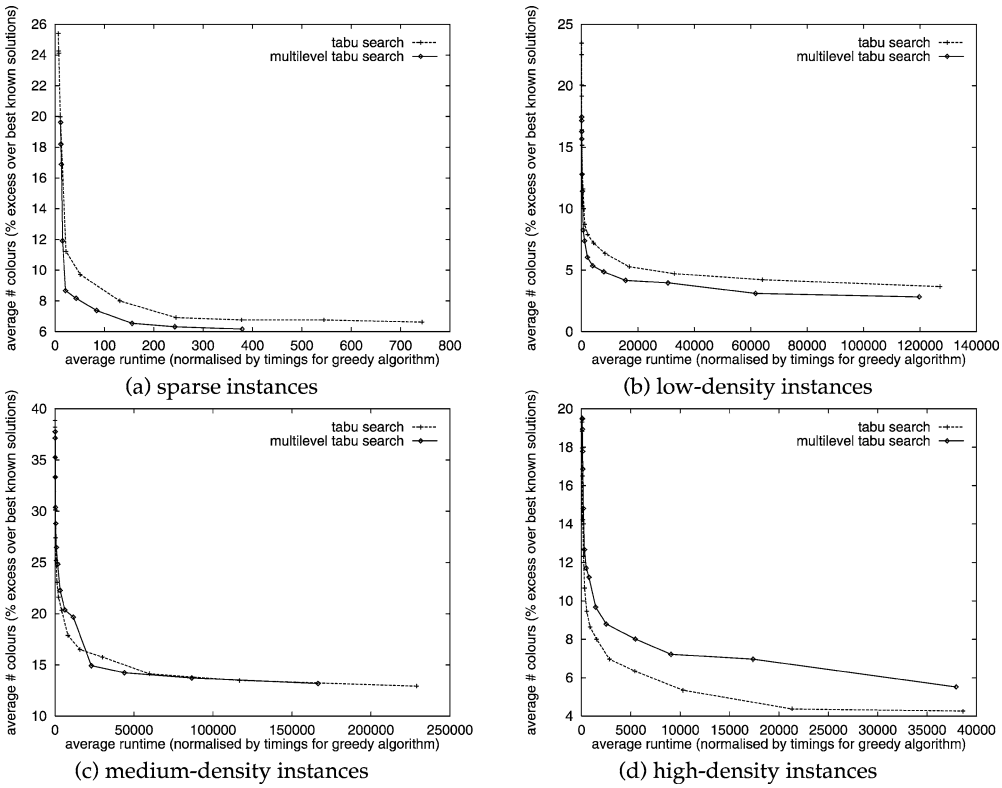

Figure 9. Plots of convergence behaviour for the GCP.

As in (Walshaw, 2001b) and in section 2.2 above, the colouring test suite is split into 3 subclasses based on edge density Δ: low ($0\% \leqslant \Delta \leqslant 33\frac{1}{3}\%$) with 58 out of 90 instances, medium ($33\frac{1}{3}\% < \Delta \leqslant 66\frac{2}{3}\%$) with 23 instances and high ($66\frac{2}{3} < \Delta \leqslant 100\%$) with just 9 instances. Figures 9(b)–(d) show the convergence behaviour on these subclasses and compare TS$^\lambda$ (with $\lambda = 2^m$ and $m = 0, \ldots, 15$) against MLTS$^\lambda$ (with $\lambda = 2^m$ and $m = 0, \ldots, 14$). The comparison is inconclusive on the medium-density test cases, figure 9(c), and both variants seem to be reaching approximately the same asymptotic convergence. Indeed, for the high-density examples in figure 9(d), the multilevel framework actually appears to hinder the colouring and MLTS has poorer convergence than TS. It is only for the low-density test cases, figure 9(b), that the multilevel version truly dominates and MLTS is slightly although distinctly better than TS. These results appear to be in line with a general trend established in the previous sections and we discuss the overall behaviour in section 6.1.

As for the GPP and TSP, we computed the standard deviation, σ, for each data point to measure the variation of performance. However, although at low intensities, the values of σ for TS (and IG below) are up to around 50% worse than MLTS (and MLIG), the asymptotic values of σ are very similar across all 4 methods indicating that none of them exhibit very different variations in performance.

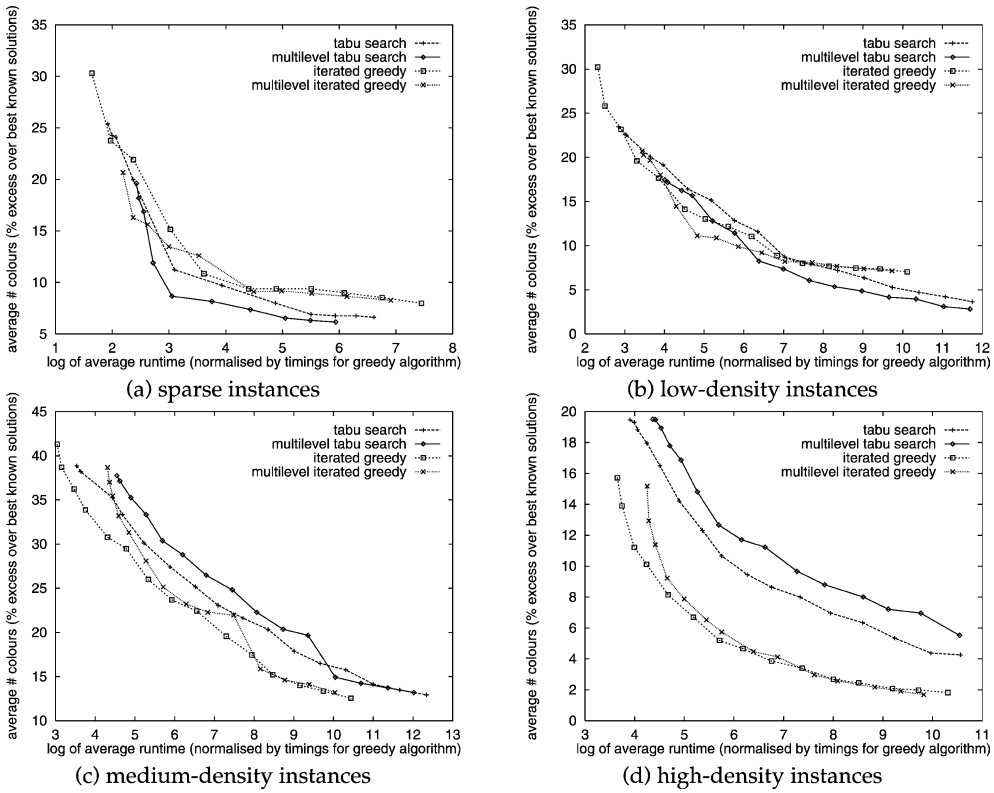

Figure 10. Plots of convergence behaviour for the GCP with logarithmic runtime scaling.

Finally, although we do not present raw data here, it can be found online[15] and a snapshot of the results can be found in (Walshaw, 2001b). To give typical examples though, for $\lambda = \lambda^* = 128$, the runtime for MLTS on a large sparse graph, 4elt, with $|V| = 15{,}606$ and $|E| = 45{,}878$ is around 962 seconds (although the methods are not optimised for sparse graphs). Meanwhile for the colouring suite and $\lambda = 2{,}048$, the random low, medium and high-density graphs, DSJC1000.1, DSJC1000.5 and DSJC1000.9, each with $|V| = 1{,}000$ and $|E| = 49{,}629$, $|E| = 249{,}826$ and $|E| = 449{,}449$, can be coloured in around 27, 26 and 24 seconds, respectively.

4.2.2. The iterated greedy algorithm

We augment the above results by running similar tests on the iterated greedy algorithm, this time testing IG$^\lambda$ (with $\lambda = 2^m$ and $m = 0, \ldots, 10$ for the sparse suite and $m = 0, \ldots, 14$ for the colouring suite) against MLIG$^\lambda$ (with $\lambda = 2^m$ and $m = 0, \ldots, 9$ for the sparse suite and $m = 0, \ldots, 13$ for the colouring suite). Once again, as in sections 2.2.2 and 3.2.2, we use logarithmic scaling for the runtime; although it does not give such a good visual representation of convergence this helps with the comparison of the two different local searches. It also elaborates the low intensity values shown in figure 9 for TS.

As can be seen from the plots, for the sparse and low-density instances, MLIG and IG both appear to reach the same asymptotic convergence (which was poorer than both TS and MLTS) although MLIG appears to converge to this limit faster than IG. For the medium and high-density subclasses IG and MLIG outperform both TS and MLTS. However although IG leads for low intensities, it is inconclusive as to which is better out of MLIG and IG as the convergence levels off and over most of the range tested (the right-hand half of the curves which, because of the logarithmic scaling, represents far more of the time taken than would appear).

Again to give some typical runtime data, for $\lambda = \lambda^* = 64$, the runtime for MLIG on 4elt, with $|V| = 15{,}606$ and $|E| = 45{,}878$ is around 438 seconds. For the colouring suite the random low, medium and high-density graphs, DSJC1000.1, DSJC1000.5 and DSJC1000.9 can be coloured in around 19, 46 and 111 seconds, respectively, for $\lambda = 1{,}024$.

5. Multilevel combinatorial optimisation

In this section we attempt to draw together the common elements of the examples in the previous sections. We give a possible explanation for the strengths of the multilevel paradigm and derive some generic guidelines for future attempts at other combinatorial problems.

5.1. The generic multilevel paradigm

As we have seen, 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 and then iteratively refined, usually with a local search algorithm, at each level in reverse order. Considered from the point of view of the hierarchy, a series of increasingly 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 $\mathcal{X}$. 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 either the partitioning or colouring problems that two vertices $u, v \in G_l$ are matched and coalesced into a single vertex $v' \in G_{l+1}$. When a refinement algorithm is subsequently used on G_{l+1} and whenever v' is assigned to a subset (or colour class), both u and v are also both being assigned to that subset. In this way the

matching restricts the refinement of G_{l+1} to consider only those configurations in the solution space in which u and v lie in the *same* subset, although the particular subset to which they are assigned is not yet specified. Similarly for the travelling salesman problem, when two vertices $u, v \in G_l$ are matched, the problem is restricted to consider only those tours in which u and v are adjacent, although their exact position in the tour is not yet specified. Since, in all 3 cases, many vertex pairs are generally coalesced from all parts of G_l to form G_{l+1} this set of restrictions is in some way a filtering of the solution space and hence the surface of the objective function.

This is an important point. Previously authors have made a case for multilevel schemes (and in particular partitioning) on the basis that the coarsening successively *approximates* the problem. In fact it is somewhat better than this; the coarsening *filters* the solution space by placing restrictions on which solutions the refinement algorithm can visit. Furthermore, an investigation of how the filtering actually performs for the GPP and TSP is carried out in (Walshaw and Everett, 2002) and it is shown that typically the coarsening filters out the higher cost solutions at a much faster rate than the low cost ones.

We can then hypothesise that, if the coarsening manages to filter 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 metaheuristic. 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. On a more pragmatic level this same process also allows the refinement to take larger steps around the solution space (e.g., for the GPP and GCP, rather than swapping single vertices, the local search algorithm can swap whole sets of vertices as represented by a single coarsened vertex).

Figure 11 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 filtered and smoothed (the filtration 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). Recursion of 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 filtration points for this example for which the steepest descent policy

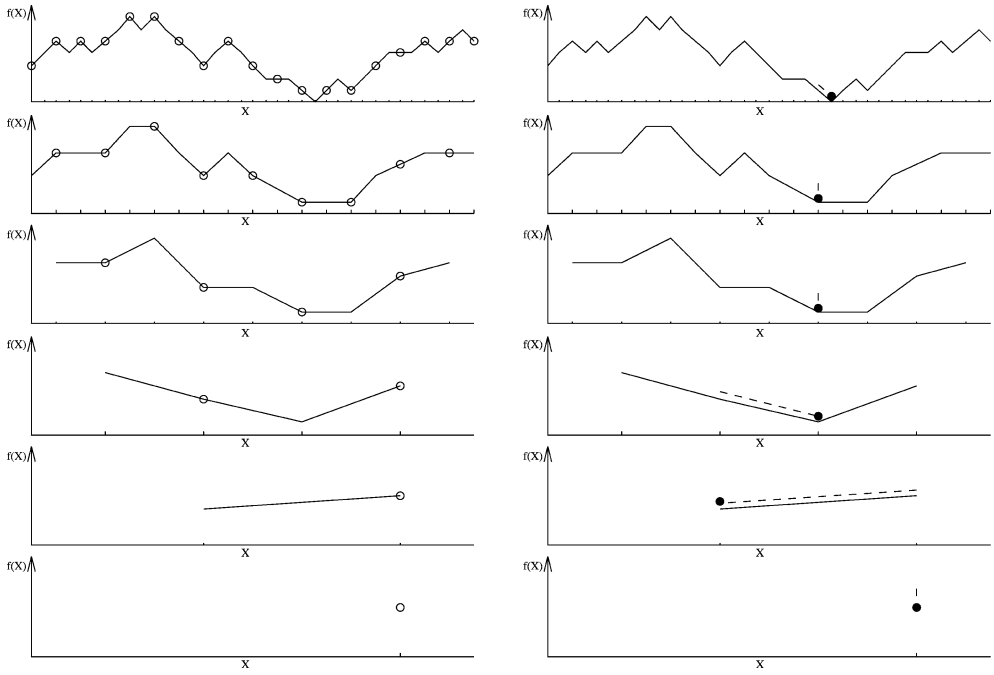

Figure 11. The multilevel scheme in terms of a simple objective function.

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 heuristics (such as repeated random starts combined with steepest descent local search, or even simulated annealing) applied to the same problem; without lucky initial guesses either might require many iterations to find the optimal solution.

To summarise the paradigm, multilevel optimisation combines a coarsening strategy together with a refinement algorithm (employed at each level in reverse order) to provide an optimisation metaheuristic. Figure 12 contains a schematic of this process in pseudo-code. Here P_l refers to the coarsened problem after l coarsening steps, C_l is a solution of this problem and C_l^0 denotes the initial solution.

5.2. Algorithmic requirements

Assuming that the above analysis does apply, how can a multilevel strategy be implemented for a given combinatorial problem? Here we discuss the algorithmic requirements illustrated by examples from the graph partitioning, graph colouring and travelling salesman problems (the GPP, GCP and TSP, respectively).

First of all let us assume that we know of a refinement algorithm for the problem, which refines in the sense that it attempts to improve on existing solutions. If no such refinement algorithm exists (e.g., if the only known heuristics for the problem are based

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

Figure 12. A schematic of the multilevel refinement algorithm.

on construction) it is not clear that the multilevel paradigm can be applied. 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 sophisticated techniques and examples of multilevel partitioning schemes exist for simulated annealing (Vanderstraeten et al., 1996), tabu search (Battiti, Bertossi, and Cappelletti, 1999; Vanderstraeten et al., 1996), genetic algorithms (Kaveh and Rahimi-Bondarabady, 2000), cooperative search (Toulouse, Thulasiraman, and Glover, 1999), and even ant colony optimisation (Langham and Grant, 1999). The refinement algorithm must also be able to cope with any additional restrictions placed on it by the coarsening (e.g., for the GPP coarsened graphs are always weighted whether or not the original is; for the TSP the refinement must not flip fixed edges in the coarsened levels).

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 suggest that the extension algorithm can be a simple and obvious reversal of the coarsening step which preserves the same cost. The initialisation is also generally a simple canonical mapping where by canonical we mean that a (non-unique) solution is 'obvious' (e.g., GPP – assign k vertices one each to k subsets; GCP – colour a complete graph; TSP – construct a tour to visit 2 cities) 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 three 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.

- Any solution in a coarsened space should have the same cost with respect to the objective function as its extension to the original space. This requirement ensures that the coarsening is truly filtering 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 and the examples (above and in section 5.5) suggest that it is very much problem-specific. 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 (e.g., Karypis and Kumar, 1998a; Walshaw et al., 1999). In this respect it seems likely that the most difficult aspect of finding an effective multilevel algorithm for a given problem and refinement scheme is the (problem-specific) task of devising the coarsening strategy.

5.3. Typical runtime

One of the concerns that might be raised by a newcomer to multilevel algorithms is that instead of having just one problem to optimise, the scheme creates a whole hierarchy of approximately $O(\log_2 N)$ problems (assuming that the coarsening approximately halves the problem size at each level). In fact, it is not too difficult to show that there is an approximate factor of two runtime between a local search algorithm at intensity λ, LS^λ, and the multilevel version, MLLS^λ. In other words, if $T(\text{A})$ denotes the runtime of algorithm A, then $T(\text{MLLS}^\lambda) \approx 2T(\text{LS}^\lambda)$.

Suppose first of all that the LS algorithm is $O(N)$ in execution time. Now suppose that the multilevel coarsening manages to halve the problem size at every step. This is an upper bound and in practice the coarsening rate is actually somewhat lower (e.g., between $5/8$ to $6/8$ is typical for the examples provided rather than the theoretic maximum of $1/2$) but experience indicates that typically this is not too far off. Let $T_L = T(\text{LS}^\lambda)$ be the time for LS^λ to run on a given instance of size N and T_C the time to coarsen and contract it. The assumption on the coarsening rate gives us a series of problems of size $N, N/2, \ldots, N/N$ whilst the assumption on LS^λ having linear runtime gives the total runtime for MLLS^λ as $T_C + T_L/N + \cdots + T_L/2 + T_L$. If λ is small then T_C can take large proportion of the runtime and so multilevel algorithms using purely greedy refinement policies (i.e. typically $\lambda = 0$) tend to take more than twice the runtime of the equivalent local search although this depends on how long the coarsening takes compared with the solution construction (typically both $O(N)$). However, if λ is large enough then typically $T_C \ll T_L$ and so we can neglect it giving a total runtime of $T_L/N + \cdots + T_L/2 + T_L \approx 2T_L$, i.e. MLLS^λ takes twice as long as LS^λ to run.

Furthermore, if the local search scheme is also linear in λ, the intensity, it follows from $T(\text{MLLS}^{\lambda}) \approx 2T(\text{LS}^{\lambda})$ that $T(\text{MLLS}^{\lambda}) \approx T(\text{LS}^{2\lambda})$. This effect is particularly visible for the TSP where the data points in figure 6 line up vertically. In this case the local search scheme, CLK, although not linear in N, is certainly subquadratic (Johnson and McGeoch, 1997), and more importantly is also linear in λ, which expresses the number of solution perturbations as a factor of N. For the GPP and GCP however, even where the local search schemes are linear (e.g., the iterated greedy algorithm for the GCP), λ expresses the number of failed iterations of some loop of the scheme and so this factor of two 'rule of thumb' breaks down. Nonetheless it does give a good 'ball-park' figure for the cost of a multilevel scheme.

Finally note that if the multilevel procedure is combined with an $O(N^2)$ or even $O(N^3)$ refinement algorithm then this analysis comes out even better for the multilevel overhead, i.e. $T(\text{MLLS}^{\lambda}) < 2T(\text{LS}^{\lambda})$, as the final refinement step would require an even larger proportion of the total (and this conclusion is backed up by experimental data for the LKH algorithm in figure 7).

5.4. Solution-based coarsening and iterated multilevel algorithms

We have seen (in section 2.1.3) an example of solution-based coarsening for use with an iterated multilevel partitioning algorithm. In fact this procedure can be easily generalised to the other problem areas. Thus, if a solution of a given problem already exists prior to optimisation it can be reused during the multilevel procedure to carry out solution-based coarsening by insisting that, at each level, every vertex matches with a candidate vertex that will not change the cost (e.g., for the TSP with one of its 2 neighbours in the existing tour and for the GCP with a vertex of the same colour). When no further coarsening is possible this will result in a solution for the coarsest problem with the same cost as the initial solution for the original problem. Once again, provided the refinement algorithm guarantees not to find a worse solution than the initial one the multilevel refinement can guarantee to find a new solution to the original problem with a cost no worse than the initial one. The multilevel process can then be iterated by using repeated coarsening/uncoarsening loops where at each iteration the best solution found previously is used to create a solution-based coarsening. A random element to the matching then means that then each iteration is likely to give a different hierarchy of graphs to previous iterations and hence allow the refinement algorithm to visit different solutions in the search space. We have not yet made any serious tests of this procedure for the TSP and GCP and indeed initial investigation has proved discouraging, however initial results for the GPP (section 2.2.3) indicate that it can sometimes be very helpful and further investigation is worthwhile.

5.5. Related work

Because multilevel algorithms are well-known in many other areas of mathematics there is a large body of literature which could be said to be related to the methods presented here. For interested readers a good start are the overview papers (Brandt, 1988; Teng,

1999). In particular however, an idea related in scope and design to the principles behind multilevel refinement is the search space smoothing scheme of Gu and Huang (1994). This uses recursive smoothing (analogous to recursive coarsening) to produce versions of the original problem which are simpler to solve. Thus in the example application Gu and Huang apply their technique to the TSP by forcing the inter-city edges to become increasingly uniform in length at each smoothing phase (if all edges between all cities are the same length then every tour is optimal). The obvious drawback is that each smoothing phase distorts the problem further (so that a good solution to a smoothed problem may not be a good solution to the original). In addition, the smoothed spaces are the same size as the original problem, even if the solution is potentially easier to refine, and hence may be equally as expensive to optimise. By contrast, multilevel coarsening filters rather than smoothing directly (although with the obvious drawback that the best solutions may be removed from the coarsened spaces) and so the coarsened spaces are smaller and hence can be refined more rapidly. It is also unclear whether search space smoothing is as general as coarsening and hence whether it could be applied to problems other than the TSP.

More specifically, multilevel schemes are starting to appear for other NP-hard combinatorial problems. For example, Boman and Hendrickson describe the use of a multilevel algorithm for reducing the envelope of sparse matrices (Boman and Hendrickson, 1996), a technique which aims to place all the non-zeroes as close as possible to the diagonal of a matrix and which can help to speed up the solution of sparse linear systems. They report good results, better than some of the commonly used methods, although they conclude that their scheme would probably be better if combined with a state-of-the-art local search algorithm. This conclusion is confirmed by Hu and Scott who have also developed a multilevel method for the same problem and which uses such a scheme, the hybrid Sloan algorithm, on the coarsest graph only (Hu and Scott, 2000). They report results which are of similar quality to the standalone hybrid Sloan algorithm (i.e. as good as the best known results) but which, on average, can be computed in half the time. Very recently Koren and Harel described the use of a multilevel algorithm for the linear arrangement problem (Koren and Harel, 2002), which has the aim of ordering the vertices of a graph so that the sum of edge lengths in the corresponding linear arrangement is minimised, a problem with several diverse applications. They report good results, similar in quality to the best known approaches, but with much better running times. In (Romeijn and Smith, 1999), Romeijn and Smith describe a parallel algorithm for approximately solving shortest path problems in large scale directed graphs. They only test the scheme using a 2-level (aggregation based) algorithm but prove a result on time complexity for a multilevel version.

In addition multilevel schemes have been applied to a number of variant partitioning problems including hypergraphs and problems with multiple constraints and/or costs, e.g., see (Hendrickson and Kolda, 2000; Schloegel, Karypis, and Kumar, 2004; Walshaw, 2001d). These provide further evidence for the flexibility of the paradigm.

6. Summary and future work

6.1. Review of empirical data

It is clear from the examples above that the multilevel paradigm can positively affect the results of local search algorithms, sometimes dramatically so – e.g., figures 2(a), 2(b) for the GPP and figure 6(d) for the TSP. However it is also clear that under certain conditions adverse effects can occur – e.g., figure 2(c) for the GPP and figure 9(d) for the GCP. Furthermore the ability of multilevel refinement to aid local search varies from problem class to problem class (e.g., it seems to be much easier to obtain improvements for the GPP and TSP than it is for the GCP).

After some consideration, and with the experience of many more results not presented here, we believe that it may be possible to characterise these effects by two features. Firstly it seems likely that the multilevel approach is better able to aid local search algorithms for problems in which the cost function has a *sensitive dependence on local conditions*. By this characterisation we mean that changing just a few elements of the solution local to some region will change the value of cost function, even if only a little and typically it occurs because the optimisation problem is intended to minimise a sum. This is certainly true for the GPP and TSP and tends to mean that the optimal solution of a given problem instance depends not only on some global quality in the solution, but also on local elements. We illustrate this in figure 13 for the GPP. The optimal bisection partition is shown in figure 13(a) with a cut-weight of 4. Two other partitions are shown in figures 13(b) and 13(c); however, whilst both have a cut-weight of 8, clearly figure 13(b) is globally poor but locally optimal, whilst figure 13(c) is close to the global solution but locally poor. We believe that through this dependence of the cost function on local conditions, the multilevel framework may offer the refinement algorithm a smooth transition through the problem levels whilst optimising the cost function.

By contrast the GCP does not have a very sensitive dependence on local conditions and, given a solution (especially a sub-optimal one) for a particular instance, is it often easy to change many elements of the solution without changing the value of the cost function. In other words the surface of the cost function has large plateau-like areas which render local search algorithms much more directionless – i.e. it is not easy to tell if changes in the solution will eventually result in a lower cost. This may mean that coarsened versions of the problem contain high quality solutions which are nonethe-

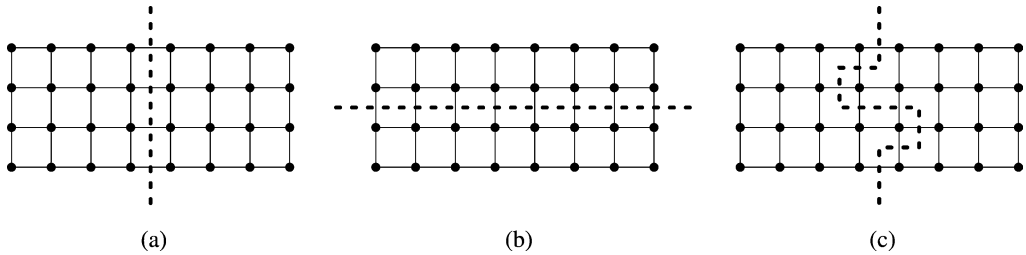

Figure 13. Example bisection partitions showing different global and local qualities.

less far away from the best solutions of the original problem and hence the multilevel techniques are not as helpful to the local search as they could be.

For a given problem class, the second characteristic in which may help in deciding whether a particular problem instance will be susceptible to multilevel refinement appears to be related to the density of that instance. For both the GPP and GCP, the multilevel framework applied to medium/high-density instances appears either to add no particular benefits or to actually hinder the local search. We believe that this may be because it is so unclear which vertices to match together (because there are so many possible candidates). As a result unfavourable matching may take place (i.e. between vertices which tend not to be in the same set in high quality solutions) and the filtering may remove the better solutions from the coarsened spaces. In this sense it is perhaps not strictly the density which determines whether or not a problem instance is likely to be susceptible to multilevel refinement, but whether the instance has inherent *matching affinities*. In other words, if it is easy to pick matches with good probability that the match will appear as part of a high quality solution then it is likely that multilevel refinement will be successful. The density is then a good indicator of inherent matching affinities with sparse and low-density instances providing the clearest matching choices.

Indeed this explanation can be extended to account for the behaviour of the multilevel TSP algorithm on random instances, figure 6(c), since matching priorities are not very clear for a uniform distribution of vertices. We can even compute a density indicator, given a Euclidean TSP instance of size N, by superimposing a grid of spacing $1/\sqrt{N}$, say. By counting the occupied cells we can then define the density Δ as the number of occupied cells, N_o, over the total number of cells, $\Delta = N_o/N$. Random instances with a uniform distribution will then typically have a high density with most of the cells occupied, whilst clustered instances, whether randomly generated or real-life instances, will typically have a much higher incidence of cells occupied by many vertices and hence a much lower density.

These conclusions are confirmed in (Walshaw and Everett, 2002) which investigates how the filtering of the solution spaces actually performs for the GPP and TSP. In particular it is shown that the coarsening manages to filter out the high cost solutions at a much faster rate than the low cost ones but that this process is inhibited as the density of the problem increases.

One further general observation that can be made from the computation of standard deviations is that the multilevel framework often seems to have the effect of stabilising the performance of the local search schemes. Thus for the GPP and TSP (and the GCP at low intensities) the multilevel versions appear to have much lower performance variation. Furthermore this is true even for those cases, e.g., figure 2(c), where the multilevel version produces worse average results than the original local search scheme.

Finally, the results also give a salutary warning to practitioners about the dependency of the computational experiments on the test suite. If we had just considered random, uniformly-distributed medium-density instances (as in some existing papers on both the TSP and the GCP) we could not have demonstrated that the multilevel framework offered any significant advantages. Clearly then algorithms need to be tested on as

broad a range of examples as possible and preferably on a suite which includes real-life instances.

6.2. Conclusions

To summarise then, it seems from the results presented and from additional sources of evidence (section 5.5), that the multilevel paradigm can aid local search algorithms to find better or faster solutions for certain combinatorial problems. Two of the problem classes tested, the GPP and GCP, require a solution which splits the vertices into subsets whilst the TSP requires a permutation of the vertices. Meanwhile two of the cost functions, those for the GPP and TSP, aim to minimise a sum whilst the GCP aims to minimise the number of sets. It therefore seems that the paradigm is fairly flexible. Within each problem class it appears that multilevel algorithms are best suited to low-density or sparse problems where the number of possible vertex matches are not overwhelming and that they probably give the best results for problem classes where the objective function has a sensitive dependence on local conditions. However it appears that, even for inappropriate problem classes, there may sometimes be modifications which render useful results such as the iterated multilevel algorithm for medium-density GPP problem instances, figure 4(b).

Although these restrictions might appear somewhat limiting, it is commonly acknowledged for combinatorial problems that often no one solution technique is appropriate for all instances, e.g., (Lewandowski and Condon, 1996). Moreover many problem instances from real-life applications fall into such classes and for example Leighton says, with reference to scheduling problems, that 'for most large-scale practical applications, the edge density of the graphs to be colored is generally small' (Leighton, 1979). Indeed the original success of multilevel partitioning came from the requirement to partition large, sparse, mesh-based problems. Furthermore, the results on appropriate problem classes can sometimes be spectacular, e.g., figures 2(a) and 6(d).

Overall this augments existing evidence that, although the multilevel framework cannot be considered as a panacea for combinatorial optimisation problems, it can provide a valuable, sometimes remarkably valuable, addition to the combinatorial optimisation toolkit. In other words, although no specific multilevel scheme dominates for a given problem, we hope to have convinced the reader that if they have a favourite local search algorithm for some problem, then it could be well worth considering the implementation of a multilevel version. We therefore believe that the multilevel framework is a good candidate for a new metaheuristic. Of course strictly speaking it is not new since multilevel techniques have been in use for many years; however they have not been widely applied to combinatorial problems. In addition it is somewhat different from other metaheuristics as it cannot be used in isolation and requires a refinement algorithm. On the other hand, neither is it in competition with them and existing multilevel versions of simulated annealing, tabu search, genetic algorithms and ant colony optimisation, indicate that it can even be used alongside them. In this way we believe

that multilevel refinement could have a real impact on many applications from discrete applied mathematics through to business decision support.

6.3. Future research

Clearly it is of great interest to further validate (or contradict) the conclusions of this paper by extending the range of problem classes. The sort of problems for which a multilevel approach appears attractive are of necessity those for which we seek only an approximate solution and probably those for which the optimisation is time critical. In particular, very large problems for which simulated annealing or population-based strategies such as genetic algorithms require excessive memory/CPU time seem appropriate. However it is important to bear in mind the lessons learned above and instances with no inherent sparsity are unlikely to be susceptible.

Obvious subjects for further work on the existing problem classes include the use of different refinement strategies such as simulated annealing or even genetic/evolutionary algorithms and the further investigation of iterated multilevel algorithms. Beyond this, some more specific topics are listed below.

6.3.1. Sparsification

A slightly disappointing feature of the multilevel partitioning and colouring algorithms, at least in the manifestations described here, was the sometime inability to aid the solution of medium/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 in the case of colouring the problem size can be reduced by picking one or more independent sets S_i from G and removing the vertices in S_i plus all edges incident on S_i. Whether or not this results in a sparsification of the problem depends 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 and DeWerra (1987), and by Fleurent and Ferland (1996), 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. However it is not so easy to imagine an analogous technique for partitioning.

6.3.2. 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 likely that finding a good matching (one which will aid the refinement) for such instances is more challenging because of the large number of candidates matches and the difficulty of choosing between them. This choice is usually made using some empirically-based prioritisation and hence inherent to the matching algorithms is a built in cost function (e.g., for the GPP – maximise the edge weight between matched pairs; for the TSP – minimise the distance between matched pairs; for the GCP – maximise the number of neighbours in common to matched pairs) which is not directly related

to the cost function of the problem in question. There is also plenty of evidence, for partitioning at least, that judicious choice of the matching cost function can strongly affect the final solution quality, e.g., (Karypis and Kumar, 1998a; Walshaw et al., 1999). However, it is not always clear which cost function the matching should be aiming to optimise (apart from maximising the number of matches) and for instance the heavy edge matching scheme (Karypis and Kumar, 1998a), commonly used for partitioning can only operate on weighted graphs and hence cannot be applied for the first coarsening step if the original graph is not weighted.

The computation of matchings of this type is known as the minimum-weight perfect matching problem where here the weight refers to the matching cost function. Polynomial algorithms which can compute optimal matchings are available, e.g., (Cook and Rohe, 1999), although they are usually too expensive to include in multilevel schemes. Nonetheless it might be of interest to apply such an algorithm and investigate the effect it has on solution quality.

A second possibility might be the sort of locally optimal matching scheme proposed by Monien, Preis, and Diekmann for graph partitioning (Monien, Preis, and Diekmann, 2000). Currently in all 3 example matching schemes used above, a vertex v_0 picks a preferred candidate v_1 and they are matched regardless of whether v_1 has available matches which are more advantageous than v_0. Hence, in the scheme of Monien, Preis, and Diekmann, 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.

6.3.3. Vertical refinement

For the multilevel processes described in this paper, including the iterated multilevel scheme, we can think of the refinement as *horizontal* in nature since the local search algorithms operate on one level at a time and terminate on a given level before moving on to the next level. However, in the course of writing this report we came across an intriguing paper by Brandt, one of the foremost practitioners of multilevel optimisation in all of its forms, and dating from 1988 (Brandt, 1988). Here Brandt proposes a multilevel annealing scheme for discrete-state problems. Unlike the work presented above, however, he suggests that the refinement algorithm (in this case a version of simulated annealing) should take a step on a level l only after calculating its effects on all finer levels. In other words the potential new state s_l on problem level P_l is projected down onto the next finest level, P_{l-1}, and the refinement looks for the nearest local minimum, s_{l-1} on P_{l-1} which is then projected down onto P_{l-2} and so on recursively; only if this produces a better solution on the original problem is the new state accepted. This type of *vertical refinement* is an intriguing idea and well worth more investigation. It may be more expensive and is not without difficulties; in particular using existing refinement software as a kind of black-box (as in sections 3 and 4) will be problematic. However, since the coarsening can never give a truly representative filtration of the objective func-

tion, each step is much more precise way of moving around the solution space than the methods described above.

Notes

1. At http://staffweb.cms.gre.ac.uk/~c.walshaw/data
2. At http://staffweb.cms.gre.ac.uk/~c.walshaw/partition
3. Available from http://mat.gsia.cmu.edu/COLOR/instances.html
4. Available from http://staffweb.cms.gre.ac.uk/~c.walshaw/partition
5. Available from http://staffweb.cms.gre.ac.uk/jostle
6. At http://staffweb.cms.gre.ac.uk/~c.walshaw/data
7. Available from http://www.cs.sandia.gov/~bahendr/chaco.html
8. At http://staffweb.cms.gre.ac.uk/~c.walshaw/data
9. See http://www.research.att.com/~dsj/chtsp/
10. Available from http://www.iwr.uni-heidelberg.de/iwr/comopt/software/TSPLIB95/
11. Available from http://www.keck.caam.rice.edu/concorde/download.html
12. From http://www.dat.ruc.dk/~keld/research/LKH
13. At http://staffweb.cms.gre.ac.uk/~c.walshaw/data
14. Available from http://mat.gsia.cmu.edu/COLOR/instances.html
15. At http://staffweb.cms.gre.ac.uk/~c.walshaw/data

References

Applegate, D., R. Bixby, V. Chvátal, and W.J. Cook. (1999). "Finding Tours in the TSP." Technical Report TR99-05, Dept. Comput. Appl. Math., Rice University, Houston, TX.

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

Battiti, R., A. Bertossi, and A. Cappelletti. (1999). "Multilevel Reactive Tabu Search for Graph Partitioning." Preprint UTM 554, Dip. Mat., University Trento, Italy.

Boman, E.G. and B. Hendrickson. (1996). "A Multilevel Algorithm for Reducing the Envelope of Sparse Matrices." Technical Report 96-14, SCCM, Stanford University, CA.

Brandt, A. (1988). "Multilevel Computations: Review and Recent Developments." In S.F. McCormick (ed.), *Multigrid Methods: Theory, Applications, and Supercomputing, Proc. of 3rd Copper Mountain Conf. Multigrid Methods*, Lecture Notes in Pure and Applied Mathematics, Vol. 110. New York: Marcel Dekker, pp. 35–62.

Bui, T.N. and C. Jones. (1993). "A Heuristic for Reducing Fill-In in Sparse Matrix Factorization." In R.F. Sincovec et al. (eds.), *Parallel Processing for Scientific Computing*. Philadelphia, PA: SIAM, pp. 445–452.

Christofides, N. (1975). *Graph Theory, an Algorithmic Approach*. London: Academic Press.

Cook, W.J. and A. Rohe. (1999). "Computing Minimum-Weight Perfect Matchings." *INFORMS J. Comput.* 11(2), 138–148.

Croes, G.A. (1958). "A Method for Solving Traveling Salesman Problems." *Oper. Res.* 6, 791–812.

Culberson, J.C., A. Beacham, and D. Papp. (1995). "Hiding Our Colors." In *CP'95 Workshop on Studying and Solving Really Hard Problems*, September, pp. 31–42.

Culberson, J.C. and F. Luo. (1996). "Exploring the k-Colorable Landscape with Iterated Greedy." In D.S. Johnson and M.A. Trick (eds.), *Cliques, Coloring, and Satisfiability*, DIMACS Series in Discrete Mathematics and Theoretical Computer Science, Vol. 26. Providence, RI: AMS, pp. 245–284.

Diekmann, R., R. Luling, B. Monien, and C. Spraner. (1996). "Combining Helpful Sets and Parallel Simulated Annealing for the Graph-Partitioning Problem." *Parallel Algorithms Appl.* 8, 61–84.

Farhat, C. (1988). "A Simple and Efficient Automatic FEM Domain Decomposer." *Comput. and Structures* 28(5), 579–602.

Fiduccia, C.M. and R.M. Mattheyses. (1982). "A Linear Time Heuristic for Improving Network Partitions." In *Proc. 19th IEEE Design Automation Conf.* Piscataway, NJ: IEEE, pp. 175–181.

Fleurent, C. and J.A. Ferland. (1996). "Object-Oriented Implementation of Heuristic Search Methods for Graph Coloring, Maximum Clique and Satisfiability." In D.S. Johnson and M.A. Trick (eds.), *Cliques, Coloring, and Satisfiability*, DIMACS Series in Discrete Mathematics and Theoretical Computer Science, Vol. 26. Providence, RI: AMS, pp. 619–652.

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

Gilbert, J.R., G.L. Miller, and S.-H. Teng. (1998). "Geometric Mesh Partitioning: Implementation and Experiments." *SIAM J. Sci. Comput.* 19(6), 2091–2110.

Glover, F. (1986). "Future Paths for Integer Programming and Links to Artificial Intelligence." *Comput. Oper. Res.* 13, 533–549.

Glover, F., M. Parker, and J. Ryan. (1996). "Coloring by Tabu Branch and Bound." In D.S. Johnson and M.A. Trick (eds.), *Cliques, Coloring, and Satisfiability*, DIMACS Series in Discrete Mathematics and Theoretical Computer Science, Vol. 26. Providence, RI: AMS, pp. 285–307.

Gu, J. and X. Huang. (1994). "Efficient Local Search With Search Space Smoothing: A Case Study of the Traveling Salesman Problem (TSP)." *IEEE Trans. Syst. Man and Cybernetics* 24(5), 728–735.

Held, M. and R.M. Karp. (1970). "The Traveling Salesman Problem and Minimum Spanning Trees." *Oper. Res.* 18, 1138–1162.

Helsgaun, K. (2000). "An Effective Implementation of the Lin–Kernighan Traveling Salesman Heuristic." *Eur. J. Oper. Res.* 126, 106–130.

Hendrickson, B. and T.G. Kolda. (2000). "Graph Partitioning Models for Parallel Computing." *Parallel Comput.* 26(12), 1519–1534.

Hendrickson, B. and R. Leland. (1995a). "A Multilevel Algorithm for Partitioning Graphs." In S. Karin (ed.), *Proc. Supercomputing '95, San Diego*. New York: ACM Press.

Hendrickson, B. and R. Leland. (1995b). "The Chaco User's Guide: Version 2.0." Technical Report SAND 94-2692, Sandia Natl. Lab., Albuquerque, NM, July.

Hertz, A. and D. de Werra. (1987). "Using Tabu Search Techniques for Graph Coloring." *Computing* 39, 345–351.

Hu, Y.F. and J.A. Scott. (2000). "Multilevel Algorithms for Wavefront Reduction." RAL-TR-2000-031, Comput. Sci. and Engrg. Dept., Rutherford Appleton Lab., Didcot, UK.

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

Johnson, D.S. and L.A. McGeoch. (1997). "The Travelling Salesman Problem: A Case Study." In E. Aarts and J.K. Lenstra (eds.), *Local Search in Combinatorial Optimization*. Chichester: Wiley, pp. 215–310.

Johnson, D.S. and L.A. McGeoch. (2002). "Experimental Analysis of Heuristics for the STSP." In *The Travelling Salesman Problem and its Variations*. Dordrecht: Kluwer Academic, pp. 369–443.

Johnson, D.S. and M.A. Trick (eds.). (1996). *Cliques, Coloring, and Satisfiability*, DIMACS Series in Discrete Mathematics and Theoretical Computer Science, Vol. 26. Providence, RI: AMS.

Joslin, D.E. and D.P. Clements. (1999). " "Squeaky Wheel" Optimization." *J. Artificial Intelligence Res.* 10, 353–373.

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

Karypis, G. and V. Kumar. (1998b). "Multilevel k-way Partitioning Scheme for Irregular Graphs." *J. Parallel Distrib. Comput.* 48(1), 96–129.

Kaveh, A. and H.A. Rahimi-Bondarabady. (2000). "A Hybrid Graph-Genetic Method for Domain Decomposition." In B.H.V. Topping (ed.), *Computational Engineering Using Metaphors from Nature, Proc. of Engrg. Comput. Technology*, Leuven, Belgium, Edinburgh: Civil-Comp Press, pp. 127–134.

Kernighan, B.W. and S. Lin. (1970). "An Efficient Heuristic for Partitioning Graphs." *Bell Syst. Tech. J.* 49, 291–308.

Koren, Y. and D. Harel. (2002). "A Multi-Scale Algorithm for the Linear Arrangement Problem." Technical Report MCS02-04, Faculty Maths. Comp. Sci., Weizmann Inst. Sci., Rehovot, Israel.

Langham, A.E. and P.W. Grant. (1999). "A Multilevel k-way Partitioning Algorithm for Finite Element Meshes using Competing Ant Colonies." In W. Banzhaf et al. (eds.), *Proc. Genetic and Evolutionary Comput. Conf. (GECCO-1999)*. San Francisco: Morgan Kaufmann, pp. 1602–1608.

Leighton, F.T. (1979). "A Graph Colouring Algorithm for Large Scheduling Problems." *J. Res. National Bureau Standards* 84, 489–503.

Lewandowski, G. and A. Condon. (1996). "Experiments with Parallel Graph Coloring and Applications of Graph Coloring." In D.S. Johnson and M.A. Trick (eds.), *Cliques, Coloring, and Satisfiability*, DIMACS Series in Discrete Mathematics and Theoretical Computer Science, Vol. 26. Providence, RI: AMS, pp. 309–334.

Lin, S. (1965). "Computer Solutions of the Traveling Salesman Problem." *Bell Syst. Tech. J.* 44, 2245–2269.

Lin, S. and B.W. Kernighan. (1973). "An Effective Heuristic for the Traveling Salesman Problem." *Oper. Res.* 21(2), 498–516.

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

Martin, O.C., S.W. Otto, and E.W. Felten. (1991). "Large-Step Markov Chains for the Traveling Salesman Problem." *Complex Systems* 5(3), 299–326.

Matula, D.W., G. Marble, and J.D. Isaacson. (1972). "Graph Coloring Algorithms." In R.C. Read (ed.), *Graph Theory and Computing*. New York: Academic Press, pp. 109–122.

Monien, B., R. Preis, and R. Diekmann. (2000). "Quality Matching and Local Improvement for Multilevel Graph-Partitioning." *Parallel Comput.* 26(12), 1605–1634.

Neto, D.M. (1999). "Efficient Cluster Compensation for Lin–Kernighan Heuristics." Ph.D. Thesis, Dept. Comp. Sci., University Toronto, Canada.

Pellegrini, F. and J. Roman. (1996). "SCOTCH : A Software Package for Static Mapping by Dual Recursive Bipartitioning of Process and Architecture Graphs." In H. Liddell et al. (eds.), *High-Performance Computing and Networking, Proc. HPCN'96*, Brussels, Lecture Notes in Computer Science, Vol. 1067. Berlin: Springer, pp. 493–498.

Reinelt, G. (1991). "TSPLIB – A Traveling Salesman Problem Library." *ORSA J. Comput.* 3(4), 376–384.

Romeijn, H.E. and R.L. Smith. (1999). "Parallel Algorithms for Solving Aggregated Shortest-Path Problems." *Comput. Oper. Res.* 26(10–11), 941–953.

Schloegel, K., G. Karypis, and V. Kumar. (2004). "Graph Partitioning for High Performance Scientific Simulations." In J.J. Dongarra et al. (eds.), *CRPC Parallel Computing Handbook*, to appear. Available from `http://www-users.cs.umn.edu/~karypis/publications/partitioning.html`

Sewell, E.C. (1996). "An Improved Algorithm for Exact Graph Coloring." In D.S. Johnson and M.A. Trick (eds.), *Cliques, Coloring, and Satisfiability*, DIMACS Series in Discrete Mathematics and Theoretical Computer Science, Vol. 26. Providence, RI: AMS, pp. 359–373.

Simon, H.D. (1991). "Partitioning of Unstructured Problems for Parallel Processing." *Computing Systems Engrg.* 2, 135–148.

Simon, H.D. and S.-H. Teng. (1997). "How Good is Recursive Bisection?" *SIAM J. Sci. Comput.* 18(5), 1436–1445.

Soper, A.J., C. Walshaw, and M. Cross. (2000). "A Combined Evolutionary Search and Multilevel Optimisation Approach to Graph Partitioning." Technical Report 00/IM/58, Comp. Math. Sci., University Greenwich, London, UK, April, to appear in *J. Global Optimization*.

Teng, S.-H. (1999). "Coarsening, Sampling, and Smoothing: Elements of the Multilevel Method." In M.T. Heath et al. (eds.), *Algorithms for Parallel Processing*, IMA Volumes in Mathematics and its Applications, Vol. 105. New York: Springer, pp. 247–276.

Toulouse, M., K. Thulasiraman, and F. Glover. (1999). "Multi-level Cooperative Search: A New Paradigm for Combinatorial Optimization and an Application to Graph Partitioning." In P. Amestoy et al. (eds.), *Proc. Euro-Par'99 Parallel Processing*, Lecture Notes in Computer Science, Vol. 1685. Berlin: Springer, pp. 533–542.

Vanderstraeten, D., C. Farhat, P.S. Chen, R. Keunings, and O. Zone. (1996). "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.

Walshaw, C. (2001a). "A Multilevel Algorithm for Force-Directed Graph Drawing." In J. Marks (ed.), *Graph Drawing, 8th Intl. Symp. GD 2000*, Lecture Notes in Computer Science, Vol. 1984. Berlin: Springer, pp. 171–182.

Walshaw, C. (2001b). "A Multilevel Approach to the Graph Colouring Problem." Technical Report 01/IM/69, Comp. Math. Sci., University Greenwich, London, UK, May.

Walshaw, C. (2001c). "A Multilevel Lin–Kernighan-Helsgaun Algorithm for the Travelling Salesman Problem." Technical Report 01/IM/80, Comp. Math. Sci., University Greenwich, London, UK, September.

Walshaw, C. (2001d). "Multilevel Refinement for Combinatorial Optimisation Problems." Technical Report 01/IM/73, Comp. Math. Sci., University Greenwich, London, UK, June.

Walshaw, C. (2002). "A Multilevel Approach to the Travelling Salesman Problem." *Oper. Res.* 50(5). (Originally published as University Greenwich Technical Report 00/IM/63.)

Walshaw, C. and M. Cross. (2000). "Mesh Partitioning: A Multilevel Balancing and Refinement Algorithm." *SIAM J. Sci. Comput.* 22(1), 63–80. (Originally published as Univ. Greenwich Technical Report 98/IM/35.)

Walshaw, C., M. Cross, R. Diekmann, and F. Schlimbach. (1999). "Multilevel Mesh Partitioning for Optimising Domain Shape." *Intl. J. High Performance Comput. Appl.* 13(4), 334–353. (Originally published as University Greenwich Technical Report 98/IM/38.)

Walshaw, C. and M.G. Everett. (2002). "Multilevel Landscapes in Combinatorial Optimisation." Technical Report 02/IM/93, Comp. Math. Sci., University Greenwich, London, UK, April.