Multilevel Landscapes in Combinatorial Optimisation

Chris Walshaw and Martin G. Everett

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 : 02/IM/93

April 30, 2002

Abstract

We consider the multilevel paradigm and its potential to aid the solution of combinatorial optimisation problems. The multilevel paradigm is a simple one, which involves recursive coarsening to create a hierarchy of approximations to the original problem. An initial solution is found and then iteratively refined at each level, coarsest to finest. Although the multilevel paradigm has been in use for many years and has been applied to many problem areas (most notably in the form of multigrid methods), it has only recently been suggested as a metaheuristic for combinatorial optimisation problems. It has been proposed that, for such problems, multilevel coarsening is equivalent to recursively filtering the solution space to create a hierarchy of increasingly coarser and smaller spaces. It is also suggested that perhaps this aids the local search algorithms used to refine the solution on each level by somehow 'smoothing' the landscape of the solution spaces. In this paper, with some example problem instances drawn from graph partitioning and the travelling salesman problem, we take a detailed look at how the coarsening affects the hierarchy of solution landscapes. In particular we are interested in how the coarsening and hence filtering of the original space impacts on the maximum, minimum and average values of the cost function in the coarsened spaces. However, we also explore the manner in which the density of problem instances can moderate the effectiveness of a multilevel refinement algorithm.

Keywords: Multilevel Refinement; Combinatorial Optimisation; Metaheuristic; Graph Partitioning; Travelling Salesman.

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 and then iteratively refined at each level, coarsest to finest. Projection operators can transfer the solution from one level to another.

As a general solution strategy, the multilevel paradigm has been in use for many years and has been applied to many problem areas (for example multigrid techniques can be viewed as a prime example of the multilevel paradigm). Overview papers such as [5, 29] attest to its efficacy. However, with the exception of the graph partitioning problem, multilevel techniques have only recently been applied to combinatorial optimisation problems and in this paper we investigate further the inner mechanics of multilevel combinatorial algorithms.

In this context we are interested in the broad class of discrete systems, with a finite but usually exponential number of states (collectively known as the *solution space*), 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, [10], (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. Typically this is achieved by iterative local search schemes which can step from one solution to other 'local' solutions. In fact it is the search scheme that defines which solutions are local to each other; in other words, each solution has a *neighbourhood* of solutions, those that the scheme can reach in a single step (and note that without a local search scheme the solution space is unordered). We can then imagine the search tracing pathways through the solution space moving downwards to lower cost solutions (and often becoming trapped in local minima of the cost function) or upwards to higher cost. In this way the local search scheme induces a *landscape* on the solution space, so-named because, conceptually at least, it consists of peaks, valleys and sometimes plateaus.

In [34] the use of multilevel techniques in combinatorial optimisation is discussed with the aid of three example problems, graph partitioning, graph-colouring and the travelling salesman problem. In that paper it is shown that, under certain conditions, multilevel coarsening is equivalent to recursively filtering the solution space to create a hierarchy of increasingly coarser and smaller spaces[1]. It is also suggested that perhaps this aids the local search algorithms used to refine the solution on each level by somehow 'smoothing' the landscape of the solution spaces. In this paper, with some example problem instances drawn from graph partitioning and the travelling salesman problem, we take a detailed look at how the coarsening affects the hierarchy of solution landscapes. In particular we are interested in how the coarsening and hence filtering of the original space impacts on the maximum, minimum and average values of the cost function in the coarsened spaces. However, we also explore the manner in which the density of problem instances can moderate the effectiveness of a multilevel refinement algorithm.

1.1 Overview

The rest of the paper is organised as follows. Firstly we outline the elements of the multilevel paradigm as applied to combinatorial problems and discuss some related ideas. Then, in Sections 2 & 3 we outline two existing multilevel implementations and look at the typical landscapes produced in each case. Thus in Section 2 we discuss 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. Here we examine why that might be the case. In Section 3 we then outline 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 and we investigate why that might be so. Finally in Section 4 we draw some conclusions and present some ideas for future work.

1.2 Review of multilevel combinatorial optimisation

The primary aim of this paper is to give some insight into the inner mechanics of the multilevel method by considering characteristics of the hierarchy of solution spaces produced by the coarsening. As mentioned above, the coarsening constructs a series of approximations to the original problem; 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.

[1]In [34] this process is referred to as *sampling* the solution space; here we refer to it as *filtering*, partly because we wish to use the term sampling to describe another process in this paper, and more importantly, it is a more suggestive word for what actually occurs.

On the other hand, viewing the multilevel process from the point of view of the objective function and, in particular the hierarchy of solution spaces, is considerably more enlightening. Typically the coarsening is carried out, as described in greater detail below (§2.1.1 & §3.1.1), by matching groups (usually pairs) of solution variables together and representing each group with a single variable in the coarsened space. As discussed in [34] and illustrated below, if carried out correctly the effect is to ***filter*** the solution space by placing restrictions on which solutions the refinement algorithm can visit. That paper then goes on to hypothesise that, if the coarsening manages to filter the solution space so as to gradually ***smooth*** the objective function, the multilevel representation of the problem combined with an iterative refinement algorithm should work well as an optimisation metaheuristic. In other words, by filtering a large amount of irrelevant detail from the solution space (in particular the higher cost solutions which are not close to local optima), the multilevel component allows the refinement algorithm to find regions of the solution space where the objective function has a low average value (e.g. broad valleys). This does rely on a certain amount of 'continuity' in the objective function but it is not unusual for these sort of problems that changing one or two components of the solution tends not to change the cost very much. On a more pragmatic level this same process also allows the refinement to take larger steps around the solution space (e.g. for graph partitioning, rather than swapping single vertices, the local search algorithm can swap whole sets of vertices as represented by a single coarsened vertex).

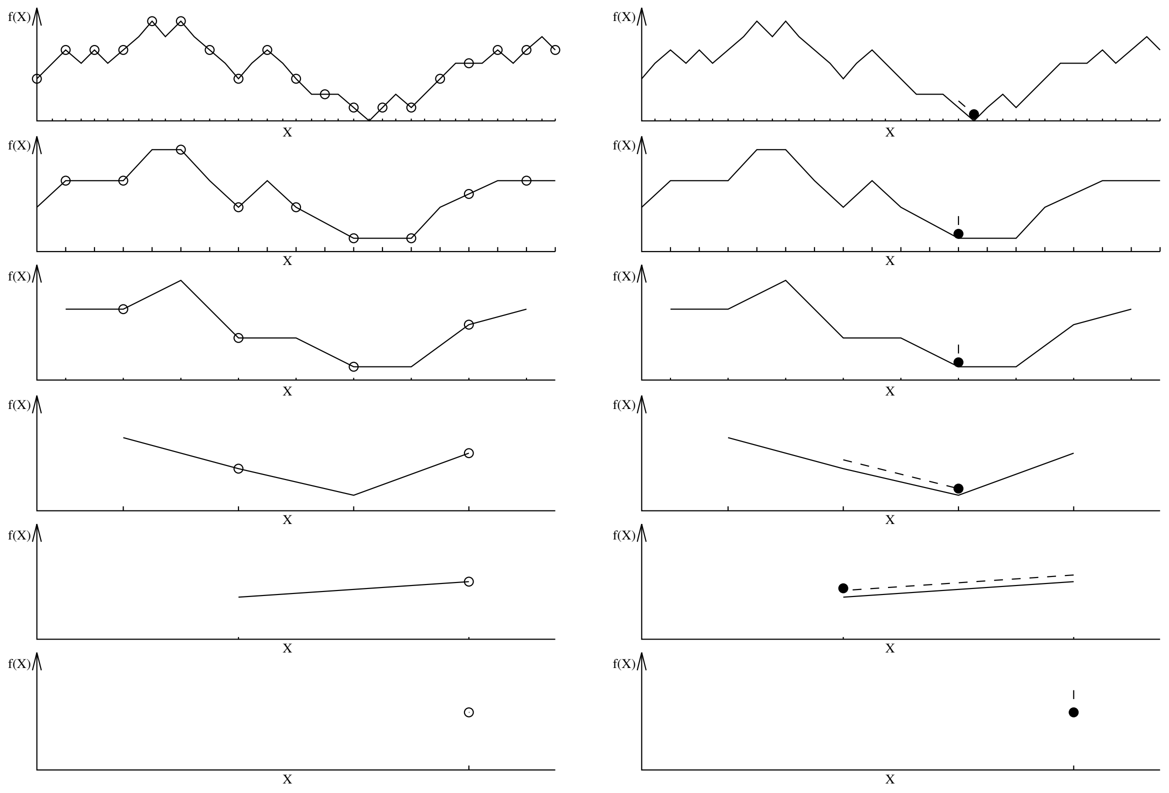


Figure 1: The multilevel scheme in terms of a simple objective function

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

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

In [34] experimental evidence is also presented which seems to indicate that higher density problem instances may be less susceptible to multilevel techniques than sparse or low-density examples. It is suggested that this is because the density somehow confuses the coarsening algorithm and it is then unclear which vertices to match together (because there are so many possible candidates). As a result unfavourable matching may take place (i.e. between vertices which tend not to be matched in high quality solutions) and the filtration may remove the better solutions from the coarsened spaces. In this sense it is perhaps not strictly the density which determines whether or not a problem instance is likely to be susceptible to multilevel refinement, but whether the instance has inherent *matching affinities*. In other words, if it is easy to pick matches with good probability that the match will appear as part of a high quality solution then it is likely that multilevel refinement will be successful. The density is then a good indicator of inherent matching affinities with sparse and low-density instances providing the clearest matching choices. Here we aim to explore this linkage with density in more detail.

The objectives of this paper are thus twofold:

(I) To examine the characteristics of the hierarchy of landscapes produced by multilevel coarsening.

(II) To investigate how the density of the problem instances can affect these landscapes.

1.3 Related work

Because multilevel algorithms are well-known in many other areas of mathematics there is a large body of literature which could be said to be related to the ideas presented here. For interested readers a good start are the overview papers [5, 29]. However a few ideas have been proposed recently which are related to those presented here but which are not specifically multilevel methods. Firstly Glover *et al.* discuss the idea of vocabulary building in [11] in particular to assemble tour fragments for the TSP. These are grown and combined (possibly with exact algorithms) to produce a population of complete solutions. The scheme then uses destructive processes to break the solutions into fragments again, which are then recombined, and the whole construction/destruction cycle is repeated iteratively. This could be viewed as a form of iterated multilevel population based scheme (see [34]) although without the use of refinement at each level. However the idea is only presented as a possible solution technique and no results are given.

A second idea perhaps more closely related to the principles behind multilevel refinement is the search space smoothing scheme of Gu & Huang, [12]. This uses recursive smoothing (analogous to recursive coarsening) to produce versions of the original problem which are simpler to solve. Thus in the example application Gu & Huang apply their technique to the TSP by forcing the inter-city edges to become increasingly uniform in length at each smoothing phase (if all edges between all cities are the same length then every tour is optimal). The obvious drawback is that each smoothing phase distorts the problem further (so that a good solution to a smoothed problem may not be a good solution to the original). In addition, the smoothed spaces are the same size as the original problem, even if the solution is potentially easier to refine, and hence may be equally as expensive to optimise. By contrast, multilevel coarsening filters rather than smoothing directly (although with the obvious drawback that the best solutions may be removed from the coarsened spaces) and so the coarsened spaces are smaller and hence can be refined more rapidly. It is also unclear whether search space smoothing is as general as coarsening and hence whether it could be applied to problems other than the TSP.

2 The Graph Partitioning Problem

The k-way graph partitioning problem (GPP) can be stated as follows: given an undirected graph $G(V, E)$ of vertices V and edges E (where both vertices and edges may be weighted), 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 well known that this problem is NP-complete, [10], so in recent years much attention has been focused on developing suitable heuristics.

Note that henceforth 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).

2.1 Multilevel graph partitioning

The GPP was the first combinatorial optimisation problem to which the multilevel paradigm was applied and there is now a considerable volume of literature about multilevel partitioning algorithms. Initially used as an effective way of speeding up partitioning schemes, it was soon recognised as, more importantly, giving them a more 'global' perspective, [18], and has been successfully developed as a strategy for overcoming the localised nature of the Kernighan-Lin (KL), [21], and other optimisation algorithms. Typically such multilevel implementations match and coalesce pairs of adjacent vertices 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. At each change of levels, the final partition of the coarser graph is used to give the initial partition for the next level down. The use of multilevel combinatorial refinement for partitioning was first proposed by both Hendrickson & Leland, [13] and Bui & Jones, [6], and was inspired by Barnard & Simon, [3], who used a multilevel numerical algorithm to speed up spectral partitioning.

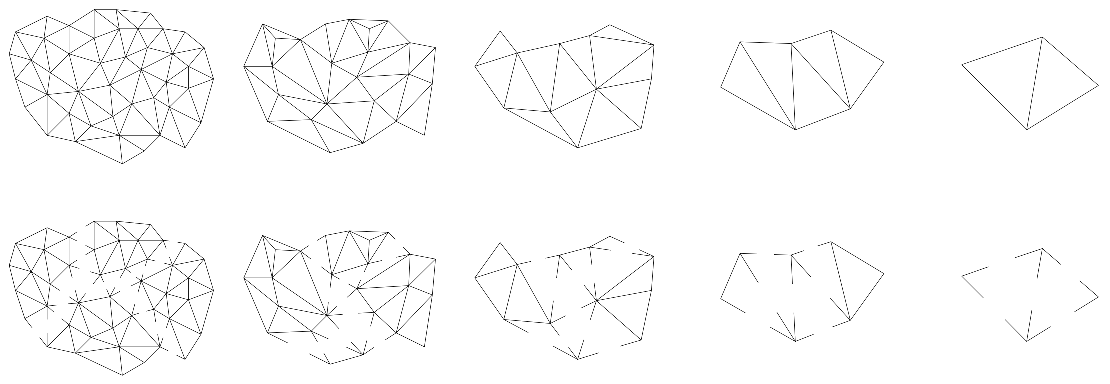


Figure 2: An example of multilevel partitioning

Figure 2 shows an example of a multilevel partitioning scheme in action. On the top row (left to right) the graph is coarsened down to 4 vertices which are (trivially) partitioned into 4 sets (bottom right). The solution is then successively extended and refined (right to left; each graph shows the final partition for that level). Although at each level the refinement is only local in nature, a high quality partition is still achieved.

2.1.1 Coarsening

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

A simple way to construct a maximal independent subset of edges is to create a randomly ordered list of the vertices and visit them in turn, matching each unmatched vertex with an unmatched neighbour (or with itself if no unmatched neighbours exist). Matched vertices are removed from the list. If there are several unmatched neighbours the choice of which to match with can be random, but it has been shown by Karypis & Kumar, [18], that it can be beneficial to the optimisation to collapse the most heavily weighted edges.

This simple but rapid algorithm (which has $O(|E|)$ complexity) is guaranteed to construct a *maximal matching* (because no more edges can be added without breaking the independence criterion). However it will not necessarily construct a *maximum matching*, a matching with the largest possible size (which is bounded above by $|V|/2$ – i.e. if every vertex is matched with another then there will be $|V|/2$ edges in the set). A maximum matching is much more costly to construct (certainly greater than $O(|E|)$ in complexity) and algorithms for this purpose are the subject of a considerable body of literature, e.g. [7]. In practice the fact that a suboptimal matching is found appears not to matter, although we discuss this topic further in §4.2.

As mentioned above this coarsening has the effect of filtering the solution space. To see this suppose 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 one of the partition subsets, both u & v are also both being assigned to that subset. In this way the matching restricts a refinement algorithm working on 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 specified at the time of coarsening. Since many vertex pairs are generally coalesced from all parts of G_l to form G_{l+1} this set of restrictions is equivalent to filtering the solution space and hence the surface of the objective function.

2.1.2 Refinement

Although for the purposes of this paper we are only interested in the coarsening process and how it affects the hierarchy of solution landscapes, we outline briefly here the rest of the multilevel partitioning scheme. Thus, 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 v_i is assigned to subset S_i.

At each subsequent change of levels the partition is extended from the graph G_l 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. This partition derived from the previous level then gives an initial partition which can (usually) be improved or 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, [19], and is extremely fast.

A more sophisticated class of method is based on the Kernighan-Lin (KL) bisection optimisation algorithm, [21], which includes limited hill-climbing to enable it to escape from local minima. This has been extended to k-way partitioning in different ways by several authors (e.g. [13, 19, 35]) and recent implementations almost universally use the linear time complexity improvements (e.g. bucket sorting of vertices) introduced to partitioning by Fiduccia & Mattheyses, [9]. However, although the KL algorithm is perhaps the most commonly used refinement scheme, in principle any iterative refinement scheme can be used and examples of multilevel implementations exist for simulated annealing, [31], tabu search, [4, 31], genetic algorithms, [20], cooperative search, [30], and even ant colony optimisation, [22].

2.2 Experimental results

Before exploring the hierarchy of landscapes produced by the multilevel coarsening we had to make a number of decisions about what to test and how to test it. Firstly, since we are interested in the typical features of a multilevel landscape and not, for the purposes of this paper at least, in the merits of different matching techniques, we decided to use the heavy edge matching heuristic (see above §2.1.1) of Karypis & Kumar, [18]. We regard this as a fairly standard algorithm since it is used by most of the public domain partitioning packages, e.g. Chaco, [13], Jostle, [35], and Metis, [19].

Next, for the sake of simplicity and because we are severely restricted in the size of graph that we can handle, at least for complete enumeration, we decided to consider only graph bisection (2-way partitioning). Indeed many researchers have approached the partitioning problem in this way because it can 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. [27].

Dealing with the issues of load-imbalance is not so straightforward. For example, it is well known that relaxing the balance constraint slightly can enable partitioning algorithms to find better solutions, [28]. However, for our purposes, even allowing an imbalance of just one vertex doubles the size of the feasible solution space and hence the enumeration runtime. Indeed Walshaw & Cross, [35], show that allowing increasing imbalance with each coarsening can aid the partitioning process. However a similar *multilevel relaxation* of the balance constraint when enumerating the solution spaces allows anomalies to creep into the landscapes and in particular, solutions which are infeasible on one level may be admitted on higher levels. This is not to denigrate the idea of multilevel relaxation, especially for the relatively homogeneous graphs, typically derived from computational meshes, e.g. [35], where it seems to work particularly well; however, it does not aid analysis of the hierarchical solution spaces.

We therefore decided to enforce the balance constraint at every level, although even this simplification can cause problems. In particular, after two or more levels of coarsening it may no longer be the case that ***any*** solutions exist at all. For example, if we have a graph of 6 vertices with weights $\{3, 3, 3, 3, 3, 1\}$ there is no perfectly balanced bisection with a vertex weight sum of 8 per set. (Of course in practice and as mentioned above multilevel partitioners avoid this issue by admitting solutions which are not balanced, even if, as is possible in the case of multilevel relaxation, they enforce perfect balance in the final partition.) However, for the purposes of this paper, perfect balancing can seriously depopulate the coarser levels and also significantly complicate the enumeration process.

As a work around to this problem we decided to modify the coarsening process very slightly by requiring that ***every*** vertex be matched with another. Thus we match pairs of vertices exactly as described in §2.1.1, but at the end of the process for each level, collect together all the unmatched vertices and force arbitrary pairs of them to match despite the fact that none of them are adjacent (if any of them were adjacent then they would have been matched by the original algorithm). This gives the nice property that if we start with a graph of 2^m vertices, each with a weight of 1, the first coarsening produces a graph of 2^{m-1} vertices each of weight 2, the next produces 2^{m-2} vertices of weight 4 and so on until the final graph has 2 vertices of weight 2^{m-1}. For such graphs perfect balance is easily enforced at every level ***and*** the enumeration of the solution space is not complicated by inhomogeneous vertex weights. The down-side of course is that the process is more likely to match vertices which would naturally prefer to be in different partitions (more likely that is than if we only match adjacent vertices). However, since very little forced matching takes place we do not believe that it distorts the landscapes too much. Indeed what distortion there is would tend to render less favourable coarsened spaces and so there is reason to believe that the true picture of a hierarchical landscape produced by a multilevel partitioner (using imbalance and restricting matches to neighbouring vertices) is actually somewhat better than that painted here.

2.2.1 The testing

We evaluate the multilevel landscapes in two ways. Given an example graph, the first (and preferable) method is to enumerate every possible solution at each level. However for a graph of size $|V|$, each 2-way partition π can be specified by a partition vector of length $|V|$ where $\pi_i = \pi(v_i) = p$ or q depending on whether vertex v_i is assigned to set p or q and where there an equal number of p's and q's (assuming for the moment that $|V|$ is even). This means that the number of solutions is ${}^{|V|}C_{\frac{|V|}{2}}$ where ${}^{n}C_{r}$ denotes the number of combinations of r objects from n with no account taken of the order and is given by $n!/r!(n-r)!$. In fact there is no loss of generality in assuming that $\pi(v_1) = p$ and so the total number of solutions is ${}^{(|V|-1)}C_{\frac{|V|}{2}}$. The enumeration testing is then limited to graphs that can practically be evaluated in a reasonable time. For a graph of size $|V| = 32$ the total number of solutions is ${}^{31}C_{16} = 300,540,195$ and using code running on a 1 GHz Pentium processor under Linux we were able to evaluate every solution in approximately 10 to 40 minutes, depending on the edge density of the example, which seems a reasonable time. (N.B. To illustrate the exponential size of these spaces, this means that had we used graphs of size $|V| = 64$, the next power of 2 up, a similar experiment would have approximately taken between $1,000$ and $4,000$ years to run!) The sizes of the coarsened spaces are then ${}^{15}C_8 = 6,435$ for level 1, ${}^{7}C_4 = 35$ for level 2, ${}^{3}C_2 = 3$ for level 3 and ${}^{1}C_1 = 1$ for level 4. Finally to generate every possible combination we use the algorithm given in [14].

Of course graphs with only 32 vertices are hardly representative of realistic applications and so the second method of testing that we tried using larger graphs was random sampling of the solution space. Thus at each level with $|V_l|$ vertices we picked S random solutions where S is the smaller of ${}^{(|V_l|-1)}C_{\frac{|V_l|}{2}}$ and $\frac{1}{|V_0|} \cdot {}^{31}C_{16}$ where $|V_0| = |V|$, the size of the original graph. This completely arbitrary choice means that for the graphs of size $|V| = 1,024$ on which we actually ran the tests, the number of samples for the coarser levels is the same as the size of the solution space (specifically for $l = 6$ then $S = 6,435$; for $l = 7$, $S = 35$; for $l = 8$, $S = 3$; and for $l = 9$, $S = 1$) whilst for each level $l = 0, \ldots, 5$, S is fixed at $293,496$. Of course it can very reasonably be argued that the number of samples should increase with the size of the solution space. On the other hand the spaces are so huge and grow so rapidly that using any non-exponential function of l to determine S seems utterly futile. In addition the evaluation time for each combination increases with the problem size and so the reasoning behind this choice of S was to find a value that would not take excessive time to run on the original uncoarsened graph and then use the same value on all but the coarsest levels. Finally to actually generate the random samples we initialised the partition to $\pi(v_i) = p$ for $i = 1, \ldots, |V|$ and then randomly set entries $\pi(v_j) = q$ until there were $|V|/2$ vertices in each set.

The test suite of problem instances thus consists of two sizes of graph, small and large; the small examples for enumeration tests each have 32 vertices, whilst the large ones which we evaluate by sampling are of size $|V| = 1,024$. For each size we then use four density subclasses: sparse, low-density, medium-density and high-density. Given a graph $G(V, E)$, we define the *edge density*, Δ, as the fraction or percentage of possible edges given by $\Delta = \frac{2|E|}{|V|(|V|-1)}$ so that a *complete* graph (where every vertex is adjacent to every other) with $\frac{|V|(|V|-1)}{2}$ edges has density 1.0 or 100% density. Although the distinction between sparse and low-density graphs is not always clear especially for small examples, 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 $|V|$, whilst 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 $|V|$.

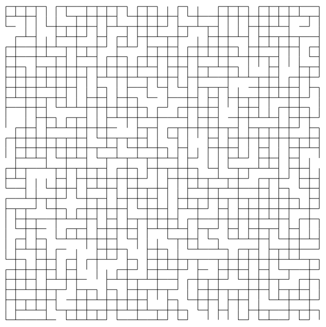


Figure 3: An example of the large sparse (semi-regular) GPP test instances

The sparse examples used here are a special case in that they were generated by constructing a grid (8×4 for $|V| = 32$ and 32×32 for $|V| = 1,024$) and then randomly deleting edges until the average vertex degree was 3 (i.e. so that the number of edges $|E| = 3|V|/2$). To some extent this simulates the semi-structured, mildly irregular nature of many partitioning examples, e.g. [13, 19, 35]. Figure 3 shows an example of a large sparse instance. Graphs in the other density classes had no intentional structure and were generated by randomly creating $\rho\frac{|V|(|V|-1)}{2}$ edges where $\rho = 0.1$ for low-density examples, 0.5 for medium, and 0.9 for high. Finally for each size and density we generated 16 graphs (using different random seeds) giving a total $16 \times 4 \times 2 = 128$ graphs.

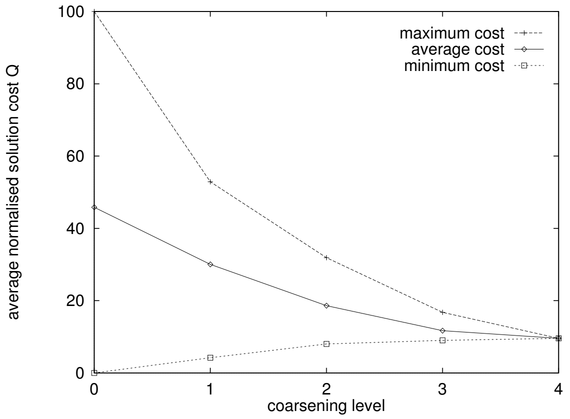
2.2.2 Results

In assessing the experimentation, one important issue that arises is how to normalise the results, firstly in order to average them and secondly to compare between different density classes. Initially, and as is common (e.g. [16]) we tried using the percentage excess over the optimal (or best-known) solution, which for a solution x with cost $f(x) = C_x$ is calculated as $Q(C_x) = 100\frac{C_x - C_{\min}}{C_{\min}}\%$, where $C_{\min}$ is the cost of the optimal solution. This works well for averaging but not so well for comparisons between the density classes. To see this consider that the small sparse instances ($|V| = 32$, $|E| = 48$) typically have values of the cost function between $C_{\min} = 3$ and $C_{\max} = 48$, where $C_{\max}$ is the value of the worst solution, and hence Q lies in the range 0% to 1,500% in excess of the optimal. By comparison, for a typical dense graph ($|V| = 32$, $|E| = 446$), $C_{\min} = 213$ and $C_{\max} = 248$ and so Q varies between 0% and 16.4%.

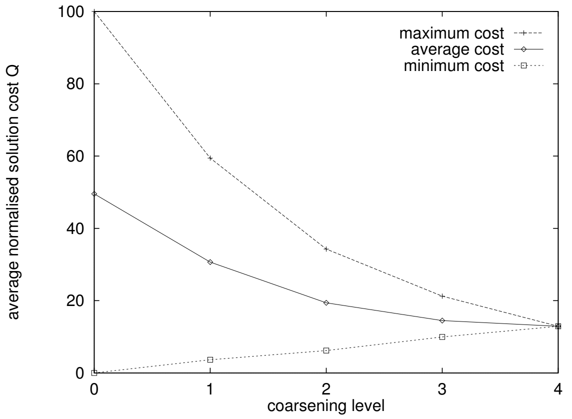
As an alternative we decided to use the excess over the optimal solution expressed as a percentage of the possible variation. Although clumsy to express in English this simply means that the normalised solution quality of a solution x with cost C_x is given by $Q(C_x) = 100\frac{C_x - C_{\min}}{C_{\max} - C_{\min}}\%$ and so $Q(C_{\min}) = 0\%$ and $Q(C_{\max}) = 100\%$.

Of course for the large sampled graphs this then gives the problem of estimating or computing $C_{\min}$ and $C_{\max}$. One possibility would be simply to use the highest and lowest values found by the sampling of the solution spaces. However as will be seen below, usually the lowest value of the cost function, which we would then use to estimate $C_{\min}$, is found on one of the coarser graphs. This would mean that the normalisation would not be independent of the coarsening and, since we believe the coarsening to be affected by the density, this would render objective comparisons between density subclasses impossible. A better alternative is to get a good estimate for the lowest value of the cost function by using a high quality partitioning algorithm, such as the iterated multilevel algorithm from [34]. It is still true that the results generated in this way are not entirely independent of graph coarsening, but the scheme was able to find far better partitions than random sampling and so we believe it gives a reasonable estimate of $C_{\min}$.

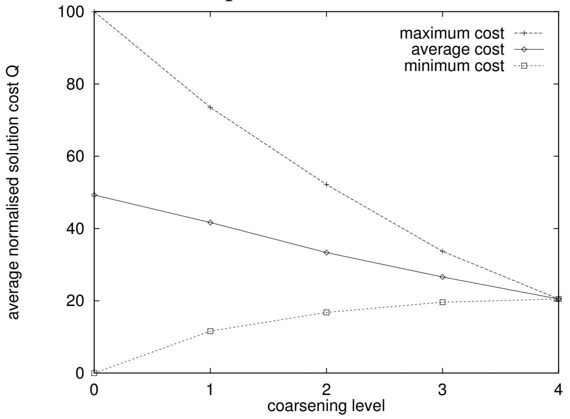
To estimate the upper bound, $C_{\max}$, of each graph $G(V, E)$ we can simply use its inverse $G'(V, E')$ where E' is the set of edges which do not exist in E (i.e. $(v_i, v_j) \in E'$ if $(v_i, v_j) \notin E$ and vice-versa). It is easy to show that a minimal cut-weight (min-cut) partition of G' is the same as a max-cut partition of G and so we just used the same iterated multilevel solver to compute a high quality partition π' of G' and then evaluated the cost of π' on G to estimate $C_{\max}$. Finally note that we validated these estimates by comparing them with the known values (computed by complete enumeration) of $C_{\min}$ and $C_{\max}$ for the small graphs; although we certainly would not claim that these results carry over to the large graphs, the computed estimates were correct in 90% of cases for $C_{\min}$ and 94% cases for $C_{\max}$.



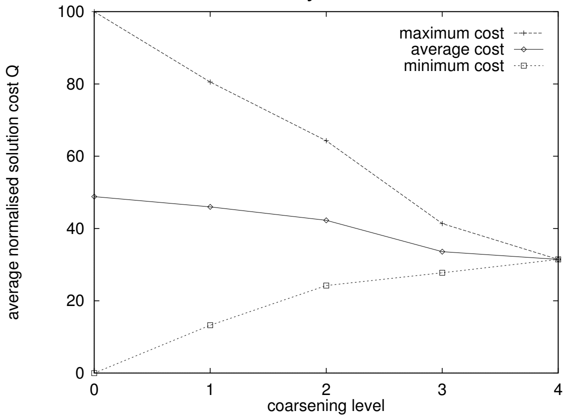
(a) sparse instances



(b) low-density instances



(c) medium-density instances

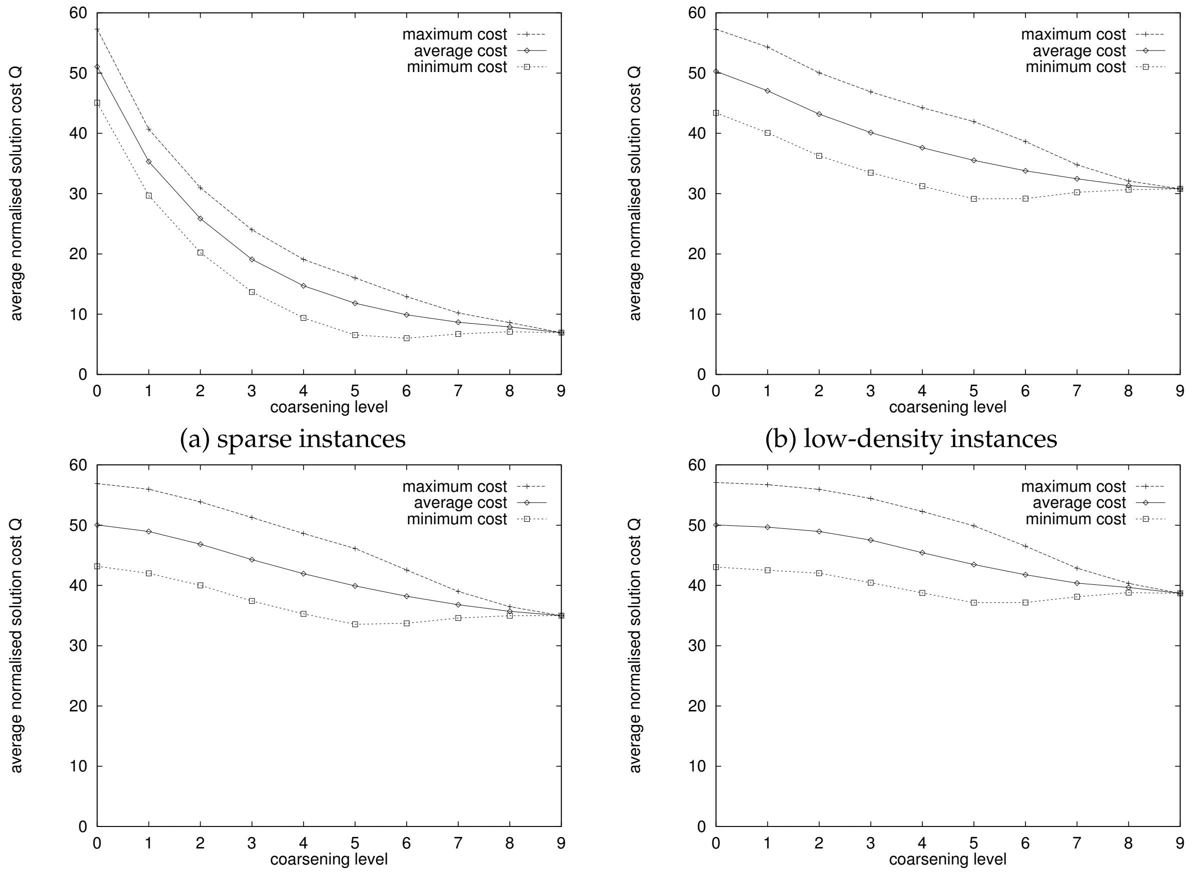


(d) high-density instances

Figure 4: Enumeration results for the small GPP instances

The results for the enumeration tests on the small graphs are shown in Figure 4. In each plot we take the maximum, average & minimum values of Q, the normalised solution quality, for each graph and each level and then average these values over the 16 graphs in each subclass. These averaged normalised values are then plotted against the coarsening level and the actual values are also shown in Table 1, on page 21. It is perhaps not immediately obvious what conclusions to draw from these plots, but we interpret the results as follows. Firstly, in all four density subclasses, the average value of the cost function decreases as the coarsening progresses (i.e. as the level increases). In other words the coarsening is filtering out the higher cost solutions at a greater rate than the lower cost ones. Ideally we would like it not to filter out any low cost or optimal solutions so that the minimum curves would be flat, although this would mean that the multilevel scheme would find the optimal solution at the end of the coarsening phase, which seems a little too much to hope for. However, a second important point to note is that the size of the coarsest spaces is relatively small (see above). Thus we might reasonably expect all but the most basic of refinement schemes (i.e. all those with the ability to escape local minima) to be able to find the best solution available at level 2 (which only has 35 possible solutions). This means that the refinement should already have a good initial solution for levels 1 and subsequently 0.

With regard to the effects of density, the observations, based on experimental data in [34], do seem to be borne out. Thus the more dense the problem, the shallower the apparent gradient of the average cost curves (or alternatively the steeper the gradient of the minimum cost curves). This is made clearer by looking at the figures in Table 1, especially with regard to the relative values of Q for the single solutions at level 4 in each subclass.



(a) sparse instances

(b) low-density instances

(c) medium-density instances

(d) high-density instances

Figure 5: Sampling results for the large GPP instances

Figure 5 and Table 2 show similar plots for the sampling results on the large graphs. Notice first of all that the y-axis only extends as far as 60% and that for the uncoarsened problems, level 0, the sampled results lie in the range 43% to 58%. This indicates how hopelessly inadequate the random sampling of the solution space is (because of its size) at getting anywhere near the best or worst values of the cost function. Furthermore the normalising values used for $C_{\min}$ & $C_{\max}$ are only estimates and the true values are very likely to be somewhat smaller & larger respectively. On the other hand the plots, and especially Figure 5(a), make a powerful point about how effective the coarsening is at filtering out the most costly solutions. Thus the best values are found by coarsening the problem and then sampling the much smaller coarsened spaces. Also, for levels 6 and above, where we take S_l samples in a space of size S_l, there is good reason to suppose that the sampling renders a fairly accurate representation of the solution space characteristics. In fact to validate this claim we ran the sampling tests on the small example graphs; although for these cases the sampling sometimes tends to miss out the highest and lowest cost solutions, it captures the average cost very well, often to 3 or 4 significant figures. Although we cannot claim that this is true for the results in Figure 5, it does seem likely that the average cost curve expresses the qualitative behaviour reasonably well.

Once again the fact that the average value of the cost function always decreases for each subclass demonstrates that the coarsening is acting as an effective filter. In addition, as for the small examples, increasing density is seen to have an adverse effect on the success of the filtration. Note finally that the marked difference between the sparse results, Figure 5(a), and the other results Figure 5(b)-(d) is primarily due to the sparsity (i.e. low-density instances in this class have $52,377$ edges whilst sparse ones have only $3,072$) and not the semi-structured nature of the sparse instances. We checked this by running some tests on sparse graphs of the same size generated completely randomly (as for the low, medium & high-density classes) and qualitatively the results were almost identical to those in Figure 5(a). To give an example result, the random sparse graphs have a final minimum, average & maximum value for Q at coarsening level 9 of 9.28% as compared with 6.24% for the grid-based sparse graphs as shown in Table 2.

3 The Travelling Salesman Problem

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

The TSP has been shown to be NP-hard, [10], 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, [17], 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, [24], 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 originally proposed by Martin, Otto & Felten, [25].

Even until recently, in spite of all the work on exotic and complex combinatorial techniques, Johnson & McGeoch, [17], concluded in 1997 that an iterated Lin-Kernighan scheme provides the highest quality tours for a reasonable cost and that variants of this algorithm are 'the most cost effective way to improve on Lin-Kernighan, at least until one reaches stratospheric running times'.

Note that it is often convenient to use graph notation for the TSP in which case we refer to the cities as vertices and the problem can be specified as a complete graph with weighted edges, i.e. there is an edge between every pair of cities and the weight of the edge specifies the distance between them.

3.1 A multilevel algorithm for the travelling salesman problem

Recently the multilevel paradigm has been applied to the TSP, [16, 32, 33]. Although the scheme is perhaps less intuitive than multilevel partitioning, clearly the LK algorithm or one of its variants should make a good refinement method. However, with no graph as such, how can the problem be coarsened?

In fact from [32] 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. As with partitioning, this process is equivalent to restricting the solution space (to all tours which contain the edge (c_a, c_b)) and in fact by fixing many distinct edges in one coarsening step we are again filtering the solution space.

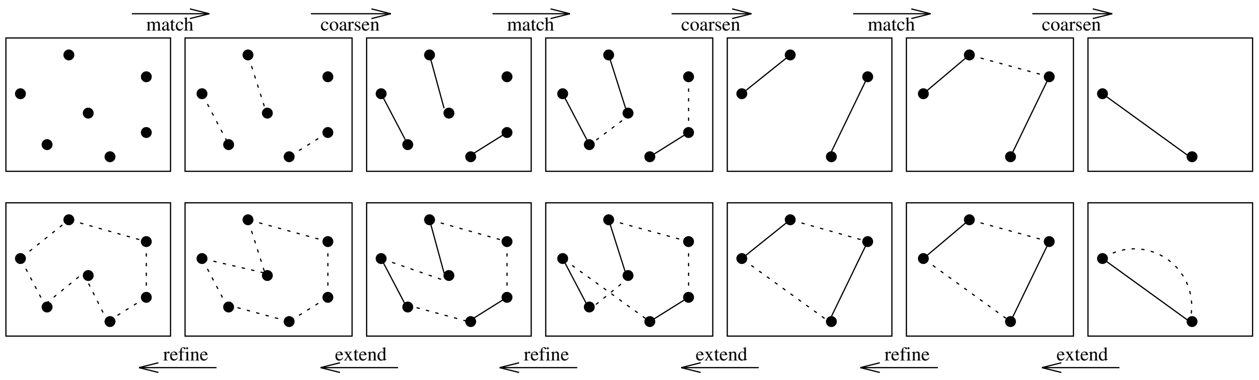


Figure 6: An example of a multilevel TSP algorithm at work

Figure 6 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 straightforward; we simply expand all fixed edges created in the corresponding coarsening step and add the free edges to give an initial tour for the refinement process. The refinement algorithm then attempts to improve on the tour (without changing any of the fixed edges) although notice that for the first refinement step no improvement is possible. The final tour is shown at the bottom left of the Figure; note in particular that fixing any edge during coarsening does not force it to be in the final tour since for the final refinement step all edges are free to be changed. However, fixing an edge early on in the coarsening does give it fewer possibilities for being flipped.

3.1.1 Coarsening

The implementation of this coarsening process in which vertices are matched and edges fixed between them is fully described in [32]. In fact it is more convenient for the data structure to use edge objects and so in practice a matching of edges is created at each level (in much the same way that a matching of vertices is created for the multilevel partitioning algorithm). Initially each edge is of zero length and has the same vertex at either end; however after the first coarsening most edges will have different vertices at either end. Figure 7(a) shows an example of this where the edges (v_1, w_1) & (v_2, w_2) are matched together.

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 7(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 have been 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 7(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

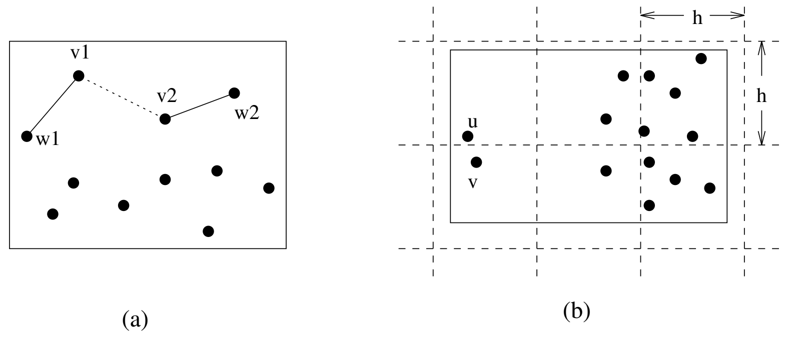


Figure 7: TSP matching examples

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, [32].

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

3.1.2 Refinement

Once again, for the purposes of this paper we are not particularly interested the refinement process, but outline it here briefly. Firstly, the initialisation of a solution is trivial at the coarsest level and simply consists of adding an edge between the two vertices to complete the tour. At each subsequent change of levels the tour is extended to the next level down by expanding all fixed edges created in the corresponding coarsening step and adding the free edges to give an initial tour for the refinement process.

TSP tour refinement can then take place 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, [8], 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, [23], exchanges 3 edges at a time. The Lin-Kernighan (LK) algorithm, [24], 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 surveys of Johnson & McGeoch, [16, 17].

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, [25], 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. [2, 15].

3.2 Experimental results

Once again we aim to assess the effectiveness of the multilevel coarsening process by examining the hierarchy of solution spaces that it produces. In fact the difficulties which arose for the partitioning problem in generating balanced solutions do not occur for the TSP because there are no constraints to satisfy. However at a more technical level it was not immediately obvious how to enumerate all the solutions of the multilevel landscape.

For uncoarsened instances, each solution can be specified by a permutation the vertices. Indeed there is no loss of generality in always fixing the vertex with index 1 in the first position of the permutation and so for a problem instance with $N = |V|$ vertices, the total number of solutions to examine is $(N-1)!$. In fact this still generates double the number of solutions we require, since reversing the order of a permutation gives the same tour, but we were unable to filter out duplicates easily and so we evaluate each tour twice.

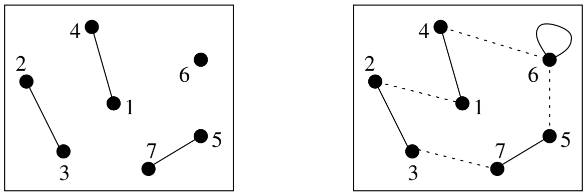


Figure 8: A small coarsened TSP instance

Unfortunately the coarsening and hence the introduction of fixed edges complicate the enumeration significantly. To see this, consider the problem instance shown in Figure 8(a) and try listing by hand all the possible tours. In fact the solution lies (as with the matching data-structures of §3.1.1) in regarding the instance as a group of fixed edges (where some of the edges may have the same vertex at either end). This is illustrated in Figure 8(b) where the four edges, $(2,3)$, $(1,4)$, $(6,6)$ & $(5,7)$, are shown as solid lines. To define a tour we then need to list a permutation of the edges and to specify the direction in which each non-trivial edge is traversed (where we regard a trivial edge as one with the same vertex at either end). For an instance with $|V|$ vertices and $|E|$ edges, it is not difficult to see that D, the number of non-trivial edges and hence directions required, is given by $D = |V| - |E|$. We can then specify the directions as a vector of binary-valued variables d_e where $d_e = -1$ means visit the vertices of edge e highest index first, then lowest index, and $d_e = +1$ means visit them lowest to highest. The tour shown in Figure 8(b) could then be written as the permutation of edges $[(2,3),(5,7),(6,6),(1,4)]$ with the directions vector $[+1,-1,-1]$ (where it is understood that each successive direction is applied whenever a non-trivial edge is encountered in the permutation). For a given permutation of edges, all possible solutions are generated by picking every possible combination of $+1$'s and -1's. Thus for the example in Figure 8, there is one combination with no -1's (i.e. $[+1,+1,+1]$), three with one -1 (i.e. $[-1,+1,+1]$, $[+1,-1,+1]$, $[+1,+1,-1]$), three with two -1's and one with three -1's. In other words the total number of solutions for each permutation of edges is $\sum_{r=0}^{r=D} {}^{D}C_r$ where once again ${}^{n}C_r = n!/r!(n-r)!$. In fact it can be shown by induction that $\sum_{r=0}^{r=D} {}^{D}C_r = 2^D$ (alternatively consider that we have a vector of length D, each entry of which can take one of two possible solutions). Thus the total number of possible solutions for an instance with $|V|$ vertices and $|E|$ edges (both trivial and non-trivial) is $(|E|-1)! \cdot 2^{(|V|-|E|)}$ (since once again there is no loss of generality in always fixing one edge as the first of the permutation). Note that this total also accords with uncoarsened problems because for such cases $|V| = |E|$.

The experimentation in [32] suggests that, as with the partitioning problem, the effectiveness of the multilevel algorithm for the TSP does have some link with the density, or more accurately the distribution, of the vertices. In particular the coarsening seems to work well in combination with distributions where the vertices are clustered together in groups rather than uniform distributions. There is no exact analogue of edge density in the TSP but we can compute a density indicator, for Euclidean TSP instances at least, by superimposing a grid (much as in §3.1.1) and counting the relative numbers of occupied and unoccupied cells. Thus if we determine $x_{\max}$ & $x_{\min}$, the highest & lowest values of x (so that the x coordinates of the vertices lie in the interval $[x_{\min}, x_{\max}]$), then let the grid spacing be $h_x = (x_{\max} - x_{\min})/\sqrt{N}$ in the x direction and similarly for h_y in the y direction. This gives us a rectangular grid with approximately $\sqrt{N} \times \sqrt{N} = N'$ cells and we can then define the density Δ as the number of occupied cells, N_o, over the total number of cells, $\Delta = N_o/N'$. Random instances with a uniform distribution will then typically have a high density with most of the cells occupied, whilst clustered instances, whether randomly generated or real-life instances, will typically have a much higher incidence of unoccupied cells and hence a much lower density.

3.2.1 The testing

Once again we use two sizes of problem instance and two methods of testing; small instances with complete enumeration of the solution space hierarchy and large instances with random sampling. Because the sizes of the solution spaces are even larger, relative to $|V|$, than those for partitioning we limit the small instances to just $|V| = 12$, meaning that the solution space of the uncoarsened problem is of size $(|V|-1)! = 39,916,800$. The large instances have $|V| = 512$. For each size we then randomly generated (with different seeds) 16 'low-density' clustered instances and 16 'high-density' uniformly distributed instances using code written by Johnson, Bentley & McGeoch for the 8th DIMACS implementation challenge[2]. In particular the clustered instances were generated using `portcgen` which we modified to always produce a minimum of 4 clusters (rather than $|V|/100$), although in a couple of the small cases some of the clusters (whose centre is generated randomly) were superimposed meaning that the actual number of apparent clusters can be just 3 or even 2. The uniformly generated clusters were generated with the `portgen` code which we used unmodified. The density indicator described above then gave the densities of the small instances as in the range 25% to 43.75% for the clustered examples and 37.5% to 68.75%. However with only 12 vertices it is not surprising that the values vary so much and for the large examples the ranges were a much more homogeneous 17.39% to 28.73% for clustered instances and 59.17% to 64.46% for the uniform ones.

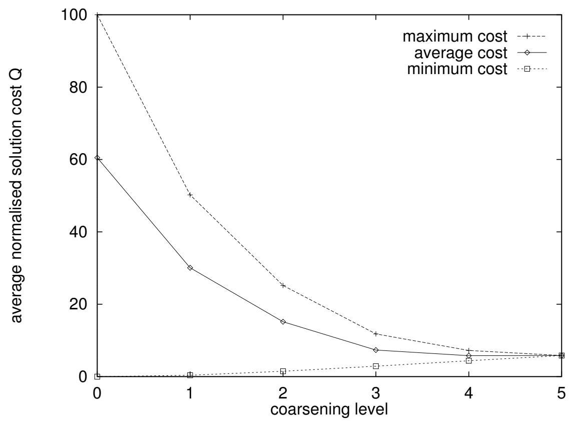
Because we have no need for the forced matching, used in the partitioning case to avoid problems with the balance constraint, the coarsening rate is not uniform across the problem instances. For typical example though, the size of the problems and the solution spaces, expressed as a triple $(|V_l| : |E_l| : S_l)$, are for level $l = 0$, $(12 : 12 : 39,916,800)$; level 1, $(12 : 7 : 23,040)$; level 2, $(8 : 5 : 192)$; level 3, $(5 : 3 : 8)$; level 4, $(4 : 2 : 4)$; and level 5, $(2 : 1 : 1)$. We used a recursive code developed in house to generate the permutations and then, for the coarsened spaces, the same combination algorithm, [14], as in §2.2.1, to generate all possible combinations of directions for the fixed edges. For the sampling, using a similar reasoning to §2.2.1, we set the number of samples for level l to be the smaller of $(|E_l| - 1)! \cdot 2^{|V_l| - |E_l|}$ and $12!/|V_0|$ where $|V_0|$ is the size of the original instance. For the large instances this means that, for all but the coarsest levels, the number of samples is fixed at $935,550$. The actual sampling algorithm swaps $|E_l|$ randomly picked pairs of edges from a permutation vector, then randomly picks the number of -1 directions and generates a appropriate combination of directions as above, §2.2.1.

3.2.2 Results

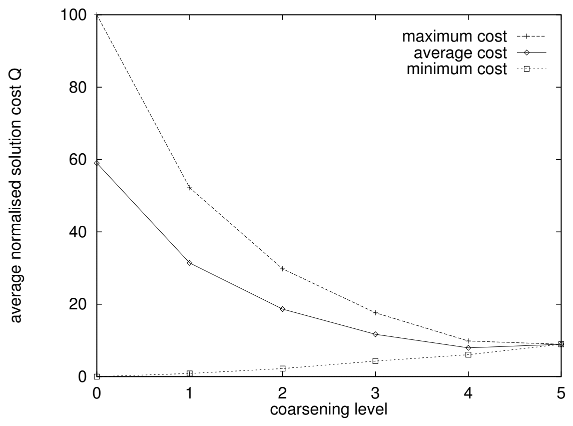
Once again, as with partitioning, and for the same reasons, in order to normalise the results we decided to use the excess over the optimal solution expressed as a percentage of the possible variation, i.e. the normalised solution quality of a solution x with cost C_x is given by $Q(C_x) = 100\frac{C_x - C_{\min}}{C_{\max} - C_{\min}}\%$. For the large sampled graphs we were then able to compute $C_{\min}$ using the optimisation package `concorde`[3], [1]. We were also able to compute $C_{\max}$ using a similar trick to the partitioning case and finding the optimal solution to the inverse of each instance. Thus for a problem instance P, if $d(v_i, v_j)$ is the distance between vertices v_i and v_j, we can construct P', the inverse of P, by explicitly writing out the distance matrix of P' where the entry in the i^{th} row and j^{th} column is given by 0 if $i = j$ and by $K - d(v_i, v_j)$ if $i \neq j$ for some constant K (in fact we used $K = d_{\max} = \max_{i,j} d(v_i, v_j)$ so that all distances were non-negative). It is then easy to prove that the optimal tour for P' gives the worst possible tour for P.

[2]see `http://www.research.att.com/~dsj/chtsp/`

[3]available from `http://www.keck.caam.rice.edu/concorde/download.html`



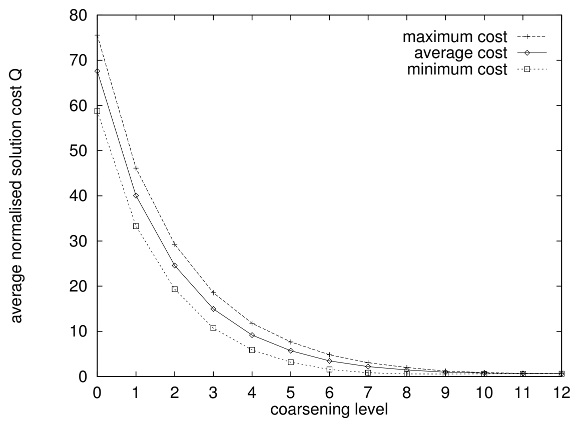
(a) instances with clustered distribution



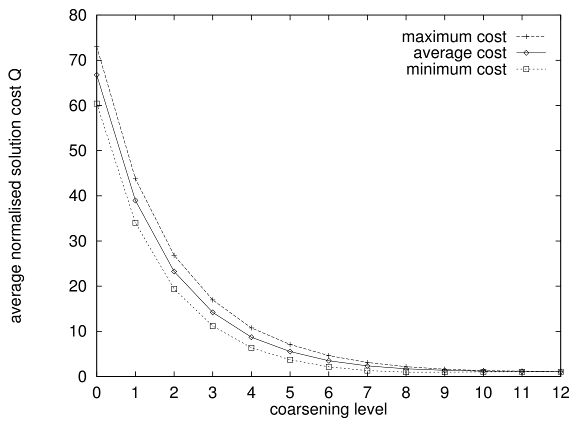
(b) instances with uniform distribution

Figure 9: Enumeration results for the small TSP instances

The enumeration results for the small examples are illustrated in Figure 9 (and listed in Table 3). Once again these show how effective the coarsening is at filtering out the higher cost solutions, in fact even more effective than for the partitioning problem. However perhaps this is more an indication of the characteristics of each problem since the TSP has the unusual feature that it is easy to compute very accurate lower bounds on solution cost and relatively easy to compute solutions within a few percent of this bound, e.g. [17, 16]. In addition, once again the 'density' does appear to affect the success of the coarsening, although this is only really noticeable by studying the figures in Table 3.



(a) instances with clustered distribution



(b) instances with uniform distribution

Figure 10: Sampling results for the large TSP instances

Figure 10 and Table 4 meanwhile give the sampled results for the large instances. Note that the y-axis only extends as far as the 80% point, and again we can see how inadequate the sampling is at picking up the highest and lowest cost solutions in the original problem (level 0). In fact it is considerably worse than for partitioning, although this may be related to the fact that the solution spaces for TSP instances of 512 vertices are ***much*** larger (approximately $6.8 \times 10^{1,163}$) than those for graph partitioning instances of $1,024$ vertices (approximately 2.2×10^{306}). What is remarkable however is the fact that the coarsening manages to get so close to the optimal solutions; as can be seen from Table 4 the final coarsest spaces are just 0.63% & 1.07% away for the clustered and uniform instances respectively. For practitioners familiar with the TSP however, it should be noted that these values are in fact around 25% in excess of the optimal solution when expressed in the more common manner $Q(C_x) = 100\frac{C_x}{C_{\min}}\%$, thus indicating that the refinement algorithm does have some non-trivial work to reach the high quality solutions found in [32, 34]. Finally, once again the ‘density’ does appear to affect the success of the coarsening (see Table 3), although not to the same extent as for partitioning.

4 Summary and Future Work

4.1 Conclusions

We have explored the multilevel landscapes produced by coarsening for the graph partitioning and travelling salesman problems. Plots of minimum, average and maximum cost solutions indicate that generally the coarsening is filtering out higher cost solutions at a much faster rate than low cost ones thus rendering small coarsened spaces populated by relatively low cost solutions. Of course this is no guarantee that subsequent refinement may be able to reach an even better solutions, but experimental data elsewhere, e.g. [34], appears to indicate that this is the driving force behind the multilevel combinatorial schemes.

We have also looked at how the density of the problem instance affects the multilevel landscape. Again, and as suggested in [34], the increasing density appears to inhibit the filtering process somewhat and more of the low cost solutions are lost during the coarsening.

4.2 Future research

The experimentation described in this paper has set out a framework by which we can explore the effect of coarsening on the multilevel landscapes. An interesting possibility, not explored here, might be to utilise this framework with the aim of developing and/or testing better (although possibly more expensive) matching algorithms or at least matching schemes more suited to medium/high-density problems.

It seems likely that finding a good matching (one which will aid the refinement) for such instances is more challenging because of the large number of candidates matches and the difficulty of choosing between them. This choice is usually made using some empirically-based prioritisation and hence inherent to the matching algorithms is a built in cost function (e.g. for the GPP – maximise the edge weight between matched pairs; for the TSP – minimise the distance between matched pairs) which is not directly related to the cost function of the problem in question. There is also plenty of evidence, for partitioning at least, that judicious choice of the matching cost function can strongly affect the final solution quality, e.g. [18, 36]. However, it is not always clear which cost function the matching should be aiming to optimise (apart from maximising the number of matches) and for instance the heavy edge matching scheme, [18], commonly used for partitioning, can only operate on weighted graphs and hence cannot be applied for the first coarsening step if the original graph is not weighted.

The computation of matchings of this type is known as the minimum-weight perfect matching problem where here the weight refers to the matching cost function. Polynomial algorithms which can compute optimal matchings are available, e.g. [7], 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 the hierarchy of landscapes.

A second possibility might be the sort of locally optimal matching scheme proposed by Monien *et al.* for graph partitioning, [26]. Currently in the example matching schemes used above (§2.1.1 & §3.1.1), a vertex v_0 picks a preferred candidate v_1 and they are matched regardless of whether v_1 has available matches which are more advantageous than v_0. Hence, in the scheme of Monien *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. Once again it would be interesting to test the effect of this scheme using the methodology applied here.

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

[8] G. A. Croes. A Method for Solving Traveling Salesman Problems. *Oper. Res.*, 6:791–812, 1958.

[9] 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.

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

[11] F. Glover, M. Laguna, and R. Marti. Fundamentals of Scatter Search and Path Relinking. *Control & Cybernetics*, 39(3):653–684, 2000.

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

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

[14] J. Hopley. Algorithm 152 Nexcom. *Comm. ACM*, 6(7):385, 1963.

[15] 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.

[16] D. S. Johnson and L. A. McGeoch. Experimental Analysis of Heuristics for the STSP. In G. Gutin and A. Punnen, editors, *The Travelling Salesman Problem and its Variations*. (To appear[4]).

[17] 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.

[18] 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.

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

[4]available from `http://www.research.att.com/~dsj/chtsp/`

[20] 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).

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

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

[23] S. Lin. Computer Solutions of the Traveling Salesman Problem. *Bell Syst. Tech. J.*, 44:2245–2269, 1965.

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

[25] 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.

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

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

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

[29] 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.

[30] 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.

[31] 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.

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

[33] C. Walshaw. A Multilevel Lin-Kernighan-Helsgaun Algorithm for the Travelling Salesman Problem. Tech. Rep. 01/IM/80, Comp. Math. Sci., Univ. Greenwich, London SE10 9LS, UK, September 2001.

[34] C. Walshaw. Multilevel Refinement for Combinatorial Optimisation Problems. Tech. Rep. 01/IM/73, Comp. Math. Sci., Univ. Greenwich, London SE10 9LS, UK, June 2001.

[35] 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).

[36] 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).

Table 1: Enumeration results for the small GPP instances

	sparse			low density			medium density			high density		
level	Min	Avg	Max	Min	Avg	Max	Min	Avg	Max	Min	Avg	Max
0	0.00	45.83	100.00	0.00	49.55	100.00	0.00	49.29	100.00	0.00	48.83	100.00
1	4.21	30.03	52.89	3.64	30.67	59.44	11.61	41.67	73.50	13.26	46.00	80.54
2	8.03	18.62	31.88	6.20	19.41	34.28	16.79	33.34	52.17	24.22	42.27	64.28
3	9.01	11.70	16.77	9.96	14.48	21.24	19.62	26.59	33.68	27.75	33.59	41.36
4	9.57	9.57	9.57	12.92	12.92	12.92	20.50	20.50	20.50	31.46	31.46	31.46

Table 2: Sampling results for the large GPP instances

	sparse			low density			medium density			high density		
level	Min	Avg	Max	Min	Avg	Max	Min	Avg	Max	Min	Avg	Max
0	45.09	51.07	57.29	43.40	50.29	57.27	43.18	50.03	56.89	43.05	50.03	57.08
1	29.67	35.31	40.70	40.09	47.06	54.32	42.00	48.96	55.96	42.52	49.67	56.72
2	20.23	25.86	30.98	36.27	43.19	50.03	40.02	46.84	53.86	42.03	48.95	55.95
3	13.68	19.09	24.00	33.47	40.13	46.88	37.42	44.28	51.27	40.45	47.52	54.43
4	9.38	14.71	19.07	31.25	37.61	44.25	35.28	41.95	48.61	38.76	45.43	52.27
5	6.54	11.82	16.02	29.14	35.52	41.95	33.55	39.92	46.13	37.14	43.46	49.89
6	6.02	9.89	12.91	29.17	33.78	38.64	33.72	38.20	42.56	37.16	41.78	46.52
7	6.72	8.66	10.21	30.22	32.46	34.78	34.60	36.80	39.00	38.12	40.38	42.86
8	7.09	7.88	8.60	30.66	31.32	32.10	34.98	35.70	36.47	38.81	39.67	40.34
9	6.94	6.94	6.94	30.80	30.80	30.80	34.99	34.99	34.99	38.69	38.69	38.69

Table 3: Enumeration results for the small TSP instances

	low density			high density		
level	Min	Avg	Max	Min	Avg	Max
0	0.00	60.51	100.00	0.00	59.02	100.00
1	0.39	30.08	50.23	0.88	31.42	52.19
2	1.48	15.16	25.20	2.24	18.66	29.78
3	2.89	7.34	11.80	4.30	11.69	17.64
4	4.40	5.80	7.23	6.06	7.95	9.84
5	5.83	5.83	5.83	8.95	8.95	8.95

Table 4: Sampling results for the large TSP instances

	low density			high density		
level	Min	Avg	Max	Min	Avg	Max
0	58.75	67.59	75.56	60.40	66.77	73.01
1	33.29	40.04	46.13	34.04	38.97	43.77
2	19.35	24.57	29.28	19.38	23.25	26.84
3	10.70	14.96	18.54	11.20	14.20	16.96
4	5.87	9.18	11.79	6.37	8.71	10.77
5	3.17	5.72	7.66	3.71	5.53	7.11
6	1.56	3.46	4.81	2.14	3.50	4.63
7	0.85	2.20	3.07	1.31	2.36	3.11
8	0.58	1.46	2.00	0.97	1.72	2.18
9	0.56	0.96	1.21	0.97	1.38	1.63
10	0.60	0.77	0.89	1.03	1.19	1.32
11	0.62	0.66	0.71	1.04	1.13	1.20
12	0.63	0.63	0.64	1.06	1.07	1.08