Multilevel Refinement for Combinatorial Optimisation Problems

Chris Walshaw

Computing and Mathematical Sciences, University of Greenwich, Old Royal Naval College, Greenwich, London, SE10 9LS, UK.
Email: C.Walshaw@gre.ac.uk; URL: www.gre.ac.uk/∼c.walshaw

Mathematics Research Report : 01/IM/73

June 21, 2001

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 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. As a general solution strategy the multilevel procedure has been in use for many years and has been applied to many problem areas (most notably multigrid). 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 meta-heuristic. The results provide compelling evidence that, although the multilevel framework cannot be considered as a panacea for combinatorial problems, it can provide a 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; Meta-heuristic; 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 procedure 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). Survey papers such as [55] 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 (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 (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. [61, 62, 63, 64, 65]. 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 below, §2.2 & §2.3). More recently the paradigm was 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, [59]. 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, [58], and for graph colouring, [60].

In this paper, we aim to draw together some of this work, address the issue of multilevel refinement for combinatorial problems and, with the aid of examples in graph partitioning, graph colouring and the travelling salesman problem, make a case for its use as a meta-heuristic. In seeking to establish whether the multilevel paradigm can be used as a 'meta-heuristic', we use that term to mean a strategy which can be applied to enhance the performance of a local search algorithm. In this sense we are classing it alongside strategies where typically the meta-heuristic governs the 'large-scale' exploration of the state space and the local search algorithm finds the nearest (or at least a nearby) local minimum. In the case of simulated annealing this has been described as *chained local optimisation* or *large step Markov chains*, [41], whilst genetic algorithms which use this approach are sometimes called *memetic* algorithms, [44].

In this paper 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, under certain circumstances, impart a more 'global' quality to the final solution. We shall not define what we mean by global but allow experimental results to indicate that. The evidence is confined to results from three example problem areas, but within these areas it is compelling.
- 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.

1.1 Overview

The rest of the paper is organised as follows. In Sections 2, 3 & 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 optimisation problems, the multilevel techniques were not universally successful. However for graphs of low-density (in this case $\leq 30\%$) they were able to improve the convergence behaviour of two different local search algorithms, iterated greedy and tabu search. 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. Finally we summarise the findings in Section 6 and suggest some future research.

In writing this paper it was not entirely clear in which order to place these sections. In chronological order it should be Section 2 followed by Section 5 and then Sections 3 & 4, as the explanation and generalisation of the multilevel paradigm (Section 5) was originally derived from the multilevel partitioning examples (Section 2) and subsequently applied to derive multilevel approaches for the TSP (Section 3) in [58] and GCP (Section 4) in [60]. However since the generalisation is also informed and clarified by examples from the TSP & GCP, we have chosen to present the examples first and follow them with a generic overview.

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 either accelerate the convergence rate of the local search or even improve 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 typically we pick local search algorithms which are 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 (as in §2.1.2 & §4.1.2) in terms of the number of failures to achieve a better solution. 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

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

We then normalise these values with 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 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 (§2.2, §3.2 & §4.2). The time normalisation is more simple and we just calculate $\overline{T}_{\lambda,p}/T^{\mathrm{A}}_{1,p}$ where $T^{\mathrm{A}}_{1,p}$ is the average runtime on an instance p for some well known reference algorithm, A.

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

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

The tests for the examples discussed 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 §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.

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 processing (mesh partitioning). It is quite common in applications such as mesh partitioning to slightly relax the balancing constraint (i.e. that each set contains the same vertex weight) and admit partitions for which the largest set contains no more than $1 + \epsilon$ times the optimum weight for some small ϵ (e.g. $\epsilon = 0.03$). It is well known that this problem is NP-complete, [18], so in recent years much attention has been focused on developing suitable heuristics.

Typically many researchers have approached this problem by studying its restriction to 2 subdomains, 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. [52]. It is still widely used and is able to give guarantees on satisfying the balancing constraint, although the resulting partition quality may be limited, [53]. However with the advent of robust k-way partitioning algorithms, e.g. [31, 62], 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. [4, 6, 22, 29, 46] and k-way algorithms although here we illustrate the performance with the latter.

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 graph partitioning schemes, it was soon recognised as, more importantly, giving them a global perspective and has been successfully developed as a strategy for overcoming the localised nature of the Kernighan-Lin (KL), [34], and other optimisation algorithms. Typically such multilevel implementations match pairs of adjacent vertices to form *clusters*, use the clusters to define a new graph and recursively iterate this procedure until the graph size falls below some threshold. The coarsest graph is then partitioned (possibly with a crude algorithm) and the partition is successively refined on all the graphs starting with the coarsest and ending with the original. The multilevel idea was first proposed by Barnard & Simon, [3], as a method of speeding up spectral bisection and improved by both Hendrickson & Leland, [22] and Bui & Jones, [6], who generalised it to encompass local refinement algorithms. Several algorithms for carrying out the matching have been devised by Karypis & Kumar, [29].

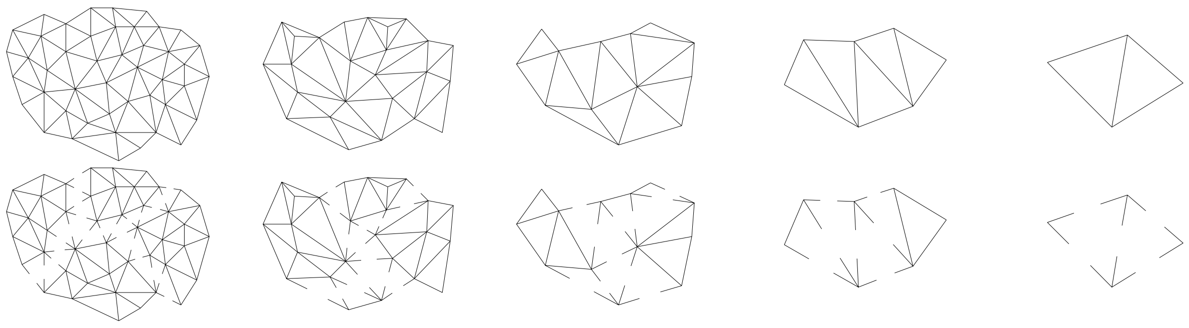


Figure 1: An example of multilevel partitioning

Figure 1 shows an example of the technique. On the top row the graph is coarsened down to 4 vertices. Notice that each coarsened graph approximates the previous one and this has often been used as a justification of why the multilevel paradigm is effective. The four vertices are then distributed to the 4 subdomains (bottom right) and the solution successively extended & refined (right to left). Although the refinement is only local in nature a good solution is still achieved.

2.1.1 Multilevel elements

Graph contraction. A common (although not universal) 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 & Leland, [22]. 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 graph weight remains the same, $||V_{l+1}|| = ||V_l||$, and the total edge weight is reduced by an amount equal to the weight of the collapsed edges.

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 & Kumar, [29], that it can be beneficial to the optimisation to collapse the most heavily weighted edges.

This simple but rapid algorithm of complexity $O(|E|)$ 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. In practice the fact that a suboptimal matching is found appears not to matter although we discuss this topic further in §6.2.2.

The initial partition. Having constructed the series of graphs until the number of vertices in the coarsest graph is smaller than some threshold, 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 coarsening so that the coarsest graph G_L still retains enough vertices, $|V_L|$, to achieve a balanced initial partition (i.e. so that typically $|V_L| \gg k$), [22, 29], or by incorporating load-balancing techniques alongside the refinement algorithm, e.g. [62]. Indeed Walshaw & Cross also demonstrate that relaxing the balance constraint on the coarser levels and tightening it up gradually can enhance the resulting partition quality, [62].

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, [31], and is extremely fast.

A more sophisticated class of method is based on the Kernighan-Lin (KL) bisection optimisation algorithm, [34], which includes limited hill-climbing to enable it to escape from local minima. This has been extended to k-way partitioning by several authors (e.g. [22, 31, 62]) and recent implementations almost universally use the linear time complexity improvements (e.g. bucket sorting of vertices) due to Fiduccia & Mattheyses, [15]. We outline the KL refinement algorithm to illustrate the process, however in principle any iterative refinement scheme can be used and examples of multilevel implementations exist for simulated annealing, [57], tabu search, [4, 57], and even genetic algorithms, [33].

A typical KL 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 optimal 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 provides the user specified intensity of the search, λ, i.e. the maximum number of continuous failed iterations of the inner loop. 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 optimal cost was attained.

Variants of this algorithm, together with conditions for vertex transfer acceptance and confirmation are described in e.g. [22, 31], and the particular version used for the results here is described in [62].

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 (see above). 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. [62]).

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 sort of technique is frequently used in graph partitioning for dynamic load-balancing, e.g. [49] although because the initial partition is usually unbalanced the quality guarantee is lost. However it can also be used to find very high quality partitions, albeit at some expense and we can iterate the multilevel procedure by using repeated coarsening & uncoarsening. 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 is done randomly, each iteration is very likely to give a different hierarchy of graphs to previous iterations and hence give the possibility for the refinement algorithm to visit different solutions in the search space.

We refer to this process as an *iterated multilevel algorithm* (see also [56] for a variation of this technique). 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 however, we choose λ so that it is close to the turning point in the gradient of a typical MLKL curve provided by the experimentation below (because this should give the best return in solution quality for the least runtime cost). We can then vary γ, the outer intensity parameter, to test the iterated multilevel algorithm and below we give some sample results for this scheme (which has also been used to find very high quality partitions[1] for benchmarking purposes).

[1]collected at `http://www.gre.ac.uk/~c.walshaw/partition`

2.2 Experimental results

To illustrate the potential gains that the multilevel paradigm can offer we conducted a number of experiments on example graphs. These are not meant to be exhaustive in any way but merely give an indication of typical performance behaviour.

We decided to use two test suites, one of which is a smallish collection of 16 sparse, mostly mesh-based graphs drawn from a number of real-life applications. These sparse instances are collected together with some larger examples[1] for benchmarking partitioning algorithms, [54]. We used the smaller problem instances here, all those with $|V| < 20,000$, because we also wished to use the same test set for graph colouring experiments, §4.2.1, and currently the implementation of the colouring schemes that we use is coded in terms of an adjacency matrix. Thus instances with $|V| > 20,000$, and hence a matrix with $> 400,000,000$ entries, start to become unmanageable. However the results are indicative only and more thorough testing of this sort of problem instance, including much larger examples, can be found in e.g. [22, 29, 62].

The other test suite consists of 90 instances compiled to test graph colouring algorithms for the 2nd DIMACS implementation challenge, [27], augmented by further examples added since then[2] 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 [60] 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 [60], into 3 density classes; low ($0\% \leq \Delta \leq 33\frac{1}{3}\%$) with 58 out of 90 instances, medium ($33\frac{1}{3}\% < \Delta \leq 66\frac{2}{3}\%$) with 23 instances and high ($66\frac{2}{3} < \Delta \leq 100\%$) 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 test suite contains a series of randomly generated graphs of different sizes but fixed density of around 0.1).

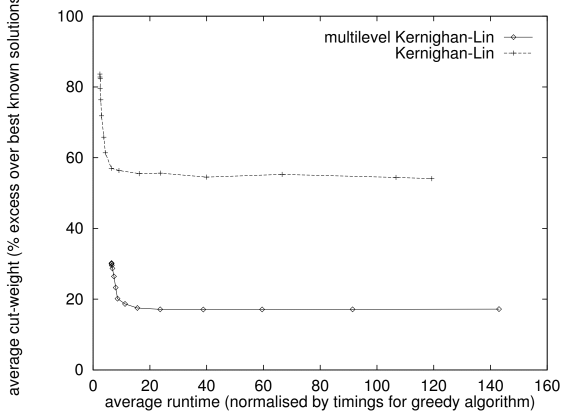
The tests compare the Kernighan-Lin (KL) algorithm (with an initial partition provided by the greedy algorithm, [14]) against a multilevel version (MLKL) at a range of intensities (see §1.3). The intensity parameter was the number of failed iterations of the inner loop (see §2.1.2) and for the multilevel 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 $\lambda_0 = \lambda$. The solution quality normalisation is against the best known solutions found to date. All tests on both suites were required to look for 16-way partitions and were allowed an imbalance of 3%; again this is indicative only and more thorough testing, at least for sparse graphs, can be found elsewhere. The algorithms used are available within the framework of JOSTLE, a mesh partitioning software tool developed at the University of Greenwich and freely available for academic and research purposes under a licensing agreement[3].

2.2.1 Sparse test suite

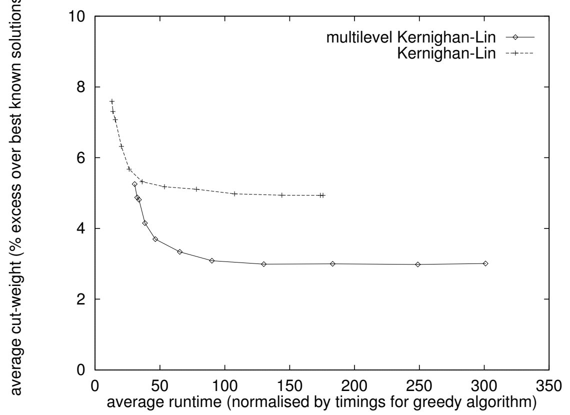
Using the methodology described in §1.3 we conducted tests to compare KL$^\lambda$ (with $\lambda = 0$ and $\lambda = 2^m$ with $m = 0, \ldots, 14$) against MLKL$^\lambda$ (with $\lambda = 0$ and $\lambda = 2^m$ with $m = 0, \ldots, 13$). Figure 2(a) shows the results for the sparse test suite and the dramatic improvement in quality 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 above §2.1.2) the MLKL solution quality is far better than KL and it is results like these that have helped to promote multilevel partitioning algorithms to the status they enjoy today.

The curves also illustrate two further points, true for all the plots in Figure 2 and related to the fact that the intensity controls the number of iterations of the inner loop rather than the outer loop (this is possibly

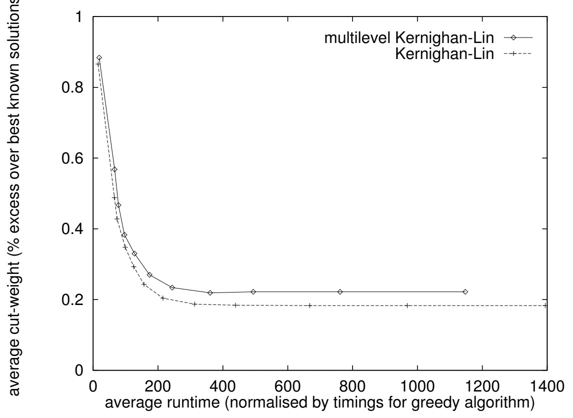
[2]available from `http://mat.gsia.cmu.edu/COLOR/instances.html`
[3]available from `http://www.gre.ac.uk/jostle`



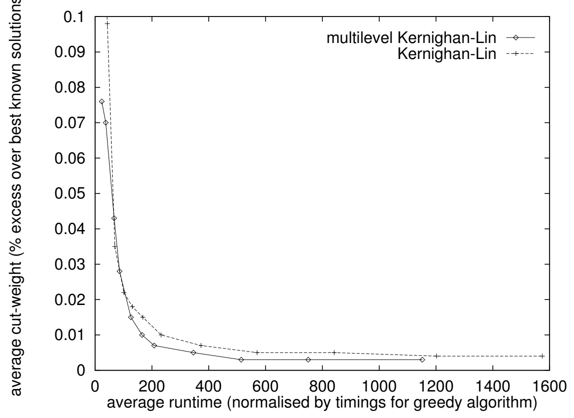
(a) sparse instances



(b) low-density instances



(c) medium-density instances



(d) high-density instances

Figure 2: Plots of convergence behaviour for the partitioning test suites

a minor deficiency of the implementation). Firstly there is an upper limit to the intensity and increasing λ eventually ceases to have any effect even on the runtime. This is because the number of iterations of the inner loop is limited to $|V|$, the number of vertices, and so for KL any value of $\lambda > |V|$ will give the same solution and runtime (and similarly for MLKL if $\lambda_l > |V_l|$ at every level l). This effect manifests itself on the plot by the final points on each curve being close together (e.g. the KL curve in Figure 2(b)) and indeed using larger values of λ would simply add a cluster of points at, or close to, the end of each curve.

Secondly, if the intensity did control the outer loop we would expect, for KL at least, that the curve should decrease monotonically. This is because for $\lambda_1 < \lambda_2$, a test at intensity λ_2 typically just extends 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, as can be seen, the quality does sometimes degrade slightly as the intensity increases.

2.2.2 Colouring test suite

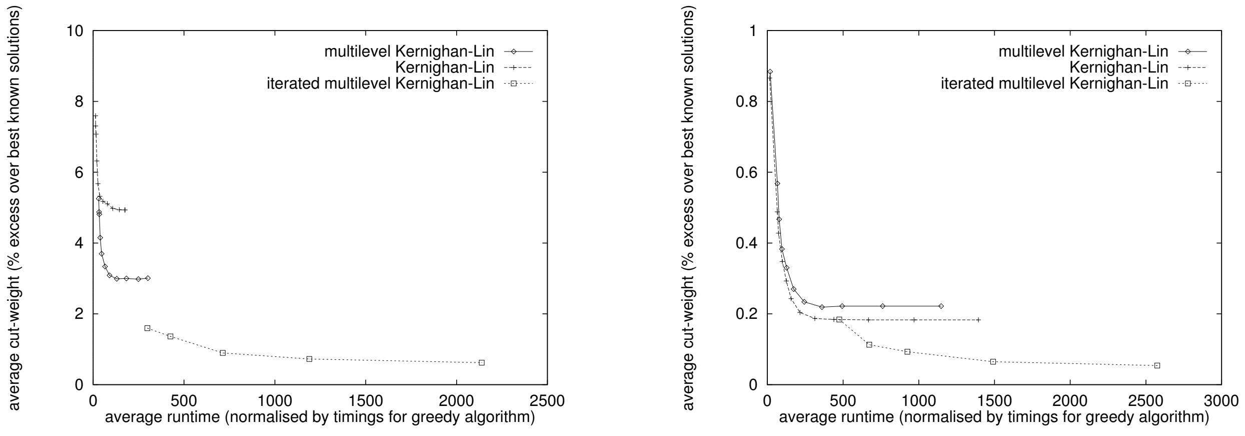
Figures 2(b)-(d) show the partitioning results for the colouring test suite. Here we test KL$^\lambda$ (with $\lambda = 0$ and $\lambda = 2^m$ with $m = 0, \ldots, 10$) against MLKL$^\lambda$ (with $\lambda = 0$ and $\lambda = 2^m$ with $m = 0, \ldots, 9$). Figure 2(b) more or less confirms the conclusions for the sparse results and although the curves are closer together, MLKL 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), MLKL is still the leading algorithm, although only marginally. However for the medium-density results, Figure 2(c), MLKL fails to achieve the same performance as KL and the multilevel framework appears to actually hinder the optimisation. For now we leave this as an observation but discuss it further in §6.1.

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.

2.2.3 Iterated multilevel results

Figure 3 illustrates the results for the iterated multilevel algorithm (IMLKL) described in §2.1.3 alongside the MLKL and KL results for low and medium-density subclasses of the colouring suite. These plots contain the same information about MLKL & KL as Figures 2(b) & 2(c) only it is more compressed here because of the long IMLKL runtimes. For the IMLKL results the inner intensity parameter was fixed at $\lambda = 64$ (for reasons explained in §2.1.3) whilst the iterated intensity parameter, γ, is varied with $\gamma = 2^l$ and $l = 0, \dots, 4$.



(a) low-density instances (b) medium-density instances

Figure 3: Plots of convergence behaviour including iterated multilevel partitioning results

We do not show results for the sparse and high-density instances because they are not so interesting. For the sparse suite IMLKL more or less continues the MLKL curve in Figure 2(a) with a few percentage points improvement and very shallow decay whilst for the high-density instances IMLKL does not appear to offer much improvement at all. However for the low and medium-density subclasses, in Figures 3(a) & 3(b) respectively, the asymptotic performance offered by IMLKL is impressive and worthy of further and more thorough investigation. In both cases IMLKL dramatically improves on MLKL and, for the medium-density instances, even appears to overcome the shortcomings of MLKL and exceeds the KL results.

2.3 Variant problems

A number of multilevel partitioning variants have been successfully applied to related problems and it is worth mentioning a few here to illustrate certain features of the paradigm as well as the flexibility of the approach. For example in [63] Walshaw & Cross describe the modifications required to map a k-way partition of a graph onto a k vertex complete graph with edge weights so that the mapping cost is minimised. This is a generalisation of the quadratic assignment problem and is useful for mapping a graph onto a parallel computer with heterogeneous communications links (a successful mapping is then one in which adjacent subdomains generally lie on 'adjacent' processors). In fact the coarsening algorithm is left unchanged and the cost function is optimised solely by simple changes to the gain function. This makes an interesting point; the ***global*** layout of the final partition can be radically changed using a multilevel algorithm just by changing the cost function in a ***local*** search scheme. We see similar effects in [64] for mesh partitioning where the aspect ratio (shape) of the subdomains is optimised, again by simple changes to the cost function and, in this case, the matching scheme. Meanwhile in [61] a dynamic mesh partitioning variant is described which, for the case of adaptively refined meshes, aims to minimise both the cut-weight and the changes to the partition (so that for parallel processing most of the data stays in place and does not need to migrate to another processor). Interestingly here, the earlier the coarsening is terminated, the smaller the global changes to the partition (although with a corresponding fall-off in partition quality). This evidence strongly suggests that the multilevel framework adds some sort of global perspective (or 'global view' in [29]) to partitioning schemes.

Other interesting multilevel partitioning variants include variant schemes which can optimise multiple objective function and/or multiple balancing constraints, [30, 50, 65] (although sometimes these are mutually incompatible and so a graceful trade-off is sought). Also multilevel partitioning algorithms have been extended to hypergraphs (a generalisation of a graph in which a hyperedge connects an arbitrary number of vertices as opposed to the two connected by an edge in a classical graph), [32]. Again this all adds evidence to the flexibility of the multilevel paradigm, at least in the field of graph partitioning.

3 The Travelling Salesman Problem

Recently the multilevel paradigm has been applied to the travelling salesman problem (TSP), [58], which 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, [17], 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 optimisation problem. For example Johnson & McGeoch, [26], survey a wide range of approaches which run the gamut from local search, through simulated annealing, tabu search & genetic algorithms to neural nets. Remarkably, and despite all this interest, the local search algorithm proposed by Lin & Kernighan in 1973, [38], still remains at the heart of the most successful approaches. In fact Johnson & 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 & Felten, [40], in 1991.

Even today, in spite of all the work on exotic and complex combinatorial optimisation techniques, Johnson & McGeoch, [26], conclude that an iterated or chained Lin-Kernighan (ILK/CLK) scheme provides the highest quality tours for a reasonable cost. In fact it ***is*** usually possible to improve on the quality of (suboptimal) CLK/ILK tours, for example by sophisticated tour merging techniques similar to genetic algorithm crossovers (see e.g. [1]), but Johnson & McGeoch suggest that 'the ILK variant . . . , is the most cost effective way to improve on Lin-Kernighan, at least until one reaches stratospheric running times'.

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 & Karp, [21], 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 (especially if some or all of them do not have known optimal tours), 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 §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

A multilevel TSP algorithm is perhaps not so intuitive as that for the GPP. Clearly the LK or CLK/ILK algorithms should make a good refinement method although in principle any iterative refinement procedure including the well-known 2-opt, [9], and 3-opt, [37], algorithms could be used. However, with no graph as such, how can the problem be coarsened?

In fact from [58] 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 (even if it is a poor solution). 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.

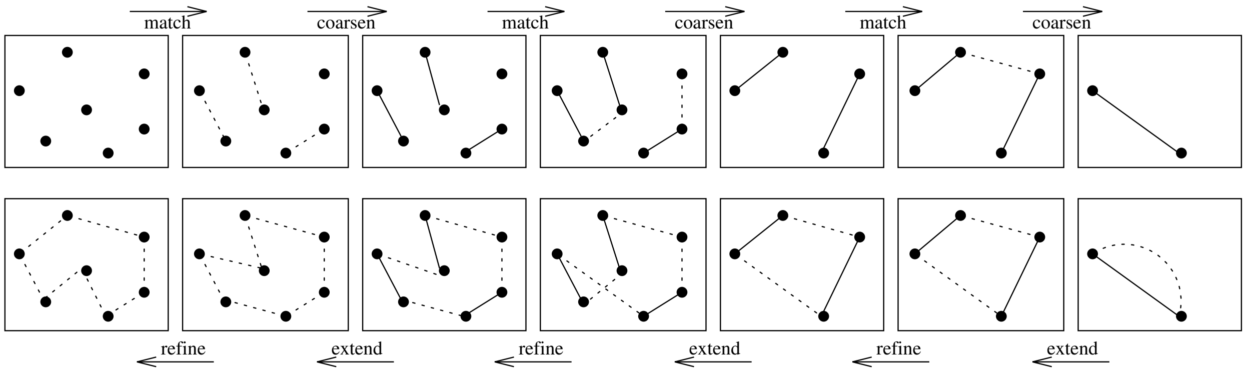


Figure 4: An example of a multilevel TSP algorithm at work

Figure 4 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 & 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 trivial; 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 less possibilities for being flipped.

3.1.1 Multilevel elements

Matching and coarsening. The implementation of this matching & coarsening process is fully described in [58]; vertices are matched and edges fixed between them. 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). Figure 5(a) shows an example of this where the edges (v_1, w_1) & (v_2, w_2) are matched together. 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.

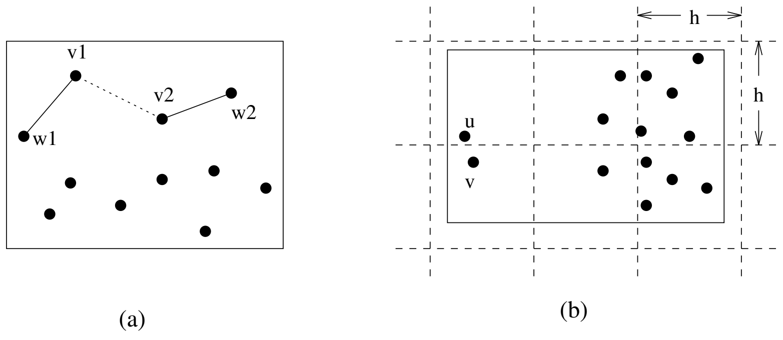


Figure 5: TSP matching examples

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. For example, consider Figure 5(b); it is difficult to imagine an optimal tour which does not include the edge (u, v) and so ideally the matching should fix it early on in the process. Indeed, if by some good fortune, the matching ***only*** selected optimal edges then the optimal tour would be found by the end of the matching process and the refinement would have no possible improvements. However, in the absence of any other 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 (e.g. Figure 5(b)) to avoid $O(N)$ searches to find the nearest neighbours.

An important implementation detail 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 not only aids fast searches for nearby vertices but also prevents long-range matching on the lower levels of the coarsening. This appears to have an important effect on the results (see §3.2).

Initialisation. The coarsening ceases when further coarsening 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 (the other edge of the tour being the fixed one).

3.1.2 Refinement: the (chained) Lin-Kernighan algorithm

The multilevel TSP algorithm described in [58] uses the chained Lin-Kernighan algorithm for the refinement step of our multilevel procedure, probably the most successful local search technique for iteratively optimising a TSP tour.

Typically TSP tour optimisation take places by 'flipping' edges. For example, if the tour contains the edges (v_1, w_1) & (w_2, v_2) in that order, then these two edges can always be flipped to create (v_1, w_2) & (w_1, v_2). This sort of step forms the basis of the 2-opt algorithm due to Croes, [9], 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, [37], exchanges 3 edges at a time. The Lin-Kernighan (LK) algorithm, [38], 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 much on its efficient implementation together with some additional ideas to improve its quality, and for an excellent overview of techniques see the survey of Johnson & McGeoch, [26]. For more details of the particular implementation used in §3.2 see [1, 2].

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 & Felten's important contribution to the field, [40, 41], 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. [1, 2, 24, 26]. The version used here takes this approach and is known as the chained Lin-Kernighan (CLK) algorithm.

Fixed edges. The only change to the implementation of the CLK algorithm required for the multilevel version in [58] is to ensure that none of the fixed edges are exchanged. 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.

3.2 Experimental results

The multilevel strategy is tested in detail in [58] and we summarise the results here. The experiments were carried out on a test suite of 79 TSP problem instances (a large subset of the 89 instances compiled for the 8th DIMACS implementation challenge[4] which is aimed at characterising approaches to the TSP). The instances are in three groups:

(I) All 33 symmetric and geometric instances of 1,000 or more vertices from `TSPLIB`[5], a collection of sample TSP instances, including some from real-life applications, compiled by Reinelt, [47, 48].

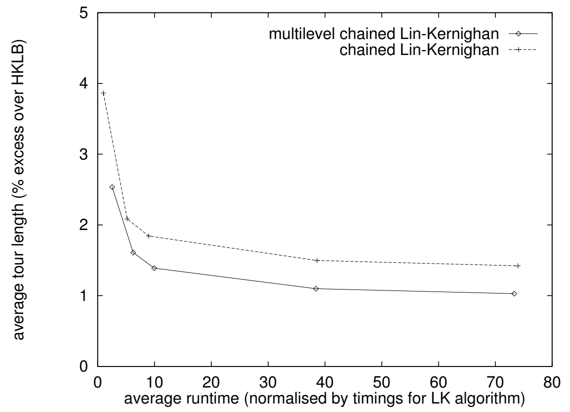
(II) 24 randomly generated instances with uniformly distributed vertices. These range in size from 1,000 to 1,000,000 vertices, going up in size gradations of $\sqrt{10}$ and were constructed by Johnson, Bentley & McGeoch specifically to study asymptotic behaviour in tour finding heuristics.

(III) 22 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.

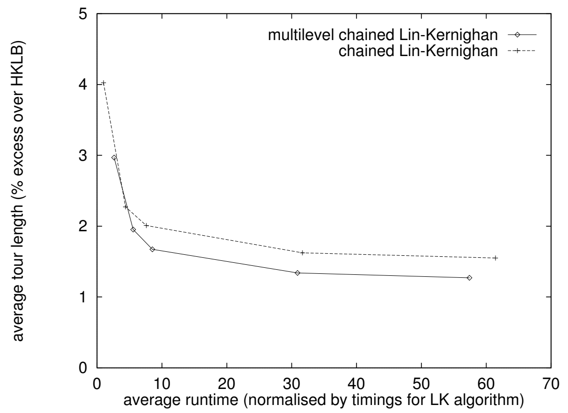
The CLK software is contained in an optimisation package written by Applegate *et al.*, [1], and known as `concorde`[6]. The intensity parameter, λ, (see §1.3) was chosen as the number of outer iterations or chainings expressed as a fraction of N the problem size, i.e. $\lambda = xN$ for some factor 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.

In [58], 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 λ =

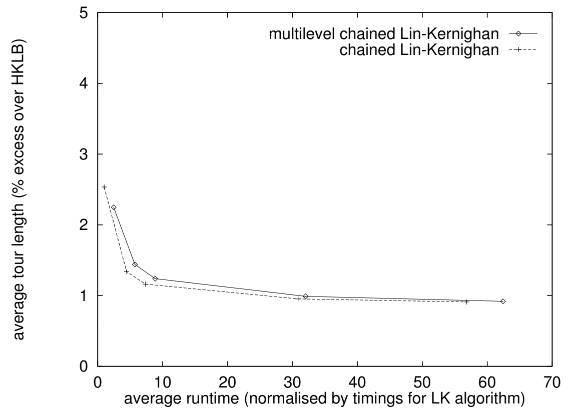
[4]see `http://www.research.att.com/~dsj/chtsp/`
[5]available from `http://www.iwr.uni-heidelberg.de/iwr/comopt/software/TSPLIB95/`
[6]available from `http://www.keck.caam.rice.edu/concorde/download.html`



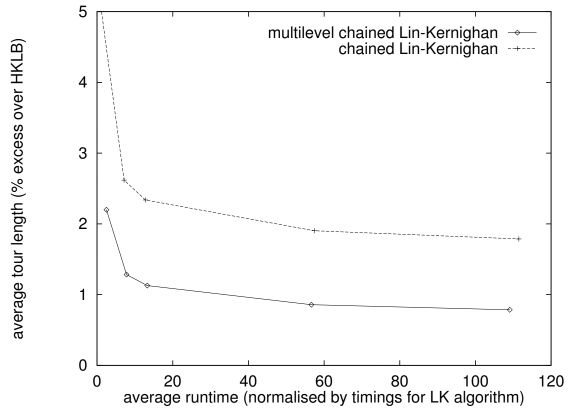
(a) complete test suite



(b) `TSPLIB` instances



(c) random uniform instances



(d) random clustered instances

Figure 6: Plots of convergence behaviour for the travelling salesman test suite

$0, N/20, N/10, N/2, N$ 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 in §5.3). Figure 6(a) illustrates the results 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 §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 dramatic 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.

The results in Figure 6(a) are based on averaged results over all 79 test instances and in [58] reveal some interesting consistencies. However, looking in more detail at individual instances in [58] the results are a little less clear. Firstly comparing C$^{\lambda}$LK and MLC$^{\lambda/2}$LK for the uniformly distributed random instances, class (II), it is clear that the multilevel process does not really contribute to the quality. On the other hand, for the clustered random examples, class (III), those which the CLK algorithm finds more difficult to optimise, the multilevel strategy significantly enhances the tour quality. This is even more striking for some of the real life instances from `TSPLIB`. CLK variants all have great difficulty with these examples (this is also remarked on in [1, 26] and indeed Neto has suggested improvements in the CLK algorithm to make it *cluster-aware*, [45]) and yet the multilevel variants find very good solutions. 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 a cluster is typically only going to 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.

To explore this dependency on different classes of problem instance a little 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). The averaged results are illustrated graphically in Figures 6(b)-(d) and as can be seen 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, although the asymptotic convergence looks to be about the same. Finally for the randomly clustered examples (Figure 6(d)) the ability of the multilevel framework to aid the convergence is at its most dramatic.

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.

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. [7, 13, 35, 36]. However it is also often cited as one of the most difficult combinatorial optimisation problems, e.g. [27]. The chromatic number of G, denoted $\chi(G)$, is the minimum number of colours required to colour the graph. Not only is the problem of finding $\chi(G)$ NP-hard, [17], but Lund & Yannakakis have even shown that, for some $\epsilon > 0$, approximate graph colouring within a factor of N^ϵ is also NP-hard, [39].

4.1 A multilevel algorithm for the graph colouring problem

Recently a multilevel algorithm has been introduced for the GCP, [60]. 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 & de Werra, [23], and Glover *et al.*, [20], have applied tabu search, Johnson *et al.* have looked at simulated annealing, [25], and Culberson *et al.* have investigated the iterated greedy algorithm, [10, 11, 12]. In [60] multilevel versions of two such approaches, tabu search and the iterated greedy algorithm, are investigated; here we summarise and discuss the tabu search results (and briefly mention those of iterated greedy).

4.1.1 Multilevel elements

Since the problem is graph-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 7 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.

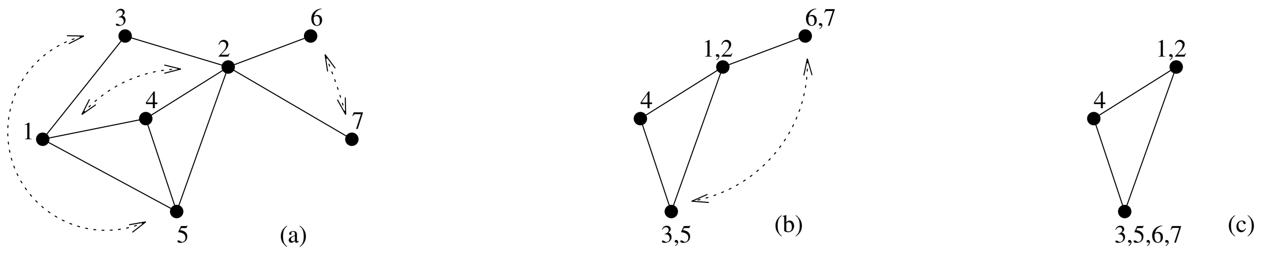


Figure 7: An example of graph contraction for the colouring problem

Vertex matching. The matching procedure is also based on algorithms used in graph partitioning, e.g. [22, 62], 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 rather than any non-adjacent vertex. Thus in Figure 7(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 §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 [60] is essentially the same as for many of the graph partitioning implementations, e.g. [22, 29, 62]. 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 & 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 & 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, [60].

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 [60] 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 7(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 [60]. Disconnected graphs (see §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, [60].

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, $G(V, E)$, every vertex must have a different colour and so $\chi(G) = |V|$.

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

In this section we outline the tabu search algorithm and the way that it can be used for the refinement phase of the multilevel procedure. This is a summary of the more detailed description in [60] in which the iterated greedy algorithm is also investigated. These two algorithms were selected because the source code, written by Culberson, is available[7] and tabu search has been chosen here primarily because in comparison with iterated greedy, the results are a little more demonstrative.

Tabu search is a general technique, proposed by Glover, [19], for finding approximate solutions to combinatorial optimisation problems. Given an existing solution, the search moves stepwise through the solution space and at each iteration steps to the neighbouring solution with the lowest cost (even if that cost is higher than the current one). However to prevent cyclic behaviour, i.e. stepping straight back to the solutions that the algorithm has just left and hence becoming stuck in local minima, a 'tabu' list is maintained containing disallowed moves to states that the algorithm has recently visited. Generally moves only remain tabu during a certain number of iterations and so the tabu list is normally implemented as a fixed length queue where the oldest move is dropped every time a new move is added.

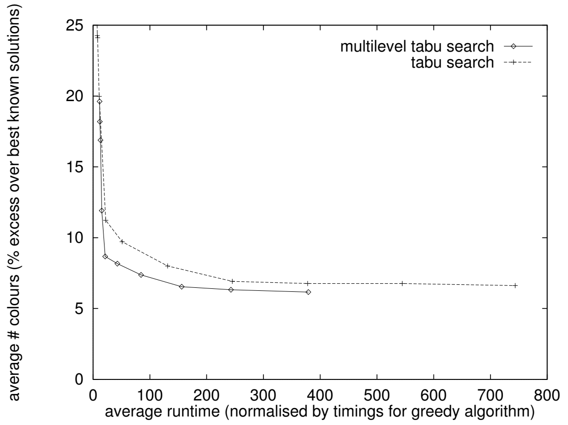
Hertz & de Werra proposed a tabu search algorithm for the graph colouring problem in [23]. Strictly speaking this implementation does not move through the solution space but instead moves through a closely related space to try and find a legitimate colouring. Thus given an existing k-colouring of the graph and a target colour, $k_t < k$, the vertices of the graph are placed in k_t colour classes (with inevitable colouring conflicts). This is a point of the search space and neighbours of this point can be generated by picking any vertex in conflict and moving it to a different colour class. Hertz & de Werra's algorithm works by generating a 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, [12]) after λ iterations. In [60] 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.

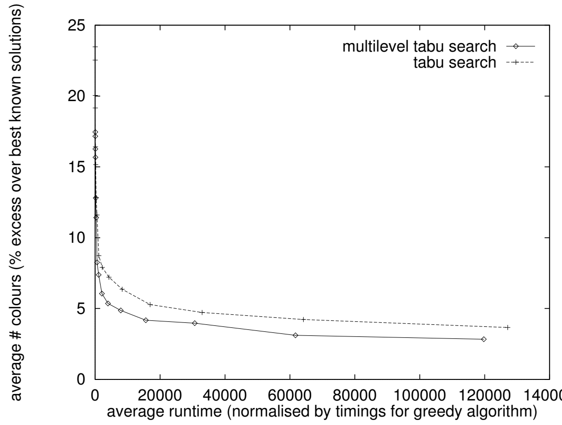
4.2 Experimental results

In [60] the above procedures are tested on a large suite of 90 instances which consists of the examples compiled for the 2nd DIMACS implementation challenge, [27], augmented by further examples added since then[8]. As well as summarising the results from [60] here we have extended them by including an additional set of experiments on the sparse test suite used for the GPP in §2.2 (mirroring the testing of the partitioning algorithms on the colouring suite). Again the test methodology in §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 and in particular the papers [12, 16, 20, 28, 36, 43, 51]. 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 $\lambda_0 = \lambda$.

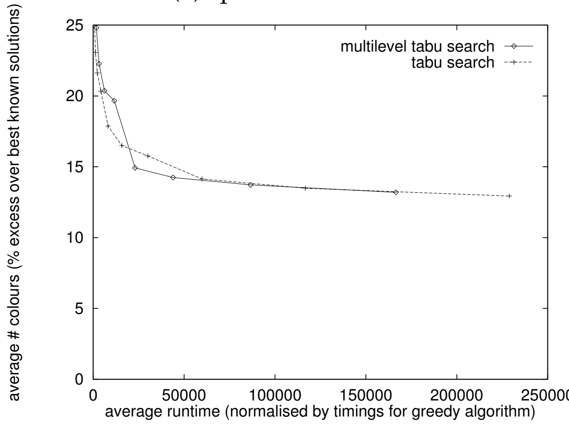
[7]via `http://www.cs.ualberta.ca/~joe/Coloring/Colorsrc/index.html`
[8]available from `http://mat.gsia.cmu.edu/COLOR/instances.html`



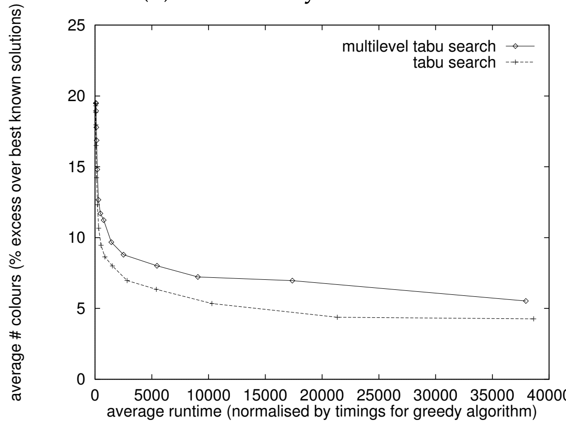
(a) sparse instances



(b) low-density instances



(c) medium-density instances



(d) high-density instances

Figure 8: Plots of convergence behaviour for the colouring test suites

4.2.1 Sparse test suite

Figure 8(a) shows the colouring results for the sparse test suite. Here we test TS$^\lambda$ (with $\lambda = 2^m$ and $m = 0, \ldots, 10$) against MLTS$^\lambda$ (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 §1.3 & §5.3) does not hold true and 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, as mentioned in §2.2, 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.

4.2.2 Colouring test suite

As in [60] and in §2.2.2 above, the colouring test suite is split into 3 subclasses based on density Δ: low ($0\% \leq \Delta \leq 33\frac{1}{3}\%$) with 58 out of 90 instances, medium ($33\frac{1}{3}\% < \Delta \leq 66\frac{2}{3}\%$) with 23 instances and high ($66\frac{2}{3} < \Delta \leq 100\%$) with just 9 instances. Figures 8(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 8(c), and both variants seem to be reaching approximately the same asymptotic convergence. However, for the high-density examples in Figure 8(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 8(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 further in §6.1.

Note that in [60] similar comparisons were made for the iterated greedy algorithm (IG) and a multilevel version (MLIG). For the low-density instances, MLIG had faster convergence than IG although both appeared to reach the same asymptotic convergence which was poorer than both TS and MLTS. The results for the medium-density instances were inconclusive and the asymptotic convergence for both IG & MLIG was more or less the same as for TS & MLTS. However for the high-density subclass IG outperformed both TS & MLTS and MLIG was marginally better still. Finally in all cases IG & MLIG converged much faster than TS & MLTS (at least in terms of runtime).

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 & 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 & 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 & 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 sampling 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 partitioning on the basis that the coarsening successively ***approximates*** the problem. In fact it is somewhat better than this; the coarsening ***samples*** the solution space by placing restrictions on which states the refinement algorithm can visit. Furthermore, this methodology is not confined to multilevel partitioning but can be applied to other combinatorial optimisation problems.

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

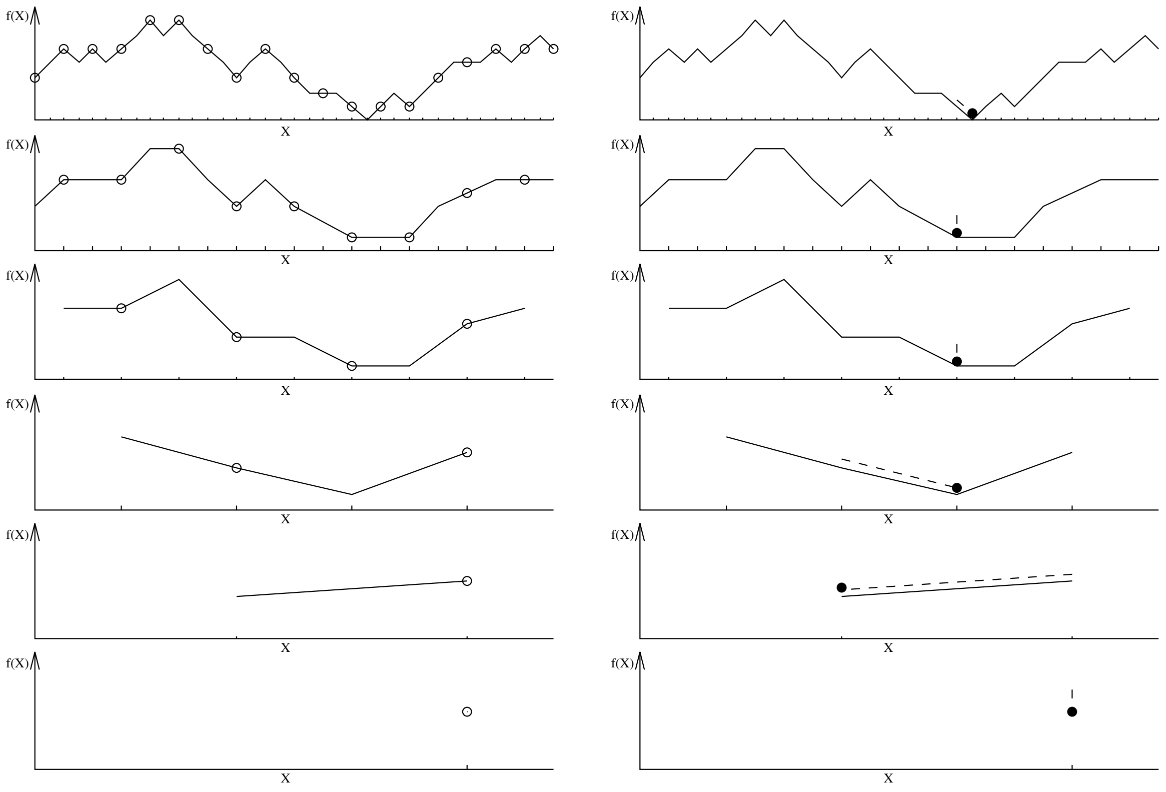


Figure 9: The multilevel scheme in terms of a simple objective function

Figure 9 shows an example of how this might work for a search space $\mathcal{X}$ and objective function $f(\mathcal{X})$ which we aim to minimise. On the left hand side the objective function is gradually sampled and smoothed (the sampled points are circled and all intermediate values removed to give the next coarsest representation). The initial solution for the final coarsened space (shown as a black dot in the bottom right hand figure) is then trivial (because there is only one possible state) although the resulting configuration is not an optimal solution to the overall problem. However this state is used as an initial configuration for the next level up and a *steepest descent* refinement policy finds the nearest local minimum (steepest descent refinement will only move to a neighbouring configuration if the value of the objective function is lower there). Recursing this process keeps the best found solution (indicated by the black dot) in the same region of the solution space. Finally this gives a good initial configuration for the original problem and (in this case) the optimal solution can be found. Note that it is possible to pick a different set of sampling points for this example for which the steepest descent policy will fail to find the global minimum, but this only indicates, as might be expected, that the multilevel procedure is somewhat sensitive to the coarsening strategy.

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

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

Figure 10: A schematic of the multilevel refinement algorithm

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 meta-heuristic. Figure 10 contains a schematic of this process in pseudo-code. Here P_l refers to the coarsened problem after l coarsening steps, $C_l\{P_l\}$ is a solution of this problem and $C_l^0\{P_l\}$ denotes the initial solution.

5.2 Algorithmic requirements

Assuming that the above analysis does apply, how can a multilevel strategy be implemented for a given combinatorial optimisation problem? Here we discuss the algorithmic requirements illustrated by examples from the graph partitioning, graph colouring and travelling salesman problems (the GPP, GCP & 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 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 complex techniques and there is no reason (other than execution time) why it should not be a more sophisticated scheme. Indeed, examples of multilevel partitioning schemes exist for simulated annealing, [57], tabu search, [4, 57], and even genetic algorithms, [33].

The refinement algorithm must also be able to cope with any additional restrictions placed on it by using a coarsened problem (e.g. for the GPP the coarser 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 two principles seem to hold:

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

This still does not tell us ***how*** to coarsen a given problem and the examples suggest that it is very much problem dependent. Furthermore it has been shown (for partitioning at least), that it is usually more profitable for the coarsening to respect the objective function in some sense (see e.g. the heavy edge matching strategy in [29] and the template cost matching in [64]). In this respect it seems likely that the most difficult aspect of finding an effective multilevel algorithm for a given problem and given refinement scheme is the (problem dependent) task of devising the coarsening strategy.

5.3 Typical runtime

As we have seen, common to all 3 examples presented here is an approximate factor of two runtime between a local search algorithm at intensity λ, LS$^\lambda$, and the multilevel version, MLLS$^\lambda$. In other words, if $T(\mathrm{A})$ denotes the runtime of algorithm A, then $T(\mathrm{MLLS}^\lambda) \approx 2T(\mathrm{LS}^\lambda)$. In fact it is not too difficult to give a justification why the multilevel strategy should add this factor of two. Suppose first of all that the LS algorithm is $O(N)$ in execution time (and this assumption is generally true for the local search algorithms used here which, if not actually $O(N)$, fall somewhere between sublinear and subquadratic). 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 less than this (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 $T_L = T(\mathrm{LS}^\lambda)$ be the time for LS$^\lambda$ 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$^\lambda$ having linear runtime gives the total runtime for MLLS$^\lambda$ as $T_C + T_L/N + \ldots + T_L/2 + T_L$. If λ is large enough then typically $T_C \ll T_L$ and so we can neglect it giving a total runtime of $T_L/N + \ldots + T_L/2 + T_L = 2T_L$, i.e. MLLS$^\lambda$ takes twice as long as LS$^\lambda$ to run.

If λ is small then T_C can take a much larger 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 initialisation (typically both $O(|V| + |E|)$). Of course the assumptions made render this only very approximate but nonetheless this factor of two is a good 'rule of thumb' (and in fact the only examples that appear to significantly break this rule are Figures 2(b) & 8(a)). Finally note that if the multilevel procedure were to be combined with an $O(N^2)$ or even $O(N^3)$ refinement algorithm then this analysis comes out even better for the multilevel overhead as the final refinement step would require an even larger proportion of the total.

5.4 Solution-based coarsening and iterated multilevel algorithms

We have seen (in §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 v 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.

Thus, provided the refinement algorithm guarantees not to find a worse solution than the initial one (in fact even if it does find a worse solution it can be replaced by the initial one) the process 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/refinement loops. At each iteration the best solution found previously is used to create a solution-based coarsening and a new hierarchy of graphs is constructed. If the matching is carried out randomly then each iteration is very likely to give a different hierarchy of graphs to previous iterations and hence give the possibility for the refinement algorithm to visit different solutions in the search space. 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 (§2.2.3) indicate that it can sometimes be very helpful and further investigation is worthwhile.

6 Conclusions

6.1 Summary of results

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 8(d) for the GCP. Furthermore the ability of multilevel refinement to aid local search varies from problem class to problem class (e.g. it is much easier to obtain improvements for the GPP and TSP than it is for the GCP).

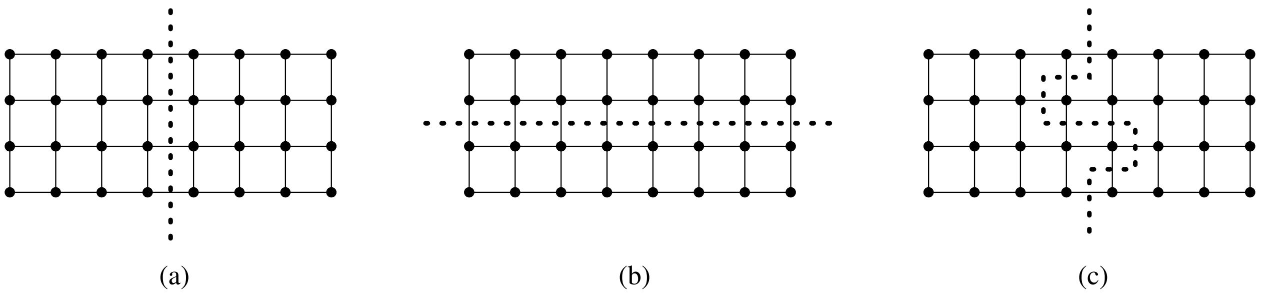


Figure 11: Example bisection partitions showing different global and local qualities

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 11 for the GPP. The optimal bisection partition is shown in Figure 11(a) with a cut-weight of 4. Two other partitions are shown in Figures 11(b) & 11(c); however, whilst both have a cut-weight of 8, clearly Figure 11(b) is globally poor but locally optimal, whilst Figure 11(c) is close to the global solution but locally poor. We believe that this dependence of the cost function on local conditions, if present in a given problem class, may allow the multilevel framework 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 a changes in the solution will eventually result in a lower cost. Indeed this is the reason why methods such as Culberson's iterated greedy algorithm look for improvements in the colouring sum rather than just the cost function (see e.g. [12]). This may mean that coarsened versions of the problem are not as much help 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). Hence unfavourable matching takes place and perhaps pulls the solution into a suboptimal basin of attraction on the surface of the objective function. 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 the 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 even be extended to account for the behaviour of the multilevel TSP algorithm on random instances, Figure 6(c) since matching is not very clear for a uniform distribution of vertices and we can even define a density indicator. Thus given a Euclidean TSP instance of size N, if we superimpose a grid of spacing $1/\sqrt{N}$, then we can count the number of occupied cells, N_o, and define the density Δ as the number of occupied cells 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.

To summarise then, it seems from the results presented, 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 & 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 & 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 3(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. [36]. 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', [35]. Indeed the original success of multilevel partitioning was built around the requirement to partition sparse, mesh-based problems. Furthermore, the results on appropriate problem classes can sometimes be spectacular, e.g. Figures 2(a) & 6(d). This augments existing evidence that, although the multilevel framework cannot be considered as a panacea for combinatorial optimisation problems, it can provide a useful, sometimes very useful, addition to the combinatorial optimisation toolkit.

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 some existing papers on the TSP and other combinatorial optimisation problems do) 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 so that theoretical benefits are tested for, and hence can be realised in, practical applications.

6.2 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.

6.2.1 Sparsification

A slightly disappointing feature of the multilevel partitioning & colouring algorithms, at least in the manifestations described here, was the 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 would result in a sparsification of the problem would depend on the relative number of vertices and edges removed but it seems quite promising as a technique. It also fits in with the hybrid scheme successfully used by Hertz & DeWerra, [23], and by Fleurent & Ferland, [16], 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.

6.2.2 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. There is also plenty of evidence, for partitioning at least, that the matching procedure, and the cost function, if any, which it aims to optimise, can strongly affect the final solution quality, e.g. [29, 64], although there is no direct relationship between the two. Indeed it is not always clear what the matching should be aiming to optimise (apart from maximising the number of matches) and for instance the heavy edge matching scheme, [29], commonly used for partitioning can only operate on weighted graphs and hence does not apply 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 and can represent some chosen function for multilevel applications (e.g. maximising the weight of edges collapsed for the GPP). Polynomial algorithms which can compute optimal matchings are available, e.g. [8], 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 *et al.* for graph partitioning, [42]. Currently, in all 3 example matching schemes above, a vertex v_0 picks a preferred candidate v_1 and they are matched. However, v_1 may have available matches which are much more advantageous than v_0. Hence, in the scheme of Monien *et al.*, rather than a match being made at this point, v_1 then picks its preferred match and the process is repeated until a pair of vertices is generated that mutually select each other as a match. Tie-breaking of matches by random selection is forced to be commutative to prevent infinite loops.

6.2.3 Vertical refinement

For the multilevel process described above 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, [5]. 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 & 4) will be problematic. However, since the coarsening can never give a truly representative sampling of the objective function, each step is much more precise way of moving around the solution space than the methods described above.

References

[1] D. Applegate, R. Bixby, V. Chvátal, and W. J. Cook. Finding Tours in the TSP. Tech. Rep. TR99-05, Dept. Comput. Appl. Math., Rice Univ., Houston, TX 77005, 1999.

[2] D. Applegate, W. J. Cook, and A. Rohe. Chained Lin-Kernighan for large traveling salesman problems. Tech. Rep., Dept. Comput. Appl. Math., Rice Univ., Houston, TX 77005, July 2000.

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

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

[5] A. Brandt. Multilevel computations: Review and recent developments. In S. F. McCormick, editor, *Multigrid Methods: Theory, Applications, and Supercomputing (Proc. 3rd Copper Mountain Conf. Multigrid Methods)*, volume 110 of *LNPAM*, pages 35–62. Marcel Dekker, New York, 1988.

[6] T. N. Bui and C. Jones. A Heuristic for Reducing Fill-In in Sparse Matrix Factorization. In R. F. Sincovec *et al.*, editor, *Parallel Processing for Scientific Computing*, pages 445–452. SIAM, Philadelphia, 1993.

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

[8] W. J. Cook and A. Rohe. Computing minimum-weight perfect matchings. *INFORMS J. Comput.*, 11(2):138–148, 1999.

[9] G. A. Croes. A method for solving traveling salesman problems. *Oper. Res.*, 6:791–812, 1958.

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

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

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

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

[14] C. Farhat. A Simple and Efficient Automatic FEM Domain Decomposer. *Comput. & Structures*, 28(5):579–602, 1988.

[15] C. M. Fiduccia and R. M. Mattheyses. A Linear Time Heuristic for Improving Network Partitions. In *Proc. 19th IEEE Design Automation Conf.*, pages 175–181. IEEE, Piscataway, NJ, 1982.

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

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

[18] M. R. Garey, D. S. Johnson, and L. Stockmeyer. Some simplified NP-complete graph problems. *Theoret. Comput. Sci.*, 1:237–267, 1976.

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

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

[21] M. Held and R. M. Karp. The Traveling Salesman Problem and Minimum Spanning Trees. *Oper. Res.*, 18:1138–1162, 1970.

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

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

[24] D. S. Johnson. Local Optimization and the Traveling Salesman Problem. In M. S. Paterson, editor, *Proc. 17th Colloq. on Automata, Languages and Programming*, volume 443 of *LNCS*, pages 446–461. Springer, Berlin, 1990.

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

[26] D. S. Johnson and L. A. McGeoch. The travelling salesman problem: a case study. In E. Aarts and J. K. Lenstra, editors, *Local Search in Combinatorial Optimization*, pages 215–310. John Wiley & Sons, Chichester, 1997.

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

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

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

[30] G. Karypis and V. Kumar. Multilevel Algorithms for Multi-Constraint Graph Partitioning. In D. Duke, editor, *Proc. Supercomputing '98, Orlando*. ACM SIGARCH and IEEE Comp. Soc., 1998. (CD-ROM).

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

[32] G. Karypis and V. Kumar. Multilevel k-way hypergraph partitioning. In *Proc. 36th Design Automation Conf.*, pages 343–348, 1999.

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

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

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

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

[37] S. Lin. Computer solutions of the traveling salesman problem. *Bell Syst. Tech. J.*, 44:2245–2269, 1965.

[38] S. Lin and B. W. Kernighan. An effective heuristic for the traveling salesman problem. *Oper. Res.*, 21(2):498–516, 1973.

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

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

[41] O. C. Martin, S. W. Otto, and E. W. Felten. Large-step Markov chains for the TSP incorporating local search heuristics. *Oper. Res. Lett.*, 11(4):219–224, 1992.

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

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

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

[45] D. M. Neto. *Efficient Cluster Compensation for Lin-Kernighan Heuristics*. PhD thesis, Dept. Comp. Sci., Univ. Toronto, Canada, 1999.

[46] F. Pellegrini and J. Roman. SCOTCH : A Software Package for Static Mapping by Dual Recursive Bipartitioning of Process and Architecture Graphs. In H. Liddell *et al.*, editor, *High-Performance Computing & Networking, Proc. HPCN'96, Brussels*, volume 1067 of *LNCS*, pages 493–498. Springer, Berlin, 1996.

[47] G. Reinelt. TSPLIB— A Traveling Salesman Problem Library. *ORSA J. Comput.*, 3(4):376–384, 1991.

[48] G. Reinelt. TSPLIB95. Tech. Rep., Inst. Angewandte Math., Univ. Heidelberg, 1995.

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

[50] K. Schloegel, G. Karypis, and V. Kumar. A New Algorithm for Multi-objective Graph Partitioning. In P. Amestoy *et al.*, editor, *Euro-Par'99 Parallel Processing*, volume 1685 of *LNCS*, pages 322–331. Springer Verlag, Heidelberg, 1999.

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

[52] H. D. Simon. Partitioning of Unstructured Problems for Parallel Processing. *Computing Systems Engrg.*, 2:135–148, 1991.

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

[54] A. J. Soper, C. Walshaw, and M. Cross. A Combined Evolutionary Search and Multilevel Optimisation Approach to Graph Partitioning. Tech. Rep. 00/IM/58, Comp. Math. Sci., Univ. Greenwich, London SE10 9LS, UK, April 2000.

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

[56] M. Toulouse, K. Thulasiraman, and F. Glover. Multi-level Cooperative Search: A New Paradigm for Combinatorial Optimization and an Application to Graph Partitioning. In P. Amestoy *et al.*, editor, *Proc. Euro-Par '99 Parallel Processing*, volume 1685 of *LNCS*, pages 533–542. Springer, Berlin, 1999.

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

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

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

[60] C. Walshaw. A Multilevel Approach to the Graph Colouring Problem. Tech. Rep. 01/IM/69, Comp. Math. Sci., Univ. Greenwich, London SE10 9LS, UK, May 2001.

[61] C. Walshaw and M. Cross. Load-balancing for parallel adaptive unstructured meshes. In M. Cross *et al.*, editor, *Proc. Numerical Grid Generation in Computational Field Simulations*, pages 781–790. ISGG, Mississippi, 1998.

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

[63] C. Walshaw and M. Cross. Multilevel Mesh Partitioning for Heterogeneous Communication Networks. *Future Generation Comp. Syst.*, 17(5):601–623, 2001. (originally published as Univ. Greenwich Tech. Rep. 00/IM/57).

[64] C. Walshaw, M. Cross, R. Diekmann, and F. Schlimbach. Multilevel Mesh Partitioning for Optimising Domain Shape. *Intl. J. High Performance Comput. Appl.*, 13(4):334–353, 1999. (originally published as Univ. Greenwich Tech. Rep. 98/IM/38).

[65] C. Walshaw, M. Cross, and K. McManus. Multiphase Mesh Partitioning. *Appl. Math. Modelling*, 25(2):123–140, 2000. (originally published as Univ. Greenwich Tech. Rep. 99/IM/51).