Noname manuscript No.
(will be inserted by the editor)

Tackling the Maximum Happy Vertices Problem in Large Networks

Dhananjay Thiruvady*, Rhyd Lewis and Kerri Morgan

the date of receipt and acceptance should be inserted later

Abstract In this paper we consider a variant of graph colouring known as the *Maximum Happy Vertices* (MHV) problem. This problem involves taking a graph in which a subset of the vertices have been preassigned to colours. The objective is to then colour the remaining vertices such that the number of happy vertices is maximised, where a vertex is considered happy only when it is assigned to the same colour as all of its neighbours. We design and test a tabu search approach, which is compared to two existing state of the art methods. We see that this new approach is particularly suited to larger problem instances and finds very good solutions in very short time frames. We also propose a algorithm to find upper bounds for the problem efficiently. Moreover, we propose an algorithm for imposing additional precoloured vertices and are hence able to significantly reduce the solution space. Finally, we present an analysis of this problem and use probabilistic arguments to characterise problem hardness.

Keywords: Tabu search, Graph Colouring, Maximum Happy Vertices Problem.

Dhananjay Thiruvady
School of Information Technology, Deakin University, Geelong VIC, Australia.
E-mail: dhananjay.thiruvady@deakin.edu.au
*Corresponding author

Rhyd Lewis
School of Mathematics, Cardiff University, Cardiff, Wales
E-mail: LewisR9@cardiff.ac.uk

Kerri Morgan
School of Information Technology, Deakin University, Geelong VIC, Australia.
E-mail: kerri.morgan@deakin.edu.au

1 Introduction

In this study, we consider the *Maximum Happy Vertices* (MHV) problem. The MHV problem is a variant of the well-known graph vertex colouring problem and is known to be NP-hard in general (Li and Zhang, 2015). In traditional graph colouring, colours need to be assigned to all of the vertices such that no two adjacent vertices are assigned to the same colour, while the number of colours being used is minimised. Practical problems such as resource allocation, frequency assignment, university timetabling, sports scheduling, etc. can be modelled as graph colouring since conflicting constraints and resource limits can easily be represented in this framework (Carter et al., 1996; Lewis, 2015; Lewis and Thompson, 2015; McCollum et al., 2010). Recently interest in the MHV problem has been increasing in the literature (Agrawal, 2018a; Li and Zhang, 2015; Lewis et al., 2019). In this problem we seek to maximise the number of *happy* vertices in a graph, where a vertex is defined as happy if and only if it is assigned to the same colour as all of its neighbouring vertices. This representation has applications, for example, in detecting homophily in social networks and for network clustering problems, particularly when certain vertices are already known to belong to certain groups or share certain characteristics.

Currently there are relatively few studies investigating methods for the MHV problem. In 2015, Li and Zhang (2015) proved this problem to be NP-hard and provided two simple approximation algorithms. The later study by Zhang et al. (2018) then proposed an improved approximation algorithm based on solving a linear programming relaxation of the MHV problem and then rounding non-integer decision variables to form a feasible solution. More recently, Agrawal (2018a) has shown that the complexity of this problem is W[1]-hard when parameterised by l, where l is a minimum bound on the number of happy vertices. The problem is now also known to be polynomially solvable for trees, when the number of available colours is fewer than three, and for graphs with bounded treewidths (Agrawal, 2018b; Aravind et al., 2016; Li and Zhang, 2015). Lewis et al. (2019) have also presented methods for generating upper and lower bounds on the optimal number of happy vertices in a graph. In addition they also demonstrate how problems can be subdivided and compare different constructive algorithms to an integer programming (IP) method and a metaheuristic based on the construct, merge, solve and adapt (CMSA) methodology (Blum and Roli, 2003).

Practical applications of the MHV problem can be seen in a variety of areas. Li and Zhang (2015), for example, cite a study in which large network of academic papers was taken (where edges indicate a citation of one paper in another). In this network, titles and abstracts are listed for all papers, but keywords (vertex colours) are only known in around 5% of cases. They then found that allocating colours to the remaining papers while maximising the number of happy vertices led to the correct prediction of papers' subject areas in 69% of cases. Lewis et al. (2019) have also noted various applications of the MHV problem with social networks, where vertices are used to represent people, and edges denote friendships. For example, imagine we need to partition a group of people into teams, but some of these people have already been allocated to these teams. Solving the MHV problem leads to a solution in which the number of people in teams containing all of their friends is maximised. Similar tasks might also arise in the design of seating plans for large social events such as a wedding or gala dinner (Lewis and Carroll, 2016). More generally, the MHV problem has applications in clustering based problems; specifically, where some objects have been assigned to clusters, and the aim is to assign the remaining objects to these clusters such that related objects occur in the same cluster (Everitt et al., 2011).

This study aims to find a method that performs well on difficult-to-solve problem instances, is fast, and that also scales well compared to other approaches. For this purpose we propose a tabu search approach (Glover and Laguna, 1997, 1999), which is a local search method that incorporates mechanisms to avoid cycling at local optima. Studies using tabu search with graph colouring and related problems have previously shown much promise (Di Gaspero and Schaerf, 2001; Mabrouk et al., 2009; Zufferey et al., 2008). Mabrouk et al. (2009) have also demonstrated that a parallel hybrid of an evolutionary and tabu search algorithm produces very good solutions to benchmark datasets. In another study, Zufferey et al. (2008) considered a satellite range scheduling problem, for which they applied efficient heuristic techniques for graph coloring, including tabu search. In addition to proposing and testing this tabu search method against other methods, we further analyse the problem, allowing us to propose efficient methods for reducing the search space and also to make some probabilistic arguments about what makes a problem instance non-trivial. Specifically, there are four contributions in this study: (a) a metaheuristic-based approach for the MHV problem, namely tabu search, (b) algorithms for efficiently computing upper bounds to the MHV, (c) an algorithm for precolouring additional vertices and (d) characterising hardness of problem instances via probabilistic arguments.

Section 2 analyses the problem and defines methods for finding upper bounds efficiently. Section 3 provides details of tabu search including an efficient implementation of the approach for the MHV problem. Section 4 details the experiments conducted and the results obtained from these experiments. The paper concludes in Section 5.

2 The Maximum Happy Vertices Problem

Given a graph $G = (V, E)$ with n vertices and m edges, let $\Gamma(v)$ be the neighbourhood of vertex $v \in V$ and let $c : V \to \{1, \dots, k\}$ be a complete colouring of the vertices in G. A vertex v is *happy* if $c(v) = c(u)$ for all $u \in \Gamma(v)$, otherwise it is *unhappy*. That is, a vertex v is happy only if all its neighbours are assigned the same colour $c(v)$.

Given a partial colouring $c' : V' \to \{1, \dots, k\}$ where $V' \subseteq V$, the objective in the MHV problem is to extend c' to a complete colouring $c : V \to \{1, \dots, k\}$

such that the number of happy vertices is maximised. Any vertex that is not precoloured in the problem instance is referred to as a *free* vertex. The number of happy vertices in a globally optimal solution is denoted by $H(G)^*$. A small example of this problem is shown in Figure 1.

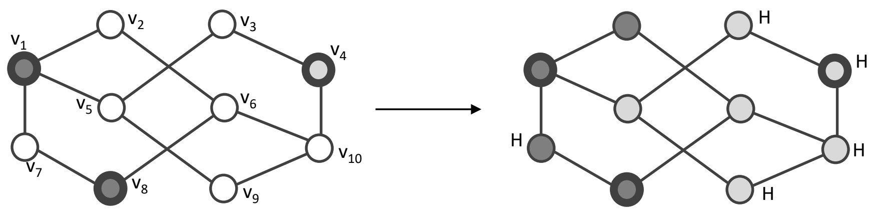


Fig. 1 Left: An example MHV problem instance with $n = 10$ vertices, $m = 8$ edges, $k = 2$ colours, and $|V'| = 3$ precoloured vertices (free vertices are shown in white). Right: A candidate solution featuring five happy vertices, indicated by H's.

2.1 Upper Bounds

Lewis et al. (2019) propose a method for computing an upper bound $\bar{H}(G)$ on the optimal number of happy vertices $H(G)^*$. This operates by identifying *unhappy paths* in a graph and removing them one by one. An unhappy path is defined as a simple path whose terminal vertices are precoloured differently and whose internal vertices (if any) are free. For example, in Figure 1 (left) unhappy paths include (v_1, v_5, v_3, v_4) and (v_4, v_{10}, v_6, v_8). It is clear that at least one vertex on any unhappy path must be unhappy in any candidate solution.

Lewis et al.'s method for computing an upper bound is described in the steps below. The basic idea is to identify and delete unhappy paths while, at the same time, maintaining a count of the maximum possible number of unhappy vertices x in the graph. At the start of the process x is set to zero.

1. Identify an unhappy path of length one (edge) $P = (u, v)$. If such a path does not exist, go to Step 3.
2. If both u and v have not yet been counted, set $x = x + 2$; otherwise if only one of u or v has been counted, set $x = x + 1$. Now mark u and v as having been counted and remove the edge $\{u, v\}$ from G. Return to Step 1.
3. Identify an unhappy path P of length two (edges) or more. If such a path does not exist, return $\bar{H}(G) = n - x$. This is the upper bound.
4. Let u and v be the terminal vertices of P. If both u and v have not yet been counted, set $x = x + 2$; otherwise if only one of u or v has been counted, set $x = x + 1$. Now mark u and v as having been counted and remove all of P's internal vertices from G. Return to Step 3.

The first two steps of this procedure focus on unhappy paths of length one, which are removed from G with x being updated accordingly. A similar process is then used in Steps 3 and 4 by eliminating all free vertices in paths of length two or more. Note that vertices are never counted more than once in x due to the conditions included in Steps 2 and 4.

For the upper bounds reported in this current paper, breadth first search (BFS) is used to select the shortest unhappy path in the current graph. This results in the smallest possible number of vertices being removed from G at each step, which leads to potentially a larger number of iterations and therefore more increments to x. While the method used here is effective (see Section 4), Lewis et al. note that other strategies might also be applied in practice.

2.2 Constructive Heuristics

As noted earlier, in their 2015 paper Li and Zhang (2015) proposed two simple constructive heuristics for the MHV problem. As we will see, these methods can be used to produce initial solutions and lower bounds on $H(G)^*$. In addition, the notation used to describe these heuristics is also helpful for the analyses given in this paper. We therefore review both heuristics here.

The first constructive heuristic proposed by Li and Zhang (2015) is the so-called Greedy-MHV method. This operates by simply taking all free vertices, assigning them to the same single colour and then calculating the resultant number of happy vertices. These steps are repeated using all of the k colours, with the best of these solutions then being used.

The second heuristic, Growth-MHV, involves placing labels on vertices and then using these to determine the ordering in which free vertices are coloured. Let v be a vertex assigned to colour i. The three possible labels for coloured vertices are defined as follows:

- v is a U-vertex if it is destined to be unhappy (i.e., at least one of its neighbours has been assigned to a colour $j \neq i$);
- v is a P-vertex if it has the *potential* to be happy (i.e., some neighbours of v are currently uncoloured, and any coloured neighbours have colour i);
- otherwise v is an H-vertex (i.e., it is happy because all neighbours of v are also assigned to colour i).

On the other hand, if v is not yet assigned to a colour, it can have one of four possible labels:

- v is an LP-vertex if it is adjacent to a P-vertex;
- v is an LH-vertex if it is not adjacent to a P-vertex, but has the potential to be happy (i.e., it is adjacent to U-vertices of only one colour);
- v is an LU-vertex if it is not adjacent to a P-vertex and is destined to be unhappy (i.e., at least two of its neighbours are assigned to different colours);
- otherwise v is an LF-vertex (and not adjacent to any coloured vertex).

Given these labels, the Growth-MHV heuristic operates by colouring one or more free vertices at each iteration, prioritising vertices with certain labels.

This process continues until all free vertices have been coloured. The rules are as follows:

1. If G contains a P-vertex v assigned to colour i, assign all neighbours of v to colour i;
2. Else if G contains an LH-vertex v, let i be the colour of an adjacent U-vertex, and assign v and all of its uncoloured neighbours to colour i;
3. Else if G contains an LU-vertex v, let i be the colour of any U-vertex adjacent to v, and assign v to colour i;
4. Else if G contains an LF-vertex v, assign v to an arbitrary colour;
5. Else all of G's vertices have been coloured, so end.

Note that when a vertex or set of vertices is coloured using these rules, labels of other vertices in the graph also need to be recalculated for subsequent iterations.

The approximation ratios of the Greedy-MHV and Growth-MHV heuristics are known to be $1/k$ and $\Omega(1/\Delta^3)$ respectively, where Δ is the maximum degree of any vertex in the graph (Li and Zhang, 2015).

2.3 Conditions Needed for all Vertices to be Unhappy

In this subsection we examine the conditions that are necessary for a problem instance's optimal solution to contain no happy vertices, giving $H(G)^* = 0$. First note the following simple thereom.

Theorem 1 *$H(G)^* = 0$ if and only if all precoloured vertices in G are U-vertices and all free vertices in G are LU-vertices.*

Proof By definition, U- and LU-vertices cannot be happy, so if all vertices of a graph are of this type, $H(G)^* = 0$.

Now consider the case where v is precoloured with colour i but is not a U-vertex. This means that it is either an H-vertex (implying that it is happy), or a P-vertex (meaning that it can be made happy by assigning all of its uncoloured neighbours to colour i). In either case this leads to $H(G)^* \geq 1$

Similarly, suppose that v is a free vertex but not an LU vertex. Therefore it must be either an LP-, an LH-, or an LF-vertex. In any of these three cases this implies the existence of at least one vertex in the set $(\Gamma(v) \cup \{v\})$ that can be happy, leading to $H(G)^* \geq 1$.

When $H(G)^* = 0$ there is clearly nothing to optimise and the corresponding MHV problem is trivial to solve. When generating problem instances we can also estimate the probability that $H(G)^* = 0$ (under certain assumptions) as we now explain.

First, when generating a problem instance assume that the $|V'|$ precoloured vertices are chosen at random and then assigned to any random colour $1, \ldots, k$. This gives approximately $\mu = |V'|/k$ precoloured vertices per colour class. Now, in order for a precoloured vertex v to be a U-vertex (and therefore destined to be unhappy in all solutions), it is necessary that at least one of its neighbours is precoloured differently. The complement of this event is that all neighbours of v are either free or coloured the same as v. Hence,

$$\begin{aligned} P(v \text{ is a U-vertex}) &= 1 - P(\text{Each neighbour of } v \text{ is free or has } v\text{'s colour}) \\ &= 1 - \frac{\binom{|V-V'|+\mu-1}{\deg(v)}}{\binom{n-1}{\deg(v)}}. \end{aligned} \tag{1}$$

Similarly, for a free vertex v to be an LU-vertex (and thus also destined to be unhappy in all solutions) it must have at least one pair of neighbouring precoloured vertices $u_1, u_2 \in \Gamma(v)$ with different colours to one another. The complement of this event is that the number of distinct colours among neighbours of v is zero or one. Hence,

$$\begin{aligned} P(v \text{ is an LU-vertex}) =&1 - P(\text{Num. colours adjacent to } v \text{ is zero or one}) \\ =&1 - \frac{\binom{|V-V'|+\mu}{\deg(v)}}{\binom{n-1}{\deg(v)}}. \end{aligned} \tag{2}$$

If we now assume that the degrees of all vertices in G are equal at $\bar{d} = \frac{1}{n}\sum_{v \in V} \deg(v)$, this gives:

$$P(H(G)^* = 0) = \left(1 - \frac{\binom{|V-V'|+\mu-1}{\bar{d}}}{\binom{n-1}{\bar{d}}}\right)^{|V'|} \cdot \left(1 - \frac{\binom{|V-V'|+\mu}{\bar{d}}}{\binom{n-1}{\bar{d}}}\right)^{|V-V'|}. \tag{3}$$

Figure 2 illustrates the effects of this probability function for differing values of n and k. Areas to the right of the lines indicate combinations of parameters (average degree $\bar{d}$ and the proportion of precoloured vertices $(|V'|/n)$) that, with $\geq 95\%$ probability, give problem instances for which $H(G)^* = 0$. We see that these areas occupy most of the figures, with only very low values of one or both of the parameters leading to non-trivial problem instances for which $H(G)^* > 0$. We must bear in mind though that this model is based on the assumptions outlined above and has the potential to become less accurate when the selection of precoloured vertices is non-random or the variance across the vertex degrees is higher.

2.4 Precolouring Additional Vertices

For some MHV problem instances it is also possible to fix the colours of some free vertices from the outset, in effect adding additional precoloured vertices to the graph. For instance, if all neighbours of a free vertex v are precoloured with the same colour i, then it is obvious that v should also assume colour i (and therefore be happy) in any optimal solution.

The following method generalizes this property. Starting from a free vertex v, let S be the set of vertices that are reachable from v in the subgraph induced

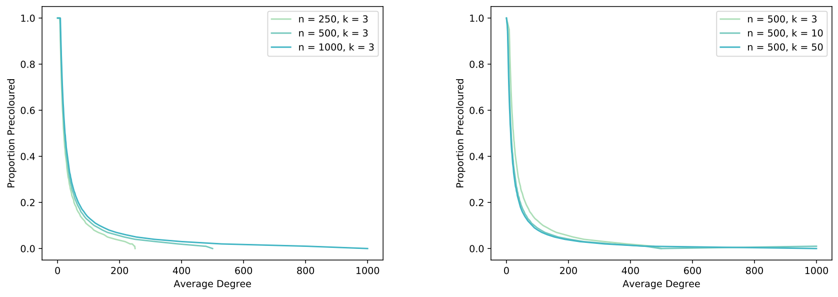


Fig. 2 Areas to the right of the curves indicate parameter combinations that, with $\geq 95\%$ confidence, lead to problem instances for which $H(G)^* = 0$.

by the free vertices of G. Now, using G consider the set of precoloured vertices that neighbour the vertices of $S \cup \{v\}$. If all of these precoloured vertices have the same colour i, then all vertices in $S \cup \{v\}$ should also be coloured with i, making them H-vertices (and therefore happy). Similarly, if none of the vertices in $S \cup \{v\}$ have a precoloured neighbour, then these vertices make up a single component in G that comprises free vertices only. In this case they can all be made into H-vertices by assigning them all to the same arbitrary colour $i \in \{1, \ldots, k\}$.

Further precoloured vertices can also be added by noting that if v is an LU-vertex and all of its neighbours are U-vertices, then v can be assigned to any arbitrary colour, also making it a U-vertex. This is because, whatever colour we choose for v, all vertices in $\Gamma(v) \cup \{v\}$ are still guaranteed to be unhappy.

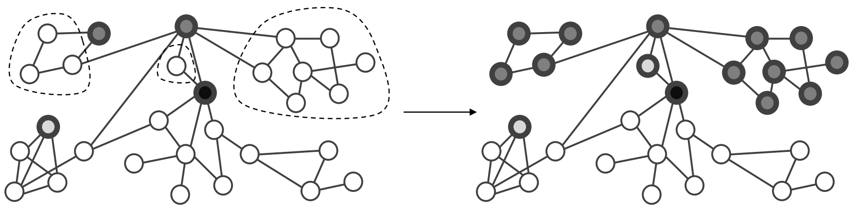


Fig. 3 Example of how additional precoloured vertices can be added to a problem instance. The graph on the left has three places where this can occur, shown by the dashed loops. The left and right loops indicate where further H-vertices can be added; the central loop shows where a U-vertex can be added. The resultant graph is shown on the right.

In the experiments reported in this paper the above conditions are checked for using a procedure called Add-Precol, which operates until all free vertices of G have been considered. In essence Add-Precol is a restricted version of BFS and therefore operates in $O(m)$ time. We use Add-Precol as a preprocessing step with all algorithms tested in this paper because fixing the colours of free vertices in this way will reduce the size of the solution space while not changing the value of $H(G)^*$. An example of the process is shown in Figure 3. Its effects of this procedure are investigated further in Section 4.1.

3 Tabu Search

In this section we describe our tabu search method for the MHV problem. While many aspects of this algorithm are similar to other applications of tabu search with graph partitioning problems (Blöchliger and Zufferey, 2008; Lewis, 2015), the main novelty comes from our high-speed method of calculating the costs of neighbouring solutions, as described in Section 3.1 below. The source code for this algorithm can be found at (Lewis, 2020).

The overall tabu process starts with an initial feasible solution and seeks to make improvements through a series of modifications. To discourage the algorithm from entering previously visited areas of the solution space, a tabu list is also maintained during the run, helping to maintain a beneficial trade-off between intensification and diversification.

In our application, additional precoloured vertices are first added to the graph (if possible) using the Add-Precol procedure. An initial solution is then formed by taking the best solution produced through applications of the Greedy-MHV and Growth-MHV heuristics (Section 2.2). The tabu search algorithm then takes this initial solution and iterates until either a time limit is exceeded, or a solution is achieved whose number of happy vertices equals the upper bound of $\bar{H}(G)$ (Section 2.1).

Candidate solutions in our approach are represented as a full partition of the vertices into k colour classes, $\mathcal{S} = (S_1, \ldots, S_k)$, where S_i contains all vertices assigned to colour i (i.e. vertices v for which $c(v) = i$). In each iteration of the algorithm a neighbouring solution is produced from the current solution $\mathcal{S}$ by taking a free, unhappy vertex v for which $\deg(v) \geq 1$, removing it from its current colour class S_i and inserting it into a new colour class S_j. As is typical for tabu search, this neighbouring solution is selected by scanning the entire neighbourhood of $\mathcal{S}$ and choosing the non-tabu solution that features the largest increase (or, failing that, the smallest decrease) in the number of happy vertices. Any ties in this criterion are broken randomly. A tabu solution can also be selected if its number of happy vertices is seen to exceed the number in the best solution seen so far in the run (this is sometimes known as an aspiration criterion). In addition, if all neighbouring solutions are seen to be tabu, then one of these is selected at random.

Here, a matrix $T_{n \times k}$ is used to represent the tabu list. If a move at iteration l of the algorithm is performed that transfers vertex v from S_i to S_j, then element T_{vi} is set to $l + t$. This is interpreted to mean that, for the next t iterations, all solutions involving the assignment of v to colour i are considered tabu and are not permitted unless the above aspiration criterion is met. To determine the value of t, we use previous studies on graph partitioning problems as a guide (Blöchliger and Zufferey, 2008; Lewis and Carroll, 2016). These suggest that t should be a random variable whose value is determined based on the current solution's quality. In essence, t should be given a high value when solution quality is low, which results in the algorithm being compelled to move into new areas of the solution space. Conversely, when solution quality is high t should be reduced, thereby allowing the algorithm to focus more on the immediate vicinity. In our case, t is determined as $r + \tau(\bar{H}(G) - f(\mathcal{S}))$. Here, r is an integer randomly chosen from the set $\{1, \ldots, 9\}$ in each iteration of the algorithm (adding an element of randomness to the function), and $\tau \in \mathbb{R}^+$ is a user-defined parameter. As before, $\bar{H}(G)$ is the upper bound on the number of happy vertices in G, whereas $f(\mathcal{S})$ is the number of happy vertices in the current solution $\mathcal{S}$.

3.1 Speeding Up the Algorithm

As might be expected, the most time-consuming part of the tabu search process is the evaluation of all neighbouring solutions at each iteration. However, in our application considerable efficiencies can be achieved through the use of an additional matrix $\mathbf{C}_{n\times k}$ where, given the current solution $\mathcal{S} = \{S_1, \ldots, S_k\}$, element C_{vj} denotes the change in the number of happy vertices that would result if vertex v were to be moved to colour j. When an initial solution is generated, values will need to be generated for all rows of $\mathbf{C}$ corresponding to free vertices; however, in each subsequent iteration of the algorithm the act of moving a vertex v to S_j will result in a new solution $\mathcal{S}'$ whose number of happy vertices $f(\mathcal{S}')$ can be calculated as simply $f(\mathcal{S}) + C_{vj}$. Since $f(\mathcal{S})$ will already be known, this means that the cost of all neighbouring solutions can be determined by simply scanning each row of $\mathbf{C}$ corresponding to free unhappy vertices.

Once a move been performed by moving v to colour j, relevant entries in $\mathbf{C}$ will need to be updated to reflect these changes. However, this only needs to be done for rows corresponding to free vertices within a distance of two from v (that is, v, neighbours of v and neighbours of neighbours of v). All other rows of the matrix will not be affected. Our method of calculating a new value for an element C_{vi} in $\mathbf{C}$ is described in the CALCULATE-C-ENTRY procedure shown in Figure 4. Lines 4 and 5 of this procedure deal with the case where v is currently happy; hence, changing its colour will make it unhappy. Similarly, Lines 6 and 7 consider the case where all of v's neighbours have colour i, meaning that changing v's colour to i will result in it becoming happy. Lines 8 to 12 carry out similar operations for all vertices that are a distance of two from v.

4 Experiments and Results

In this section we describe the experiments conducted for this research. Section 4.1 illustrates the general behaviour of the ADD-PRECOL procedure; Sec-

```
        CALCULATE-C-ENTRY $(v, i)$
(1)     result ← 0
(2)     if $c(v) = i$ or $\deg(v) = 0$ then
(3)         return result
(4)     if $v$ is happy then
(5)         result ← result−1
(6)     else if $c(u) = i, \forall u \in \Gamma(v)$ then
(7)         result ← result+1
(8)     forall $u \in \Gamma(v)$ do
(9)         if $u$ is happy then
(10)            result ← result−1
(11)        else if $c(u) = i$ and $c(w) = i, \forall w \in (\Gamma(u) - \{v\})$ then
(12)            result ← result+1
(13)    return result
```

Fig. 4 Procedure for calculating a new value for element C_{vi} in the matrix $\mathbf{C}$. As usual, $\Gamma(v)$ denotes the set of all vertices adjacent to a vertex v and $c(v)$ denotes the current colour of v.

tions 4.2 and 4.3 then provide a performance analysis of our algorithm on two contrasting graph topologies: random graphs and scale free graphs.

The problem instances used in our experiments were produced using the generator of Lewis et al. (2019). Random graphs are generated in an unbiased manner. Starting with two vertices, an edge is added between them with probability p. Further vertices are then added one by one and, in each case, edges are added between this vertex and all existing vertices with probability p. The resulting graph has a binomial-shaped degree distribution with mean $(n-1)p$ and variance $(n-1)p(1-p)$ and features an expected number of edges of $\binom{n}{2}p$.

In contrast, scale-free graphs are biased in the sense that when a new vertex is added to a graph it is more likely to be joined to vertices of high degrees. Consequently, their degree distributions follow a power law, and the ensuing graphs typically comprise a small number of high-degree vertices and a large number of very low-degree vertices. Compared to random graphs, scale-free graphs therefore show features of “preferential attachment” that are often encountered in real-world settings (Barabási and Põsfai, 2016). In our tests, scale-free graphs were constructed using the Barabási-Albert model, which operates as follows: To begin, a complete graph G with $q \in \{1, \ldots, n\}$ vertices and $\binom{q}{2}$ edges is constructed. In each iteration a new vertex v is then added to G and is connected to q existing vertices in G, with an edge between v and $u \in G$ added according to the probability:

$$P(u, v) = \frac{\deg(u)}{\sum_{w \in (V - \Gamma(v))} \deg(w)} \tag{4}$$

where $V - \Gamma(v)$ is the set of vertices that are not adjacent to v. The graph construction continues until n vertices are present in the graph, giving a total of $\binom{q}{2} + q(n-q)$ edges.

For both random and scale-free topologies, a subset of vertices is then randomly chosen to be precoloured. Colour assignments for these vertices are also made randomly, but is such a way so that each of the k colours is used by at least one vertex in the graph.

To help assess the performance of our tabu search algorithm, results for two additional methods are also included in our analysis. The first of these uses the best-performing IP formulation as proposed by Lewis et al. (2019), which is defined as follows.

Let $x_j \in \{1, \ldots, k\}$ be integer variables and y_j be binary variables for all $j \in \{1, \ldots, n\}$. The variable x_j specifies the colour of vertex v_j, and $y_j = 1$ if and only if v_j is unhappy. The problem is to now:

$$\text{maximise} \quad n - \sum_{j=1}^{n} y_j \tag{5}$$

$$\text{subject to:} \quad x_j = c(v_j) \qquad \forall v_j \in V' \tag{6}$$

$$y_j \geq \frac{|x_j - x_i|}{n} \qquad \forall v_i \in \Gamma(v_j), \forall v_j \in V. \tag{7}$$

In this formulation, precolourings are specified by Constraint (6). Constraint (7) then sets $y_j = 1$ if the colours of the vertices adjacent to the vertex v_j are not the same colour as v_j (specified in x_j). The objective maximises the number of happy vertices.

The second method used in our analysis is based on an application of the construct, merge, solve and adapt (CMSA) methodology for the MHV problem, also proposed by Lewis et al. (2019). The basic idea of CMSA is to maintain a set of problem components $\mathcal{C}'$ that are currently permitted to be used in the construction of a candidate solution. At each iteration, an exact method is then applied that attempts to find the optimal solution for the problem in which only components currently belonging to $\mathcal{C}'$ are used. This information is then used to help modify the contents of $\mathcal{C}'$ before continuing with the next iteration. For the MHV problem, $\mathcal{C}'$ is defined as a subset of all possible free-vertex/colour pairings $\mathcal{C}$, where

$$\mathcal{C} = \{(u, i) : u \text{ is free and } i \in \{1, \ldots, k\}\}. \tag{8}$$

At each iteration, the above IP formulation is then used to solve a restricted version of the problem in which only components in $\mathcal{C}'$ can be used. The idea, therefore, is that when $\mathcal{C}'$ is much smaller than $\mathcal{C}$, applications of the exact solver will require less computational effort. In addition, if the correct contents of $\mathcal{C}'$ can be identified during a run, then high-quality solutions to the overall problem will also be established.

In all cases reported here, IP optimisation was carried out using Gurobi 8.0.0. Each run was permitted a single thread, allowing a fairer comparison with our tabu search method. In addition, before each application of the IP solver, the ADD-PRECOLS procedure was executed, and a seed solution equivalent to the initial solution of the tabu search algorithm was provided to enhance performance. These augmentations therefore extend the methods described by Lewis et al. (2019).

4.1 Effects of Adding Precoloured Vertices.

In this section we consider the effects that the ADD-PRECOL procedure has on the resultant number of precoloured vertices in a graph. Figures 5 and 6 show results for both random and scale-free graphs, respectively, for two different values of k and various different graph densities (expressed as the average degree $\frac{2m}{n}$). The lines in the figures represent instances that were initially generated with the proportion of precoloured vertices stated in the legends. The heights of the lines then indicate the number of precoloured vertices present in the graph after executing ADD-PRECOL.[1]

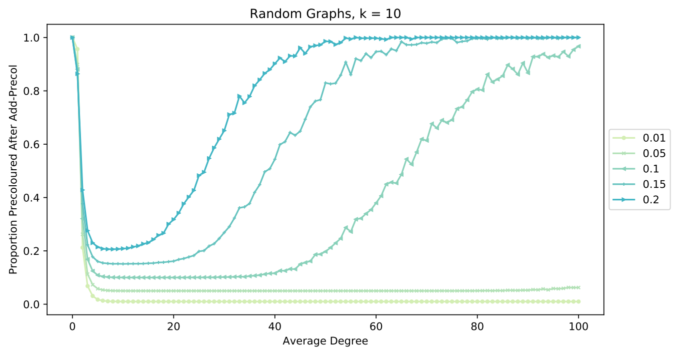
We see from Figures 5 and 6 that ADD-PRECOL has a considerable effect with sparse graphs. This is because the lack of edges means that many of the free vertices can be converted into precoloured H-vertices. (Of course, if the graph has no edges, then all vertices can be precoloured in this way, leading to $H(G)^* = n$.) On the other hand, with dense graphs the ADD-PRECOL is also able to add many additional precoloured vertices. This is because the larger number of edges will increase the number of U- and LU-vertices, meaning that further precoloured U-vertices will often be added to the graph. For the same reason, these patterns are also more pronounced when the proportion of precoloured vertices in the initial problem instance is higher, as shown in the figure.

Figures 5 and 6 also indicates that the ADD-PRECOL procedure has the smallest effect when a graph has an average degree of approximately four to ten and/or when the proportion of precoloured vertices is low. Interestingly, these instances correspond closely to those that were found to be the most difficult to solve in the computational study of Lewis et al. (2019). Also note that these patterns are consistent across the different k-values and topologies, as shown.

4.2 Random Graphs

In this section we compare the performance of the tabu search, IP, and CMSA methods on random graphs. Previous empirical research (Lewis et al., 2019)

[1] Note that when the desired number of precoloured vertices $|V'|$ is less than the number of available colours k, instances cannot be generated. As a result, such parameter combinations are not present in our analyses.



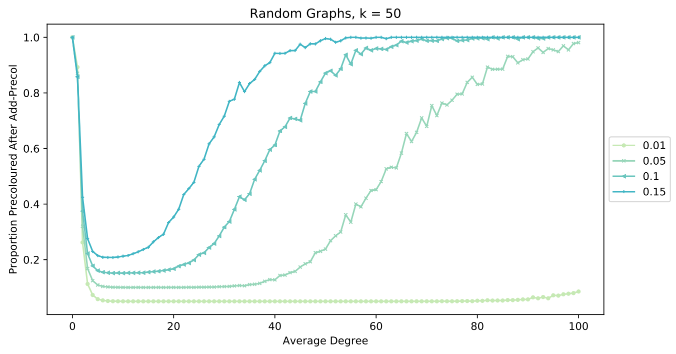
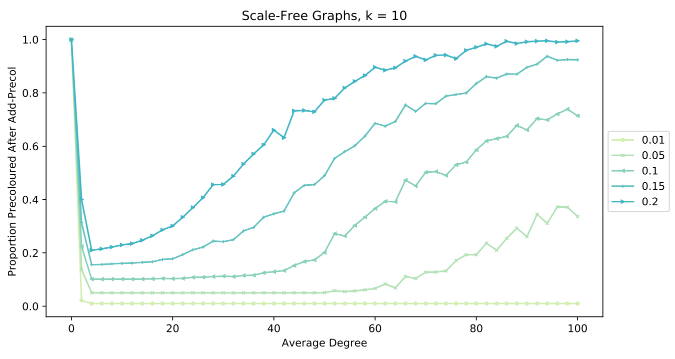


Fig. 5 Effects of the ADD-PRECOL procedure with random graphs with $k = 10$ (top) and $k = 50$ (bottom). In all cases $n = 1000$, and points are calculated as the mean across twenty runs at each average degree value.

together with the findings of Sections 2.3 and 4.1 strongly suggest that non-trivial problem instances only exist in small pockets of the instance space. Here, we focus our attention on random graphs with average degrees of five, numbers of vertices ranging from $n = 250$ to $10,000$ (giving densities of $\frac{5}{n-1}$) and k-values of 10 and 50. Twenty graphs were generated in each case with 10% of vertices being precoloured. For all methods, a maximum run time of ten minutes was imposed.[2]

Table 1 presents information on the lower and upper bounds that were obtained with these random graphs. These figures are stated as the percentage of vertices that are happy averaged across twenty problem instances of each size. The Upper Bounds column compares $\bar{H}(G)$ with the upper bound obtained by the IP solver (UB) at the time limit; the Lower Bounds column compares

[2] Methods were implemented in C++ and compiled with GCC-5.4.0. Our code is available at `http://www.rhydlewis.eu/resources/happytabu.zip`. The experiments were conducted on Monash University's Campus Cluster, where each machine in the cluster consists of 24 cores and 256 GB RAM. Each physical core has two hyper-threaded cores with Intel Xeon E5-2680 v3 2.5GHz, 30M Cache, 9.60GT/s QPI, Turbo, HT, 12C/24T (120W). For tabu search a setting of $\tau = 2$ was used.



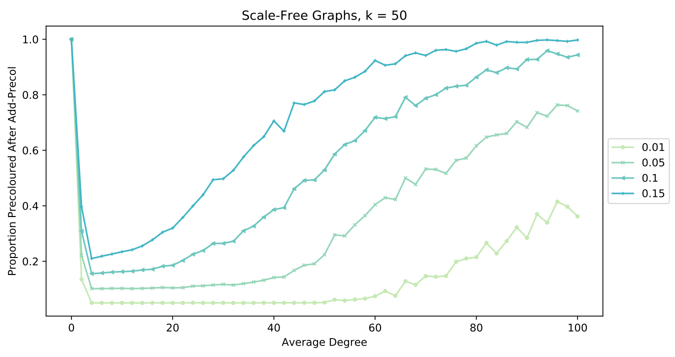


Fig. 6 Effects of the Add-Precol procedure with scale-free graphs with $k = 10$ (top) and $k = 50$ (bottom). In all cases $n = 1000$, and points are calculated as the mean across twenty runs at each average degree value.

the quality of the best solutions returned by the IP, CMSA and tabu search algorithms.

Comparing upper bounds, we see that the IP solver's upper bounds are superior for problem instances with up to $5,000$ vertices for both $k = 10$ and $k = 50$. For larger instances, $\bar{H}(G)$ is superior. This is because, for very large problems, substantial computation is required by the IP solver to improve the bounds, which does not seem to be possible within the time limit. On the other hand, computing $\bar{H}(G)$ is computationally inexpensive, though the results show that they are still some way from the optimal values.

Regarding lower bounds, for the smallest instances ($n = 250$, $k = 10$), the IP solver is the most effective—indeed, here all instances have been solved to optimality within the time limit. It is also clear that for the largest of problem instances ($n \geq 5,000$) that the tabu search is the most effective method.

Table 2 shows the time at which each of the methods achieved their best observed solutions. We see that tabu search finds solutions much more quickly than CMSA or the IP model. For tabu search, the time required to find the best solution increases with increasing values of n, though this is usually well

Table 1 Comparison of lower and upper bounds for random graphs of various sizes. Figures are presented as the mean and standard deviation of the percentages of the number of happy vertices across twenty problem instances for each problem size. Statistically significant results are marked with an asterisk, obtained according to a Wilcoxon signed rank test at the 0.05% level.

		Upper Bounds (Mean ± SD)		Lower Bounds (Mean ± SD)		
	n	$\bar{H}(G)$	UB	IP	CMSA	TS
$k = 10$	250	73.820 ± 2.158	60.760 ± 2.400	**60.760 ± 2.400**	60.740 ± 2.408	60.740 ± 2.408
	500	74.690 ± 1.437	64.780 ± 2.105	60.890 ± 2.507	60.890 ± 2.507	60.890 ± 2.507
	750	74.920 ± 0.894	65.973 ± 1.574	60.280 ± 1.293	60.287 ± 1.291	60.287 ± 1.291
	1000	74.940 ± 1.061	66.060 ± 1.178	59.855 ± 1.263	**59.860 ± 1.258**	59.850 ± 1.259
	2000	74.923 ± 0.912	67.160 ± 1.486	59.675 ± 1.145	59.720 ± 1.105	59.720 ± 1.105
	3000	74.987 ± 0.703	70.820 ± 1.343	59.223 ± 0.835	59.552 ± 0.679	**59.555 ± 0.682**
	4000	75.035 ± 0.593	70.749 ± 1.043	59.041 ± 0.809	**59.295 ± 0.757***	59.294 ± 0.758
	5000	74.877 ± 0.454	70.445 ± 0.977	58.698 ± 0.652	58.936 ± 0.606	**58.968 ± 0.602***
	7500	74.843 ± 0.442	79.846 ± 10.659	58.515 ± 0.595	58.853 ± 0.578	**58.884 ± 0.568***
	10000	74.075 ± 0.346	97.897 ± 4.945	57.444 ± 0.637	57.921 ± 0.537	**57.943 ± 0.544***
$k = 50$	500	74.190 ± 1.403	64.660 ± 2.537	56.250 ± 2.311	56.250 ± 2.311	56.250 ± 2.311
	750	74.440 ± 1.008	65.587 ± 1.749	56.813 ± 1.669	56.827 ± 1.653	56.827 ± 1.653
	1000	74.540 ± 1.076	65.500 ± 1.389	56.560 ± 1.111	56.585 ± 1.127	56.585 ± 1.127
	2000	74.433 ± 0.952	66.583 ± 1.577	56.298 ± 1.271	56.413 ± 1.163	56.413 ± 1.163
	3000	74.500 ± 0.749	69.280 ± 1.191	56.182 ± 0.787	56.392 ± 0.768	**56.393 ± 0.767**
	4000	74.566 ± 0.589	69.155 ± 0.969	55.926 ± 0.736	56.210 ± 0.702	**56.253 ± 0.696***
	5000	74.397 ± 0.425	68.997 ± 0.905	55.808 ± 0.739	55.881 ± 0.677	**55.928 ± 0.669***
	7500	74.336 ± 0.462	78.005 ± 8.140	55.548 ± 0.567	55.918 ± 0.608	**55.959 ± 0.600***
	10000	73.572 ± 0.314	92.461 ± 4.987	54.638 ± 0.476	54.913 ± 0.451	**54.944 ± 0.460***

Table 2 Comparison of run-times (seconds) required to find the best solution achieved by each algorithm. All figures are means across twenty random problem instances.

	n	IP	CMSA	TS
$k = 10$	250	8.10 ± 8.14	1.74 ± 4.70	0.00 ± 0.00
	500	12.90 ± 30.62	68.08 ± 87.01	0.00 ± 0.00
	750	12.05 ± 22.27	84.02 ± 101.62	0.00 ± 0.00
	1000	92.80 ± 162.85	135.22 ± 148.13	0.00 ± 0.00
	2000	97.10 ± 199.72	193.11 ± 166.98	0.00 ± 0.00
	3000	40.25 ± 34.96	234.38 ± 131.83	0.00 ± 0.01
	4000	119.65 ± 107.69	272.77 ± 158.64	0.01 ± 0.01
	5000	115.00 ± 127.01	238.92 ± 160.16	0.02 ± 0.01
	7500	241.10 ± 185.52	259.21 ± 179.92	0.04 ± 0.03
	10000	201.95 ± 82.71	255.37 ± 160.48	0.06 ± 0.06
$k = 50$	500	51.35 ± 134.51	12.08 ± 24.63	0.00 ± 0.00
	750	160.45 ± 182.75	18.18 ± 34.28	0.00 ± 0.01
	1000	38.00 ± 41.41	27.29 ± 56.69	0.01 ± 0.02
	2000	115.25 ± 142.76	46.06 ± 86.65	0.03 ± 0.05
	3000	82.45 ± 110.95	102.39 ± 116.07	0.07 ± 0.10
	4000	107.25 ± 88.37	162.83 ± 159.90	3.16 ± 8.15
	5000	158.00 ± 91.79	199.30 ± 181.34	0.90 ± 1.52
	7500	146.05 ± 137.27	277.87 ± 211.92	4.15 ± 6.40
	10000	233.25 ± 104.22	313.03 ± 183.67	6.71 ± 20.1

within 10 seconds. Additionally, the times needed for $k = 10$ are lower than $k = 50$. We also see that CMSA usually requires more time than the IP model to find its best solutions though, once found, these are usually superior.

4.3 Scale-Free Graphs

We now consider a similar set of results for scale-free graphs. In these trials, instances were generated using a setting of $q = 3$, leading to graphs with an average vertex degree of six. All other settings were kept the same as the previous subsection.

Table 3 shows our results in the same format as Table 1. Regarding upper bounds, in contrast to random graphs we see that those of the IP solver are always superior to $\bar{H}(G)$, even for the very largest graphs. As noted, scale-free graphs of this nature feature much higher vertex degree variances than random graphs, which means that its constraints tend to be more concentrated in certain parts of the problem instance. This seems to allow the IP solver to improve its upper bound more effectively within the 10 minute time limit.

Table 3 Comparison of lower and upper bounds for scale-free graphs of various sizes. Figures are presented as the mean and standard deviation of the percentages of the number of happy vertices across twenty problem instances for each problem size. Statistically significant results are marked with an asterisk, obtained according to a Wilcoxon signed rank test at the 0.05% level; IP = IP model, CMSA = construct, merge, solve and adapt; TS = tabu search.

		Upper Bounds (Mean ± SD)		Lower Bounds (Mean ± SD)		
	n	$\bar{H}(G)$	UB	IP	CMSA	TS
$k = 10$	250	76.040 ± 3.774	63.840 ± 5.395	63.840 ± 5.395	63.840 ± 5.395	63.840 ± 5.395
	500	74.430 ± 2.608	60.550 ± 3.417	59.610 ± 3.054	59.610 ± 3.054	59.610 ± 3.054
	750	75.713 ± 2.756	63.460 ± 4.355	60.947 ± 4.121	60.967 ± 4.141	60.967 ± 4.141
	1000	75.550 ± 2.618	63.305 ± 4.325	60.790 ± 4.026	60.790 ± 4.026	60.790 ± 4.026
	2000	75.978 ± 2.093	63.785 ± 3.469	60.020 ± 2.794	60.095 ± 2.799	60.095 ± 2.799
	3000	75.878 ± 0.960	64.787 ± 1.548	58.392 ± 1.353	**58.853 ± 1.469**	58.852 ± 1.469
	4000	76.911 ± 1.305	69.056 ± 2.512	59.749 ± 1.395	**59.946 ± 1.444**	59.944 ± 1.446
	5000	76.940 ± 1.711	68.665 ± 3.223	58.941 ± 2.357	**59.397 ± 2.519**	59.394 ± 2.518
	7500	77.343 ± 0.960	73.619 ± 2.060	59.501 ± 1.577	60.041 ± 1.306	**60.043 ± 1.306***
	10000	79.067 ± 0.897	73.504 ± 3.033	60.096 ± 1.885	60.775 ± 1.606	**60.776 ± 1.607***
$k = 50$	500	73.320 ± 3.007	59.030 ± 4.903	55.820 ± 4.246	55.830 ± 4.246	55.830 ± 4.246
	750	74.953 ± 3.026	62.953 ± 4.936	57.587 ± 3.803	57.593 ± 3.795	57.593 ± 3.795
	1000	74.750 ± 2.925	62.720 ± 5.042	57.245 ± 4.083	57.245 ± 4.083	57.245 ± 4.083
	2000	75.245 ± 2.280	63.073 ± 3.808	56.933 ± 2.797	56.955 ± 2.741	56.955 ± 2.741
	3000	75.238 ± 0.900	64.522 ± 1.934	55.862 ± 1.427	55.988 ± 1.391	55.988 ± 1.391
	4000	76.320 ± 1.359	68.288 ± 2.639	56.868 ± 1.629	57.121 ± 1.591	57.121 ± 1.591
	5000	76.270 ± 1.819	67.675 ± 3.680	56.209 ± 2.903	56.716 ± 2.503	**56.719 ± 2.503***
	7500	76.692 ± 1.019	71.613 ± 1.876	56.130 ± 1.686	57.195 ± 1.309	**57.199 ± 1.309**
	10000	78.449 ± 0.924	72.267 ± 2.412	57.322 ± 1.754	58.341 ± 1.496	**58.342 ± 1.496**

Regarding lower bounds, similar patterns to the results for the random graphs are revealed. For smaller instances (up to and including 1,000 vertices), all three methods perform equally well. However, for 2,000 vertices and above, the CMSA and tabu search methods clearly outperform the IP solver. Again, this is because the run-time limit is simply too short for the IP solver to make substantial improvements to the seed solutions. For $k = 10$, CMSA seems to have a slight advantage over tabu search for instances with 4,000 and 5,000 vertices, but for larger instances the opposite is true with tabu search producing significantly better results. For $k = 50$, CMSA never outperforms tabu search on any instance, whereas tabu search seems to have an advantage on instances with 5,000 or more vertices, though this was only seen to be significant for 5000 vertices in our trials). This effect is due to our imposed run-time limits as CMSA seems to need more computational resources to improve solutions, particularly for large instances.

Finally, also note the larger variances of the lower bounds in Table 3 compared to those of random graphs. This is due to the distribution of vertex degrees in scale-free graphs. In some instances, only low-degree vertices will happen to be precoloured by the instance generator, resulting in problems in which a higher proportion of vertices can be happy. On the other hand, in some cases vertices of high degrees (hub nodes) will be precoloured instead, resulting in instances in which substantially fewer happy vertices can be achieved. This contrasts with random graphs, where degrees amongst vertices tend to vary less, giving more consistent results.

Table 4 Comparison of run-times (seconds) required to find the best solution achieved by each algorithm. All figures are means across twenty scale-free problem instances.

	n	IP	CMSA	TS
$k = 10$	250	0.40 ± 1.57	1.09 ± 3.44	0.00 ± 0.00
	500	9.45 ± 22.45	18.02 ± 38.18	0.00 ± 0.00
	750	39.90 ± 95.30	85.59 ± 158.43	0.00 ± 0.00
	1000	11.05 ± 22.63	127.12 ± 112.45	0.00 ± 0.00
	2000	64.45 ± 116.55	208.47 ± 143.58	0.00 ± 0.01
	3000	34.70 ± 70.67	221.62 ± 167.87	0.01 ± 0.02
	4000	77.05 ± 76.21	328.00 ± 176.15	0.01 ± 0.05
	5000	104.70 ± 121.97	246.43 ± 154.40	0.02 ± 0.06
	7500	222.60 ± 131.66	356.08 ± 158.83	0.07 ± 0.13
	10000	171.60 ± 178.11	332.65 ± 181.49	0.10 ± 0.27
$k = 50$	500	14.95 ± 23.95	15.52 ± 27.44	0.00 ± 0.00
	750	42.70 ± 94.25	6.27 ± 18.75	0.00 ± 0.00
	1000	36.65 ± 91.01	6.36 ± 18.53	0.34 ± 1.52
	2000	98.60 ± 120.17	52.29 ± 85.57	0.04 ± 0.16
	3000	73.05 ± 91.01	133.11 ± 149.63	5.61 ± 15.62
	4000	83.45 ± 70.03	118.42 ± 96.65	11.73 ± 29.06
	5000	135.85 ± 70.49	144.44 ± 160.20	13.46 ± 24.58
	7500	218.20 ± 200.77	310.70 ± 147.65	30.54 ± 83.74
	10000	216.85 ± 178.82	293.85 ± 175.81	29.21 ± 116.62

Table 4 shows when all three methods found their best solutions on scale-free graphs. Similar to random graphs, we see that tabu search is substantially faster than CMSA or the IP model. Furthermore, like random graphs, the time required by tabu search gradually increases with increasing values of n and the time requirements for $k = 10$ are lower than $k = 50$. Finally, CMSA usually takes more time to find its best solution compared to the IP model, however, the solutions it finds are almost always superior to that of the IP model.

5 Conclusions

This paper has investigated the use of a tabu search algorithm for the maximum happy vertices problem. In comparison to an IP solver and a CMSA method, we have found that it is very efficient at finding good quality solutions (lower bounds), particularly with larger problem instances. Moreover, these solutions are found significantly more quickly than previous known methods for this problem. We have also proposed a method for computing upper bounds efficiently, which is more effective than our IP optimiser on very large problem instances. Furthermore, we show that a method to impose additional precoloured vertices can lead to a more efficient overall tabu search algorithm. Additionally, we are able to determine the "hardness" of problem instances via the use of probabilistic arguments.

While tabu search seems quite effective at finding good heuristic solutions (especially for large instances), combining it with other approaches might further improve the overall performance of the ensuing methods. One future possibility would be to develop a matheuristic approach that combines both the tabu search and IP. Since tabu search is extremely quick, it can be used very efficiently with IP methods such as Lagrangian relaxation or column generation to produce fast local improvements. Another option is to combine tabu search with CMSA, where the populations of solutions for the CMSA component can be found via tabu search strategies. These hybrids might provide improvements over the individual methods by utilising their relative advantages.

We have presented a first step to obtaining upper bounds. This approach is quite effective for small problem instances (outperforming IP for scale-free graphs) and we are currently exploring algorithms and strategies (including a matheuristic) that can improve on these bounds.

Compliance with Ethical Standards

The authors guarantee that there are no conflicts of interest and that this research involves no human participants nor animals.

References

Agrawal A (2018a) On the Parameterized Complexity of Happy Vertex Coloring. In: Brankovic L, Ryan J, Smyth WF (eds) Combinatorial Algorithms, Springer International Publishing, Cham, pp 103–115

Agrawal A (2018b) On the parameterized complexity of happy vertex coloring. In: Brankovic L, Ryan J, Smyth W (eds) Combinatorial Algorithms. IWOCA 2017, Lecture Notes in Computer Science, vol 10765, Springer, Cham., pp 103–115

Aravind N, Kalyanasundaram S, Kare A (2016) Linear time algorithms for happy vertex coloring problems for trees. In: Mäkinen V, Puglisi S, Salmela L (eds) Combinatorial Algorithms: IWOCA 2016, Lecture Notes in Computer Science, vol 9843, Springer Cham., pp 281–292

Barabási AL, Põsfai M (2016) Network Science. Cambridge University Press

Blöchliger I, Zufferey N (2008) A Graph Coloring Heuristic using Partial Solutions and a Reactive Tabu Scheme. Computers & Operations Research 35(3):960 – 975, part Special Issue: New Trends in Locational Analysis

Blum C, Roli A (2003) Metaheuristics in Combinatorial Optimization: Overview and Conceptual Comparison. ACM Computing Surveys 35:268–308

Carter MW, Laporte G, Lee SY (1996) Examination Timetabling: Algorithmic Strategies and Applications. The Journal of the Operational Research Society 47(3):373–383

Di Gaspero L, Schaerf A (2001) Tabu Search Techniques for Examination Timetabling. In: Burke E, Erben W (eds) Practice and Theory of Automated Timetabling III, Springer Berlin Heidelberg, Berlin, Heidelberg, pp 104–117

Everitt B, Landau S, Leese M, Stahl D (2011) Cluster Analysis. John Wiley and Sons

Glover F, Laguna M (1997) Tabu Search. Kluwer Academic Publishers, Norwell, MA, USA

Glover F, Laguna M (1999) Tabu Search, Springer US, Boston, MA, pp 2093–2229

Lewis R (2015) A Guide to Graph Colouring: Algorithms and Applications, 1st edn. Springer Publishing Company, Incorporated

Lewis R (2020) Tabu search source code. `http://www.rhydlewis.eu/resources/happytabu.zip`, accessed: 2020-01-24

Lewis R, Carroll F (2016) Creating seating plans: a practical application. Journal of the Operational Research Society 67(11):1353–1362

Lewis R, Thompson J (2015) Analysing the Effects of Solution Space Connectivity with an Effective Metaheuristic for the Course Timetabling Problem. European Journal of Operational Research 240(3):637 – 648

Lewis R, Thiruvady D, Morgan K (2019) Finding happiness: An analysis of the maximum happy vertices problem. Computers & Operations Research 103:265–276

Li A, Zhang P (2015) Algorithmic Aspects of Homophyly of Networks. Theoretical Computer Science 593:117 – 131

Mabrouk BB, Hasni H, Mahjoub Z (2009) On a Parallel Genetic–tabu Search Based Algorithm for Solving the Graph Colouring Problem. European Journal of Operational Research 197(3):1192 – 1201

McCollum B, Schaerf A, Paechter B, McMullan P, Lewis R, Parkes AJ, Gaspero L, Qu R, Burke EK (2010) Setting the Research Agenda in Automated Timetabling: The Second International Timetabling Competition. INFORMS J on Computing 22(1):120–130

Zhang P, Xu Y, Jiang T, Li A, Lin G, Miyano E (2018) Improved Approximation Algorithms for the Maximum Happy Vertices and Edges Problems.

Algorithmica 80(5):1412–1438

Zufferey N, Amstutz P, Giaccari P (2008) Graph colouring approaches for a satellite range scheduling problem. Journal of Scheduling 11(4):263–277