The Maximum Happy Induced Subgraph Problem: Bounds and Algorithms

Lewis, R.[1], Thiruvady, D.[2], and Morgan, K.[2]

[1]School of Mathematics, Cardiff University, Cardiff, Wales.
[2]School of Information Technology, Deakin University, Geelong VIC, Australia.,
LewisR9@cf.ac.uk, dhananjay.thiruvady@deakin.edu.au,
kerri.morgan@deakin.edu.au

September 4, 2020

Abstract

In this paper we consider a combinatorial optimisation problem that takes as input a graph in which some of the vertices have been preassigned to colours. The aim is to then identify the largest induced subgraph in which all remaining vertices are able to assume the same colour as all of their neighbours. This problem shares similarities with the graph colouring problem, vertex cut problems, and the maximum happy vertices problem. It is NP-hard in general. In this paper we derive a number of upper and lower bounds and also show how certain problem instances can be broken up into smaller subproblems. We also propose one exact and two heuristic algorithms for this problem and use these to investigate the factors that make some problem instances more difficult to solve than others.

Keywords: Graph Colouring; Combinatorial Optimisation; Vertex Cut Sets; Tabu Search.

1 Introduction

Given a simple graph $G = (V, E)$, the well-known graph colouring problem seeks an assignment of colours to vertices such that pairs of adjacent vertices are allocated to different colours, while the number of colours being used across the graph is minimised. In operational research and related fields, graph colouring is often used to model situations where a set of conflicting entities such as tasks, events, or people need to be efficiently allocated to a limited set of resources. Examples include school and university timetabling, sports scheduling, frequency assignment, compiler register allocation, and the construction of seating plans [20].

In the past five years or so, interest has also been growing in a different type of graph colouring problem in which related vertices are required to be assigned to the *same* colour as one another. This can have uses in areas such as social network analysis, where a suitable assignment of colours can help to identify communities of closely related individuals [11], or in cluster analysis, where a set of objects (vertices) need to be partitioned such that similar objects are assigned to same group (colour) as one another [13].

In a 2015 paper, Li and Zhang [23] introduced a related concept known as vertex "happiness", defined as follows:

Definition 1. *Let $G = (V, E)$ be a simple graph, and let $c : V \to \{1, \ldots, k\}$ be a colouring of all vertices in G. A vertex $v \in V$ is* happy *if $c(v) = c(u)$ for all $u \in \Gamma(v)$; else it is* unhappy*.*

Here, $\Gamma(v)$ denotes the set of neighbours of a vertex $v \in V$; hence, in a graph in which all vertices have been coloured, a vertex is happy if and only if it is assigned to the same colour as all of its adjacent vertices.

In their research, Li and Zhang consider graphs in which a subset of the vertices have been precoloured. They then introduce the so-called maximum happy vertices (MHV) problem, which involves allocating colours to the remaining vertices such that the number of happy vertices in the graph is maximised. A practical application of this problem might occur when we have a set of people, some of whom have been preassigned to groups (colours), and we want to assign the remaining people to these groups such that the number of happy people is maximised. This could occur when choosing groups for a team building exercise or when assigning guests to shared bedrooms at a hotel. Li and Zhang have shown that the MHV problem is NP-hard in general, although it is polynomially solvable for graphs using fewer than three colours [23]. It is now known that the problem is also polynomially solvable for acyclic graphs, but that it remains NP-hard for bipartite

graphs and split graphs [4, 5]. Lewis et al. [22] have also developed a number of upper and lower bounds for this problem, together with methods for breaking up problems into smaller sub-problems.

In this paper we take this research in a new direction by investigating a type of problem that is related to the MHV problem. In this variant, which we call the maximum happy induced subgraph (MHIS) problem, vertices are forbidden from being unhappy but, instead, can remain uncoloured in the graph. As a result, a coloured vertex v is considered happy whenever all of its *coloured* neighbours have the same colour as v, as opposed to all of its neighbours. The objective is to then maximise the number of coloured (and therefore happy) vertices, which is equivalent to minimising the number of uncoloured vertices. In the next section we define this problem, discuss its relationships to other combinatorial optimisation problems and list some motivating examples. In Section 3 we then derive some upper and lower bounds and show how problem instances can sometimes be broken up into smaller parts that can be tackled independently. Section 4 then gives three algorithms for this problem, with results and analyses appearing in Section 5. Finally, Section 6 concludes the paper.

2 Problem Definition

Consider the following definition:

Definition 2. *Let $G = (V, E)$ be a simple graph with n vertices in which a subset of vertices $V' \subseteq V$ have been coloured by the function $c : V' \rightarrow \{1, \ldots, k\}$. A* happy component *in G is any connected component C that (a) contains no coloured vertices, or (b) whose coloured vertices are all assigned to the same colour. That is, C is happy if and only if $|\bigcup_{v \in (V(C) \cap V')} \{c(v)\}| \leq 1$, where $V(C)$ is the set of vertices belonging to component C.*

It is easy to see that in any happy component C, all of the uncoloured vertices can be assigned to colours so that they are happy. Specifically, if C contains no precoloured vertices, then all of its vertices can be assigned to the same arbitrary colour; otherwise, all of its uncoloured vertices should be coloured with the same colour as all of its coloured vertices. This brings us to our main problem definition:

Definition 3. *Let $G = (V, E)$ be a simple graph in which a subset of vertices $V' \subseteq V$ have been precoloured using the partial colouring function given in Definition 2. The* Maximum Happy Induced Subgraph *(MHIS) problem involves extending c to another partial colouring $\tilde{c} : X \rightarrow \{1, \ldots, k\}$ (where $V' \subseteq X \subseteq V$) such that: (a) in the graph induced by X all vertices are happy, and (b) the cardinality of X is maximised.*

Here, any vertex in G that is not precoloured in the initial problem instance (belonging to the set $V - V'$) is known as a "free vertex". A feasible candidate solution is then any assignment of colours to a subset of the free vertices such that constraint (a) in Definition 3 is satisfied. The optimal/maximum number of happy vertices for G (i.e. the maximum possible size of X) is denoted by $H(G)^*$.

Note that in any feasible solution to this problem, paths between differently coloured vertices will always contain at least one uncoloured vertex. This implies that in order for a problem instance to feature at least one feasible solution, we cannot have any edges whose endpoints are precoloured differently. This also means that the precoloured vertices in any feasible solution must be happy; hence, $|V'| \leq H(G)^* \leq n$. On the other hand, problem instances *can* feature free vertices whose neighbours are precoloured to different colours. Such vertices are known as "bridging" vertices. Bridging vertices can never be happy in a feasible MHIS solution, and must therefore remain uncoloured (see also Section 3.1).

An useful way of representing a candidate solution to this problem is as a partition of the vertices $\mathcal{S} = (S_1, \ldots, S_k) \cup U$, where each S_i is the set of vertices assigned to colour i, and U is the set of uncoloured free vertices. Here, each element $v \in S_i$ should either be a precoloured vertex for which $c(v) = i$, or a free vertex. A feasible solution is then any solution in which the subgraph induced by each S_i comprises one or more happy components. The task is therefore to identify a feasible solution $\mathcal{S}$ that maximises the objective function

$$f(\mathcal{S}) = \sum_{i=1}^{k} |S_i|, \tag{1}$$

where $f(\mathcal{S})$ gives the number of happy (coloured) vertices. This is equivalent to minimising the number of unhappy (uncoloured) vertices $|U|$.

A small example of the MHIS problem is shown in Figure 1(a). Part (b) of this figure shows how the removal of four free vertices in this graph results in the vertex subset X, whose induced subgraph comprises four happy components, giving a feasible solution. This solution can also be written as $\mathcal{S} = (S_1, \ldots, S_k) \cup U = (\{v_2, v_3, v_{14}\}, \{v_5, v_6, v_{13}\}, \{v_8, v_9, v_{10}, v_{11}, v_{12}, v_{16}, v_{17}\}) \cup \{v_1, v_4, v_7, v_{15}\}$, which has the objective function value $f(\mathcal{S}) = 13$. Finally, Part (c) shows how the colouring function $\tilde{c} : X \rightarrow \{1, \ldots, k\}$ is created by allocating appropriate colours to the free vertices in each happy component, as described above.

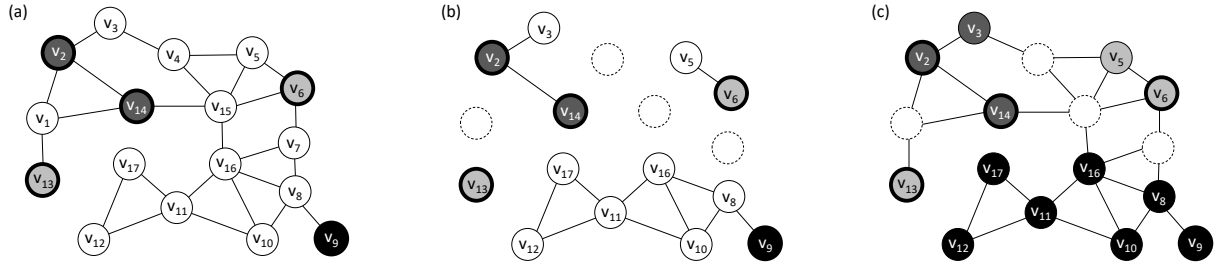

Figure 1: (a) An example MHIS problem instance with $n = 17$ vertices, $|V'| = 5$ precoloured vertices, $k = 3$ colours, and two bridging vertices (v_1 and v_{15}); (b) an example solution in which four vertices (dashed) have been removed, leaving four happy components; (c) the same solution as (b) with the remaining free vertices coloured, giving thirteen happy vertices.

2.1 Relationship to Cutting Problems

Figure 1(b) helps to demonstrate how the MHIS problem can be viewed as the task of identifying an appropriate vertex cut set in a graph. This allows parallels to be drawn with the task of finding minimum node multi-way cuts (MNMCs) in a graph. In the MNMC problem we are given a simple graph $G = (V, E)$ in which V contains, as a subset, an independent set of k vertices known as the "terminals". The problem then involves finding the smallest subset of non-terminal vertices that, when removed from G, leaves each terminal in a separate component. This problem is polynomially solvable when $k = 2$ because it is equivalent to the maximum flow (minimum cut) problem. Garg et al. [16] have also stated, without proof, that the problem is polynomially solvable for trees, though they also show that Vertex-Cover $\propto$ MNMC, proving it to be NP-hard in general. Marx [24] has also shown that when the maximum size of the vertex cut set is specified as part of the input, the MNMC problem is fixed parameter tractable; however, the associated algorithms are still prohibitively expensive in practice.

In fact, the MHIS and MNMC problems can be shown to be equivalent. To transform the MNMC problem into the MHIS problem, we simply need to allocate a different colour to each of the terminals, giving k differently-precoloured vertices; similarly, the opposite conversion can be made by considering each colour in turn, contracting all vertices of this colour into a single vertex, and then using these k contracted vertices as the terminals. Results and algorithms stated for one of these problems are therefore applicable to the other. Garg et al. [16] previously proposed an approximation algorithm for the MNMC problem based on an LP-relaxation. This features an approximation ratio of $2 - 2/k$. Integer programming-based methods are also described by Cornaz et al. [10].

2.2 Relationship to Graph k-Colouring

From a different perspective, we can also contrast the MHIS problem with the NP-hard partial graph k-colouring problem [7, 20]. In graph k-colouring the aim is to partition the vertices of G into k independent sets, whereas in the partial variant we are permitted to leave some vertices of the graph uncoloured. A candidate solution to the latter can therefore be written as $\mathcal{S} = \{S_1, \ldots, S_k\} \cup U$, where each $S_i \in \mathcal{S}$ is an independent set containing the vertices to be coloured with colour i, and U is the set of uncoloured vertices. The aim is to then identify the solution that minimises $|U|$. In contrast, for the MHIS problem we seek to minimise $|U|$ such that each $S_i \in \mathcal{S}$ corresponds to one or more happy components, as noted earlier.

2.3 Application Areas

We now consider some application areas that help to motivate the study of the MHIS problem.

Communications Networks: Consider a communications network where precoloured vertices represent end-users, free vertices represent routers, and edges join vertices that are physically connected. Now suppose that we wish to install monitoring devices that allow us to capture data sent between differently precoloured vertices (perhaps because they signify end-users who work for different agencies, or in different countries). In a feasible solution to the MHIS problem, all paths between pairs of differently-precoloured vertices will contain at least one uncoloured vertex; hence, an optimal solution will identify the smallest number of routers onto which we can place monitoring devices [10].

Social Networking: In social networks vertices are used to represent people, with edges then representing friendships between them. Now consider a situation where people need to be partitioned into groups

(e.g., for a team-building exercise), but that some of the people have already been preassigned to groups. Suppose further that people are "happy" only if they are assigned to a group containing their only friends. An optimal solution to the corresponding MHIS problem will maximise the number of happy people, with the remaining "unhappy" people not being assigned to any team.

Forest Root Systems: According to Simard et al. [26] the root systems of different trees in boreal forests often link to form networks, allowing the relaying of stress signals and the exchange of resources. Now imagine that we are interested in disconnecting a subset of trees, perhaps to stop the spread of disease, but that we want to minimise the number of trees to be felled. These felled trees will correspond to the uncoloured vertices in an optimal MHIS solution.

Stopping the Spread of Infection: Related to the previous example, an optimal solution to the MHIS problem will also identify the minimum number of people in a social network that need to be removed (quarantined) in order to stop an infection spreading to different predefined parts of the network (where these "parts" are defined using different colours).

3 Bounds for the MHIS Problem

In this section we consider some upper and lower bounds on the number of happy vertices achievable in different MHIS problem instances. We also introduce ways in which some problem instances can be broken up into smaller subproblems.

3.1 Bridging Vertices

Recall that a "bridging vertex" is a vertex with at least two neighbours that are precoloured differently (such as v_1 in Figure 1(a)). It is obvious that bridging vertices must remain uncoloured in any feasible solution; indeed, we can show the following:

Theorem 1. *$H(G)^* = |V'|$ if and only if all of G's free vertices are bridging vertices.*

Proof. Consider a graph with x bridging vertices. If $|V - V'| = x$ then all free vertices are bridging and must remain uncoloured in a feasible solution. Hence only the precoloured vertices are happy, giving $H(G)^* = |V'|$.

On the other hand consider a graph with x bridging vertices and $y \geq 1$ non-bridging free vertices. Now take a free non-bridging vertex v and make it precoloured by assigning it to any colour not currently allocated to one of its neighbours (at least $k - 1$ such colours will exist to do this). Since v is now precoloured (and happy) this means $H(G)^* > x$. Thus, if $H(G)^* = |V'|$, then all free vertices must be bridging. □

When all free vertices in a graph are bridging, there is clearly nothing to optimise, and the corresponding MHIS problem is trivial. We can also estimate the conditions under which these trivial instances will occur using certain assumptions. Consider an MHIS problem instance $G = (V, E)$ using k colours and assume that the number of precoloured vertices $|V'|$ in this graph is a positive multiple of k. Also assume that the precoloured vertices are chosen at random and assigned to random colours such that each colour class contains exactly $\bar{c} = |V'|/k$ precoloured vertices. Hence, $\bar{c} \in \mathbb{N}$. Now consider any free vertex $v \in (V - V')$ in this graph. If v is not a bridging vertex then any precoloured neighbours of v must have the same colour as each other and the remaining neighbours must be free; hence, all neighbours of v belong to a set of size $\bar{c} + |V - V'| - 1$. The complement of this event is that v is a bridging vertex, hence:

$$P(v \text{ is a bridging vertex}) = 1 - \frac{\binom{\bar{c}+|V-V'|-1}{\deg(v)}}{\binom{n-1}{\deg(v)}}. \tag{2}$$

The right term in this formula is derived from the hypergeometric distribution.

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

$$P(\text{all free vertices are bridging vertices}) = \left(1 - \frac{\binom{\bar{c}+|V-V'|-1}{\bar{d}}}{\binom{n-1}{\bar{d}}}\right)^{|V-V'|}. \tag{3}$$

The lines in Figure 2 illustrate the effects of this probability for differing values of n and k. The areas to the right of the lines indicate the combinations of parameters—mean degree $\bar{d}$ and proportion of precoloured vertices $|V'|/n$—that, with high confidence, result in problem instances where all free vertices are bridging. We see that these areas occupy the majority of the figures, with only very low values of one or both of the parameters leading to non-trivial problem instances. However, we must bear in mind that this model is based on the assumptions outlined above and will become less accurate when the selection of precoloured vertices is non-random or the variance across the vertex degrees is increased.

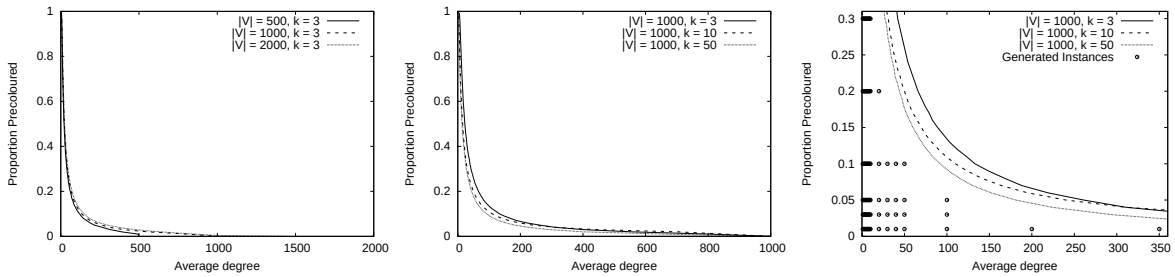

Figure 2: Areas to the right of the curves indicate parameter combinations that, with $\geq 95\%$ probability, lead to problem instances where all free vertices are bridging vertices, giving $H(G)^* = |V'|$. The points in the rightmost graph, which is a zoom-in of the middle graph, indicate the locations where our test instances (Section 5) were generated.

3.2 Lower Bounds

Lower bounds on the optimal number of happy vertices can be found by considering the maximum vertex degree in a graph $\Delta = \max(\deg(v) : v \in V)$. Recall that in this problem we do not allow any edges whose endpoints are precoloured differently.

Lemma 1. *If $\Delta \leq 1$ then $H(G)^* = n$.*

Proof. Clearly, in such graphs each component will either be an isolated vertex or a clique of size two. Free vertices belonging to cliques of size two should be assigned to the same colour as their neighbour. Isolated free vertices can be assigned to any arbitrary colour in $\{1, \ldots, k\}$. If such a process is followed then all vertices will be happy, giving $H(G)^* = n$. □

We now consider graphs for which $\Delta \geq 2$.

Theorem 2. *If $\Delta \geq 2$ then $H(G)^* \geq \max\left(|V'|, n - \left(\frac{k-1}{k}|V'|\Delta\right)\right)$*

Proof. According to this problem's definition, all precoloured vertices are happy; consequently, at least $|V'|$ vertices will always be happy in a feasible solution.

Now consider $A \subseteq V'$ to be the largest subset of precoloured vertices that are assigned to the same colour $a \in \{1, \ldots, k\}$. Hence, $|A| \geq \lceil \frac{1}{k}|V'| \rceil$ and $|V' - A| \leq \frac{k-1}{k}|V'|$. Let N denote the set of free vertices that have neighbours belonging to $V' - A$. The size of N is at most $|V' - A|\Delta$ (i.e. every neighbour in $V' - A$ has Δ distinct neighbours in N). In the worst case, every vertex in N will also have a neighbour belonging to A, making it a bridging vertex (which cannot be happy); however, any remaining free vertices in the graph (i.e., those belonging to $(V - V' - N)$) can be made happy by assigning them to colour a. Hence,

$$H(G)^* \geq n - |N| = n - (|V' - A|\Delta) \geq n - \left(\frac{k-1}{k}|V'|\Delta\right). \tag{4}$$

□

Note that when $k = 1$, all free vertices can be assigned to the same single colour, giving n happy vertices. The bound of Theorem 2 is therefore tight for this case.

Theorem 2 can also be used for the following corollary concerning d-regular graphs:

Corollary 1. *Let G be a d-regular graph. Then $H(G)^* \geq \max\left(|V'|, n - \left(\frac{k-1}{k}d|V'|\right)\right)$.*

Proof. This is found by simply replacing Δ with d in the proof of Theorem 2. □

We now consider random graphs G belonging to the set $\mathcal{G}(n, p)$. Graphs in this set have n vertices, and each pair of vertices $u, v \in V$ is adjacent with a fixed probability p.

Theorem 3. *Let $G \in \mathcal{G}(n, p)$ be a random graph. Then the expected number of happy vertices is greater than $n - (n - |V'|)p^{\lceil \frac{|V'|}{k} \rceil}$.*

Proof. Let $A \subseteq V'$ be the largest subset of precoloured vertices that are assigned to the same colour $a \in \{1, \ldots, k\}$. Now suppose we assign all vertices in the set $(V - V')$ to colour a and then uncolour all free vertices that have at least one neighbour in the set $(V' - A)$. This means that all coloured vertices in G are now happy. Observe that $\lceil \frac{|V'|}{k} \rceil \leq |A| \leq |V'| - k + 1$.

Now let X_i be a binary variable set to 0 if vertex v_i is adjacent to any vertex in the set $(V' - A)$ and 1 otherwise. The expected number of free vertices that are not adjacent to vertices in the set $(V' - A)$ will be at least:

$$\begin{aligned}
E[\sum_{v_i \in (V-V')} (X_i)] &= \sum_{v_i \in (V-V')} (E[X_i]) \\
&= \sum_{v_i \in (V-V')} P(v_i \text{ is not adjacent to any vertex in the set } (V' - A)) \\
&\geq \sum_{v_i \in (V-V')} (1 - p^{|V'-A|}) \\
&= (n - |V'|)(1 - p^{|V'-A|}) \\
&\geq (n - |V'|)(1 - p^{\lceil \frac{|V'|}{k} \rceil}). \qquad (5)
\end{aligned}$$

Thus the expected number of happy vertices is at least

$$\begin{aligned}
|V'| + (n - |V'|)(1 - p^{\lceil \frac{|V'|}{k} \rceil}) &= |V'| + n(1 - p^{\lceil \frac{|V'|}{k} \rceil}) - |V'| + |V'|p^{\lceil \frac{|V'|}{k} \rceil} \\
&= n(1 - p^{\lceil \frac{|V'|}{k} \rceil}) + |V'|p^{\lceil \frac{|V'|}{k} \rceil} \\
&= n - (n - |V'|)p^{\lceil \frac{|V'|}{k} \rceil} \qquad (6)
\end{aligned}$$

as required. □

3.3 Upper bounds

We can extend the ideas behind bridging vertices to also generate upper bounds on the number of happy vertices in a graph. This is achieved using what we call *unhappy paths*.

Definition 4. *Given a valid MHIS problem instance, an* unhappy path *is a simple path of length two or more (edges), whose internal vertices are free, and whose terminal vertices u and v are precoloured differently ($c(u) \neq c(v)$).*

Example unhappy paths from Figure 1(a) include $(v_2, v_3, v_4, v_5, v_6)$ and (v_6, v_7, v_8, v_9), of lengths four and three (edges) respectively.

Note that the presence of an unhappy path in a graph implies the necessary existence of uncoloured vertices in a solution, since at least one of its internal vertices need to be removed to allow the terminal vertices to exist in different happy components. Unhappy paths can therefore be used to generate an upper bound $\bar{H}(G)$ on $H(G)^*$. A process for doing this is described in the following steps using a copy of G. These steps involve first removing all bridging vertices from the graph, and then repeatedly identifying and removing unhappy paths while keeping count of the minimum possible number of uncoloured vertices x.

1. Let x be the number of bridging vertices in G. Remove all of these bridging vertices from G and go to Step 2.
2. Let P be an unhappy path in G. If no such path exists, go to Step 4; else go to Step 3.
3. Let u and v be the terminal vertices of the path P identified in Step 2. Remove all internal vertices in P from G, add one to x, and return to Step 2.
4. The upper bound $\bar{H}(G)$ is set to $n - x$.

Note that, in Step 2, any arbitrary unhappy path can be chosen in G. In this paper we use a greedy strategy of identifying and removing the shortest unhappy path. This is achieved by running breadth-first search from each non-isolated precoloured vertex. This results in the smallest number of vertices being removed from the current graph at each step. The intention is to allow the process to iterate for a larger number of cycles, thereby increasing x and improving $\bar{H}(G)$. Other strategies might also be applied, however.

Further reasoning might also allow us to improve $\bar{H}(G)$ in some cases. For example, consider a clique of free vertices in G that are each adjacent to a different precoloured vertex, and where these precoloured vertices all have different colours to one another. Examples of such cliques are shown in Figure 3. Note that if G contains one of these structures, then as soon as one of the free vertices in the clique is assigned to the same colour as its precoloured neighbour, any remaining vertices in the clique will become bridging vertices, meaning that they cannot be coloured in any feasible solution. In our method for generating upper bounds, we might therefore choose to include an additional step between Steps 1 and 2 that identifies these structures

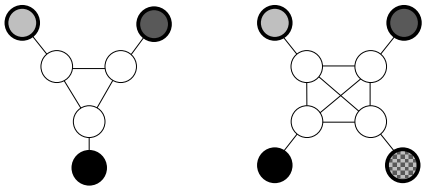

Figure 3: Example cliques of free vertices in a graph in which each vertex is also adjacent to a different precoloured vertex, and where these precoloured vertices have different colours to one another. (Cliques of size three (left), and four (right).)

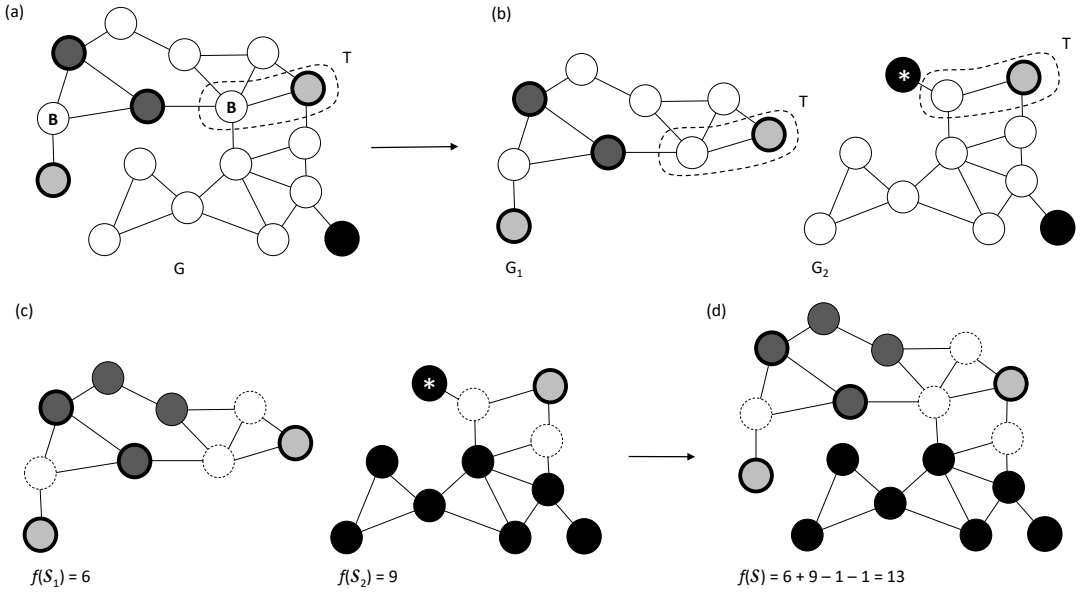

Figure 4: Example of how a graph G can be subdivided using a vertex cut set T that comprises only precoloured vertices and bridging vertices (marked with a B). Here T contains one happy vertex ($f(T) = 1$). Also, $\delta = 1$ dummy vertex (asterisked) has been created.

and, if the identified clique has l vertices, adds $l-1$ to x before removing these vertices. Note, however, that the task of identifying cliques in a graph is NP-hard in general, so this could add considerable expense to the procedure. It is therefore not used here.

3.4 Problem Subdivision

In this section we show how instances of the MHIS problem might be broken up into smaller parts, allowing us to tackle each part separately. Firstly, observe that if we have a disconnected graph G comprising multiple components $C_1, \ldots, C_l$ then the status of a vertex in one component has no effect on vertices in other components. This means that if we compute feasible solutions for each component separately, then the number of happy vertices in G is simply the sum of the number of happy vertices across all components. We now extend this idea to connected graphs.

Let $T \subseteq V$ be a vertex cut set in G comprising only precoloured vertices and bridging vertices (that is, vertices whose statuses are fixed as coloured and uncoloured respectively), and let $C_1, \ldots, C_l$ be the components resulting from the removal of T. Now, for each $C_i \in \{C_1, \ldots, C_l\}$, let G_i be the subgraph induced by the vertices of C_i and T. Finally, if G_i contains a vertex $v \in T$ that was a bridging vertex in G but is non-bridging in G_i, add additional dummy precoloured vertices and edges to G_i such that v is also bridging.

An example of this process is shown in Figure 4. In Part (a) a suitable vertex cut set T is indicated. Part (b) then shows the resultant subgraphs G_1 and G_2, together with an appropriate dummy vertex (asterisked). Part (c) then shows example feasible solutions to these subgraphs. The final solution to the original graph is formed by merging these solutions and deleting the dummy vertices, as shown in Part (d).

Theorem 4. *Let $G_1, \ldots, G_l$ be separate subgraphs constructed using an appropriate vertex cut set T, and let δ be the total number of dummy precoloured vertices created. In addition, let $\mathcal{S}_i$ be a feasible colouring*

(partition) of the vertices in G_i (for each $i \in \{1, \ldots, k\}$), and let $\mathcal{S}$ be a full feasible solution formed by merging $\mathcal{S}_1, \ldots, \mathcal{S}_l$. Then

$$f(\mathcal{S}) = \left(\sum_{i=1}^{l} f(\mathcal{S}_i) \right) - (l-1)f(T) - \delta. \tag{7}$$

Proof. Recall that a vertex is happy in a feasible solution if and only if it is coloured, else it is unhappy (and uncoloured). It is sufficient to show that a non-dummy vertex v is coloured in $\mathcal{S}_i$ if and only if it is also coloured in $\mathcal{S}$. Note that in each G_i, vertices originating from T must have the same status as their counterparts in G. If $v \in T$ is precoloured (and therefore happy) in G, then its set of neighbours in G_i are a subset of its neighbours in G; hence it is also happy in G_i. Similarly, if $v \in T$ is a bridging vertex in G, then the possible introduction of appropriate precoloured dummy vertices also ensures that it is a bridging vertex in G_i, and cannot therefore be coloured. We therefore now only need to consider vertices from the set $V(G) - T$ in G and $V(C_i)$ in G_i.

The proof is now trivial, since the neighbours of a vertex $v \in V(C_i)$ are either themselves in C_i, or are vertices in T (whose statuses are fixed). Hence the status of v has no effect on the status of any vertex outside of C_i.

Finally, the subtraction of $(l-1)f(T)$ is necessary to ensure that coloured vertices in T are only counted once when the subsolutions are combined to form $\mathcal{S}$. Similarly, δ is subtracted to ensure that dummy vertices are not counted in the final total. □

An immediate corollary of Theorem 4 is that $H(G)^* = \left(\sum_{i=1}^{l} H(G_i)^* \right) - (l-1)f(T) - \delta$. That is, if optimal solutions can be determined for each subgraph G_i, then the merged solution for G will also be optimal. This could be useful, for example, if each subgraph G_i is seen to contain fewer than three colours, as the overall problem could then be solved optimally in polynomial time.

4 Algorithms for the MHIS Problem

In this section we describe three different algorithms for the MHIS problem, one based on an integer programming (IP) formulation (Section 4.2) and two on tabu search (Sections 4.3 and 4.4). As we shall see, the IP method sometimes struggles with instances involving large numbers of vertices; our motivation for choosing tabu search is to therefore establish methods that are fast and that also scale well.

In all cases, a preprocessing step is first carried out to try to impose additional precolourings on the graph. Initial solutions are also produced using a specialised heuristic. These are described in the next subsection.

4.1 Preprocessing and Initial Solution Generation

For some MHIS problem instances it is possible to identify free vertices whose colours can be fixed, in effect adding additional precoloured vertices to a graph. For instance, if all the neighbours of a free vertex v are precoloured with colour j, then it is obvious that v should also assume colour j. A more generalised version of this process is as follows.

Starting at a free vertex v, let S be the set of vertices that are reachable from v in the subgraph induced by the free vertices of G. Now, using G consider the set of precoloured vertices that are adjacent to the vertices of $S \cup \{v\}$. If all of these precoloured vertices have the same colour j, then all vertices in $S \cup \{v\}$ can also be coloured with j. Similarly, if none of the vertices in $S \cup \{v\}$ have a precoloured neighbour, then they can all be assigned to the same arbitrary colour $j \in \{1, \ldots, k\}$. Repeat this process until all free vertices of G have been considered.

Obviously, fixing the colours of certain free vertices has the effect of reducing the size of the solution space while not changing the value of $H(G)^*$. Note also that this process will determine the colours of all free vertices in happy components; as a result, it will always achieve an optimal solution if $H(G)^* = n$.

Our procedure for producing an initial feasible solution $\mathcal{S} = (S_1, \ldots, S_k) \cup U$ (and therefore a lower bound on the number of happy vertices) is as follows.

Given a problem instance G, take each component $C_1, \ldots, C_l$ one by one. For each C_i, we then take each colour $j \in \{1, \ldots, k\}$ in turn and colour all free vertices with j, providing that they do not have a neighbour that is precoloured to something other than colour j. Our final solution for this component is formed using the colour that maximises the number of coloured (happy) vertices under this process. An example for a single component using the problem from Figure 1(a) is shown in Figure 5. Note that this method is exact when $H(G)^* = |V'|$ (as none of the free vertices will be coloured) and when $H(G)^* = n$ (all free vertices will be in happy components and will therefore be coloured).

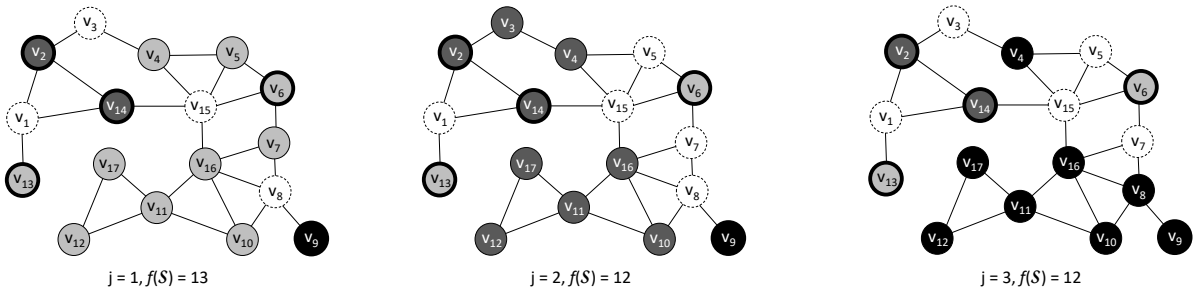

Figure 5: Example initial solutions for the problem shown in Figure 1 using colour $j = 1$ (light grey, left), $j = 2$ (dark grey, centre), and $j = 3$ (black, right). In this case the best solution is obtained using colour $j = 1$, giving thirteen happy vertices.

4.2 IP Formulation

Our IP model for the MHIS problem is defined using binary variables x_{ij}, where $x_{ij} = 1$ if vertex v_i is assigned to colour j and $x_{ij} = 0$ otherwise. In addition to the k colours, in this formulation a dummy colour with label $k + 1$ is used to indicate vertices that will not be coloured in the final solution. The objective is to then maximise the number of vertices assigned to colours $1, \ldots, k$, which is equivalent to minimising the number of vertices assigned to colour $k+1$. We now seek to maximise the objective function $n-(\sum_{i=1}^{n} x_{i,k+1})$ subject to:

$$\sum_{j=1}^{k+1} x_{ij} = 1 \quad \forall v_i \in V \tag{8}$$

$$x_{ij} = 1 \quad \forall v_i \in V' : c(v_i) = j \tag{9}$$

$$x_{i_1,j_1} + x_{i_2,j_2} \leq 1 \quad \forall \{v_{i_1}, v_{i_2}\} \in E, \forall j_1, j_2 \in \{1, \ldots, k\} : j_1 \neq j_2. \tag{10}$$

Constraint (8) states that each vertex should be assigned to exactly one of the $k + 1$ colours; Constraint (9) ensures that the precolourings are obeyed; and Constraint (10) prevents adjacent vertices from being assigned to different colours that both belong to the set $\{1, \ldots, k\}$.

For our trials this IP model was augmented by first running the steps outlined in Section 4.1, giving us a lower bound and perhaps some additional precoloured vertices. The initial solution was then used to warm-start the IP optimiser. In addition, for each instance we also calculated the upper bound $\bar{H}(G)$ using the methods from Section 3.3 allowing us to impose an additional constraint:

$$n - \left(\sum_{i=1}^{n} x_{i,k+1} \right) \leq \bar{H}(G) \tag{11}$$

We found that these augmentations nearly always improved the success rates and run times of the IP optimiser.[1]

An alternate formulation of the IP can also be obtained by replacing Constraint (10) by:

$$x_{i_1,j} + x_{i_1,k+1} \geq x_{i_2,j} \quad \forall \{v_{i_1}, v_{i_2}\} \in E, \forall j \in \{1, \ldots, k\}. \tag{12}$$

This constraint leads to fewer constraints overall, though, the formulation is relatively weak. In initial experiments it was found that solutions to most problem instances could not be found within 10 minutes.[2] Hence the remainder of this paper focuses on the IP model using Constraint (10).

4.3 Tabu Search

The second optimisation method used in our experiments is based on tabu search—a metaheuristic based on local search, but which also has mechanisms to help avoid cycling, therefore encouraging the algorithm to enter previously unexplored regions of the solution space. The general approach operates by taking an initial

[1] We conducted preliminary experiments on scale-free graphs with 1000 vertices and 10 colours with and without Constraint (11). We found that the average run time using $\bar{H}(G)$ was 70.1 seconds whereas the average run time without it was 123.3 seconds. Moreover, an improvement of run time was seen in 94% of the problem instances.

[2] We allow a 10 minute run-time limit for all our experiments. Considering problem instances with d-regular graphs and 50 colours, solutions were found for 8% of the problem instances using Constraint 12, whereas solutions were found for 90% of the problem instances using Constraint 10.

solution $\mathcal{S}$ and then repeatedly applying a neighbourhood operator to try to make improvements. At the same time, the algorithm also makes use of a memory structure called a "tabu list" that specifies parts of the solution space that cannot be visited, perhaps because they have already been considered by the algorithm in previous iterations. This is used to help the algorithm "home in" on high-quality solutions.

Our algorithm, which we refer to here as TABU, is based on similar applications of tabu search with other types of graph partitioning problem [7, 21], albeit with different operators tailored to the MHIS problem. Having produced an initial feasible solution $\mathcal{S} = (S_1, \ldots, S_k) \cup U$, the algorithm proceeds using the objective function $f(\mathcal{S}) = \sum_{i=1}^{k} |S_i| = n - |U|$, which gives the number of happy (coloured) vertices. Given a feasible solution $\mathcal{S}$, a move in the solution space is achieved by selecting an uncoloured non-bridging vertex $v \in U$ and assigning it to a colour class S_j. The move is then completed by examining each vertex $u \in \Gamma(v)$ and, if u is assigned to a colour class $S_{i \neq j}$, removing its colour by transferring it to U. Note that this operator will retain solution feasibility in all cases except where a vertex $u \in \Gamma(v)$ is precoloured to something other than colour j. If this is the case, the move is not carried out. Similarly, a move will increase the objective function value (by one) only when the coloured neighbours of v are already assigned to colour class S_j.

The tabu list of this algorithm is stored using a matrix $T_{n \times k}$. At iteration l of the algorithm, if a move is performed that involves transferring a vertex u from S_i to U, the element T_{ui} is set to $l + t$, where t is a positive integer that will be defined presently. This signifies that the transfer of u back to colour class S_i is tabu (not allowed) for the next t iterations of the algorithm. Note that this has the effect of making all solutions containing the assignment of vertex u to S_i tabu.

As is typical in tabu search, in each iteration of this algorithm the entire set of $|U| \times k$ neighbouring solutions is evaluated. The non-tabu move that brings about the largest increase (or failing that, the smallest decrease) in the objective function is then performed. Any ties in this criterion are broken randomly. In addition, we use also an aspiration criterion that allows tabu moves to be made if they are seen to improve on the best solution found so far during the run. Finally, if all possible moves are seen to be tabu, then a randomly selected move is performed. The tabu list is then updated as usual.

As we might expect, the process of evaluating all possible moves at each iteration has the potential to consume the majority of this algorithm's execution time. However, considerable speedups can be achieved using an additional matrix $C_{n \times k}$ where, given the current solution $\mathcal{S}$ and a vertex $v \in U$, element C_{vj} indicates the number of vertices that would be added to U if v were to be moved to colour class S_j. To choose the next move to perform, we then only need to scan rows of C that correspond to non-bridging vertices in U. Once a particular move has been performed, only the relevant parts of C then need to be updated: namely, all rows corresponding to vertices in U or vertices within distance one or two of v in G.

Finally, with regards to the tabu tenure t, various studies for graph partitioning problems have found that a successful approach involves making t a random variable that is proportional to the current solution's quality [7, 20]. The idea is that when the solution is of low quality, t should assume higher values, which will hopefully compel the algorithm to seek out new regions of the solution space. On the other hand when the solution has a high quality, the algorithm should focus more on the current region by using low values for t. To these ends we choose to set $t = r + \tau(\bar{H}(G) - f(\mathcal{S}))$, where r is an integer randomly selected from the set $\{0, 1, \ldots, 9\}$ at each iteration, and $\tau \in \mathbb{R}^+$ is a parameter to be tuned.

4.4 Tabu Search with Diversification

Our final optimisation method TABU+ operates in the same way as TABU except that it also includes a diversification operator that allows large jumps to be made within the solution space from time-to-time. This operator was introduced because, for some instances, TABU was seen to quickly move to a fairly good solution but then make no further improvements for the remainder of the run, perhaps suggesting that it was caught in a region of the solution space surrounding a local optima. The intention of the diversification operator is to therefore make relatively large changes, encouraging new parts of the solution space to be explored.

Our proposed diversification operator exploits the underlying structure of the MHIS problem in that it makes large changes to a solution but without compromising its quality. Using a candidate solution $\mathcal{S} = (S_1, \ldots, S_k) \cup U$, let G_i be the subgraph induced from G using the vertices belonging to S_i only, and let $C_{i1}, C_{i2}, \ldots, C_{il}$ be the components of G_i. A *free i-component* is any of these components that comprises free vertices only. Note that all neighbours of vertices in a free i-component are either themselves free and coloured with i, or are free and uncoloured. Consequently, if all vertices in a free i-component are reassigned to a new colour j, then the resultant solution will remain feasible and contain the same number of happy vertices.

Our diversification operator considers each colour i in turn and randomly identifies a single free i-component (if indeed one exists for this colour). The vertices in this component are then assigned to a new colour $j \neq i$, selected at random. Once all colours $i = 1, \ldots, k$ have been considered, the tabu list is

reset so that all potential moves are deemed non-tabu. Tabu search then continues as usual.

In our experiments, the diversification procedure was applied whenever the best solution found so far during the run had not improved over the previous ten seconds. In each application, a random 50:50 choice was also made between perturbing the run's current solution or the best solution seen so far. The latter option allowed the algorithm to sometimes focus its attentions on the very best solutions, which we found was sometimes necessary to get closer to globally optimal solutions.

5 Experimental Analysis

In this section we make use of our proposed methods to explore the characteristics that make instances of the MHIS problem difficult to solve. Our IP model was implemented using Gurobi Optimiser Version 8.0.0 and executed with memory limits of 50 GB per-run. All remaining procedures were coded in C++ and can be downloaded at [2], together with a full listing of our experimental data. Trials were carried out using Monash University's computing cluster MonARCH using a CPU time limit of 600 seconds per trial.[3] Our problem instance generator, also programmed in C++, is available at [1]. Note that due to space reasons we do not include all tables of results here; instead, each subsection shows results indicative of general trends. We then make comments on other cases as necessary. A full set of results tables can be found in supplementary material [3].

5.1 Problem Instance Generation

Three graph topologies were considered in our experiments: random, d-regular, and scale-free. Random graphs were produced by starting with a set of n isolated vertices and then adding edges between each pair of vertices with a fixed probability p. This leads to a graph with approximately $\binom{n}{2}p$ edges and a binomial-shaped degree distribution with mean $(n-1)p$ and variance $(n-1)p(1-p)$. d-regular graphs are similar to random graphs except that the degree of each vertex is equal to d, where $d \in \{0, 1, \ldots, n-1\}$. These were produced using the randomised method of Steger and Wormald [27].

Scale-free graphs were also considered in our trials because they are known to model many real-world networks such as the World Wide Web, social networks, and the citation networks of academic papers [6]. They are based around the idea of "preferential attachment" in that, when a vertex is added to a graph, it is more likely to be made adjacent to vertices of high degrees. The degree distributions of scale-free graphs therefore follow a power law, in which a small number of "hub" vertices feature disproportionately high degrees compared to the remaining vertices. In our experiments scale-free graphs were produced using the Barabási-Albert method [6], using a parameter $q \in \{0, 1, \ldots, n\}$. We start with a complete graph $G = (V, E) = K_q$, comprising q vertices and $\binom{q}{2}$ edges. In each step a new vertex v is then added to G together with q edges that connect v to vertices already in G. This is done via a series of q roulette-wheel trials where, in each case, the probability $P(u, v)$ of adding the edge $\{u, v\}$ to E is calculated as

$$P(u,v) = \begin{cases} \dfrac{\deg(u)}{\sum_{w \in (V - \Gamma(v))} \deg(w)} & \text{if } \{u,v\} \notin E \\ 0 & \text{otherwise.} \end{cases} \tag{13}$$

Here, $(V - \Gamma(v))$ denotes the set of vertices in G that are not yet adjacent to v. Vertices are added in this way until a graph with n vertices and $m = \binom{q}{2} + q(n-q)$ edges is formed. According to this method, a setting of $q = 0$ gives an empty graph on n vertices, $q = 1$ leads to acyclic graphs, and $q = n-1$ or $q = n$ gives the complete graph K_n.

Upon producing the desired graph, the precoloured vertices need to be specified. In our case this is done randomly while also ensuring that each colour $\{1, \ldots, k\}$ is used at least once. This leads to approximately equal numbers of vertices per colour. Note that when values for $|V'|$, k, and/or the graph density are high, it may not be possible to assign colours to vertices while ensuring that adjacent vertices are precoloured differently. In our case our implemented generator makes 100 independent attempts at precolouring the vertices and gives up if none of these are successful. An alternative approach would be to allow adjacent vertices to be precoloured differently and then create additional bridging vertices between them, though this would also alter the underlying structure of the graphs.

For the experiments in the next three subsections a total of 12,840 problem instances were used. Specifically, for each density value and proportion of precoloured vertices (indicated by the points in the rightmost chart in Figure 2), twenty 1000-vertex instances were generated. This was done using random, d-regular and

[3]Each machine on MonARCH has 24 cores and 256 GB RAM and each physical core consists of two hyper-threaded cores with Intel Xeon E5-2680 v3 2.5GHz, 30M Cache, 9.60GT/s QPI, Turbo, HT, 12C/24T (120W).

scale-free graphs for $k \in \{3, 10, 50\}$. Note that when the desired number of precoloured vertices in a graph is less than k, then a valid problem instance cannot be generated. In addition, instances could not be generated for higher densities using $k = 50$ because we were unable to assign the desired number of precoloured vertices while ensuring that adjacent vertices had different colours. Such parameter combinations are therefore not included in our tables of results.

Finally, note that we do not consider the subdivision of problems in our experiments here because we have found that the necessary separating sets do not seem to naturally arise in the graphs generated using our methods. They may occur more frequently if problems are generated with more targeted choices of precoloured vertices and/or using different graph topologies, however. For example, if problem instances were known to feature precoloured articulation points, dividing up graphs may well turn out to be a useful strategy.

5.2 Preprocessing and the Number of Happy Vertices

In this subsection we consider the factors that allow further precoloured vertices to be added to a graph using our preprocessing step from Section 4.1. We also give an indication of the number of happy vertices that are achievable for different types of problem instance by reporting the quality of the "best observed" solutions. For each problem instance, this value was determined by running our IP model and two tabu search variants and taking the best solution from among these. As we see in the next subsection, in many cases the best observed solutions are actually the proven global optima due to the success of the IP optimiser across this instance set.

Table 1 summarises our results for all random graphs in our test set, distinguishing instances by the number of colours k, the mean degree, and the proportion of precoloured vertices. For very sparse graphs we see that the preprocessing step has a considerable effect because the lack of edges means that many of the vertices satisfy the criteria stated in Section 4.1. For the same reason, these effects also increase slightly for instances using fewer colours. Similar patterns were also observed with d-regular and scale-free graphs, albeit with further precolourings only tending to be added for instances with mean degrees of less than three due to their more predictable topologies [3]. For 1-regular graphs, we also saw that the preprocessing step was able to precolour all free vertices, giving $H(G)^* = n$. This is due to Lemma 1.

Table 1 also shows the number of happy vertices that we can expect to achieve with different 1000-vertex random graphs. In general we see that a greater number of vertices can be happy when graphs are sparse and/or when they feature a low proportion of precoloured vertices. Such graphs represent less constrained problem instances, with free vertices having fewer neighbours whose colours they need to match. The same patterns were also observed with d-regular and scale-free graphs [3].

5.3 IP Optimiser Performance

In this section we examine the performance of the IP optimiser on our problem instance set. In general we found this method to be very successful, with 10,744 of our 12,840 problem instances (83.7%) being solved to a proven optimality within the 600 second time limit; however, we were also able to identify pockets of problem instances that presented much more difficulty.

Results for random graphs are summarised in Table 2. Success rates in the table give the proportion of instances for which a proven optimum was achieved within the time limit. Bold text is used to highlight success rates of less than one. Where possible, mean run times and optimality gaps are also given. We see that when $k = 3$, instances are solved in all cases with mean run times not exceeding six seconds; however, for $k \in \{10, 50\}$, there are clearly areas where the algorithm experiences more difficulties.

Table 2 shows that for $k \in \{10, 50\}$, easy-to-solve problem instances seem to occur in two places: (a) where problems are under-constrained, in that they have relatively few edges and/or precoloured vertices; and (b) where problems are over-constrained, because they contain very many edges and/or precoloured vertices. Difficult problems lie at the boundaries of these extremes. Areas of difficulty such as this are often known as a phase transition regions and have previously been shown to exist in other areas of combinatorial optimisation including graph k-colouring, timetabling, constraint satisfaction, and Sudoku problems [9, 12, 14, 19, 25]. Gaps between the best observed solutions and the upper bound $\bar{H}(G)$ (Section 3.3) were also seen to be largest within these phase transition regions.

For $k = 50$ we also see that the phase transition regions are wider and the optimality gaps are much higher, indicating that the IP optimiser finds these problem instances much more challenging. Indeed in the cases indicated by asterisks in the Table 2, the optimiser was not even able to complete its initial stages of pre-solve within the time limit, meaning that improvements on the initial solution were not even attempted. These difficulties seem due to Constraint (10) in our IP model, which is expressed across every edge and

	Proportion of Vertices Precoloured					
Mean deg.	0.01	0.03	0.05	0.1	0.2	0.3
$k = 3$						
1	10 (965.0) [999.6]	30 (923.0) [998.2]	50 (912.3) [996.7]	100 (890.8) [991.3]	200 (876.8) [979.3]	300 (880.9) [967.3]
2	10 (212.3) [991.2]	30 (237.4) [973.7]	50 (263.4) [959.8]	100 (323.1) [930.9]	200 (431.6) [896.1]	300 (524.9) [872.8]
3	10 (68.1) [984.6]	30 (91.0) [955.6]	50 (113.0) [926.9]	100 (169.4) [868.6]	200 (278.9) [810.1]	300 (382.1) [775.8]
4	10 (31.0) [978.7]	30 (52.3) [935.0]	50 (73.1) [897.7]	100 (125.2) [817.9]	200 (229.6) [731.3]	300 (332.3) [693.7]
5	10 (17.3) [972.8]	30 (37.8) [920.7]	50 (58.1) [874.1]	100 (109.0) [768.7]	200 (212.0) [664.7]	300 (313.3) [625.1]
6	10 (13.3) [967.7]	30 (33.3) [905.0]	50 (53.6) [846.7]	100 (104.0) [728.7]	200 (205.3) [612.2]	300 (305.8) [571.7]
7	10 (11.1) [961.0]	30 (31.3) [889.2]	50 (51.4) [825.9]	100 (101.9) [692.6]	200 (202.2) [561.3]	300 (302.8) [523.4]
8	10 (10.5) [956.9]	30 (30.5) [872.1]	50 (50.6) [801.3]	100 (100.7) [662.1]	200 (200.8) [523.2]	300 (300.9) [481.4]
9	10 (10.2) [951.7]	30 (30.3) [858.4]	50 (50.3) [783.2]	100 (100.4) [624.1]	200 (200.4) [485.1]	300 (300.4) [447.7]
10	10 (10.1) [945.9]	30 (30.1) [841.4]	50 (50.1) [758.1]	100 (100.2) [595.6]	200 (200.2) [454.3]	300 (300.2) [421.9]
20	10 (10) [899.1]	30 (30) [723.2]	50 (50) [589.0]	100 (100) [384.6]	200 (200) [283.5]	
30	10 (10) [849.0]	30 (30) [610.8]	50 (50) [451.4]	100 (100) [265.1]		
40	10 (10) [807.8]	30 (30) [522.8]	50 (50) [358.9]	100 (100) [217.4]		
50	10 (10) [763.5]	30 (30) [461.5]	50 (50) [306.2]	100 (100) [175.9]		
100	10 (10) [608.2]	30 (30) [275.5]	50 (50) [195.0]			
200	10 (10) [354.3]					
350	10 (10) [201.5]					
$k = 10$						
1	10 (956.6) [999.5]	30 (911.5) [997.4]	50 (897.4) [995.6]	100 (875.1) [988.8]	200 (859.0) [972.7]	300 (857.9) [957.5]
2	10 (212.2) [986.6]	30 (237.1) [963.8]	50 (262.2) [945.8]	100 (320.3) [915.7]	200 (423.6) [879.7]	300 (510.6) [847.6]
3	10 (68.2) [976.3]	30 (90.9) [935.7]	50 (112.6) [900.6]	100 (168.1) [830.9]	200 (274.0) [778.3]	300 (373.6) [739.8]
4	10 (31.0) [967.1]	30 (52.3) [910.0]	50 (73.1) [859.1]	100 (124.3) [762.0]	200 (227.2) [691.1]	300 (328.7) [651.5]
5	10 (17.3) [958.2]	30 (37.8) [887.6]	50 (58.1) [826.3]	100 (109.1) [704.0]	200 (211.8) [623.9]	300 (311.8) [580.2]
6	10 (13.3) [949.9]	30 (33.3) [867.4]	50 (53.5) [792.0]	100 (103.9) [655.5]	200 (204.6) [565.7]	300 (304.6) [525.2]
7	10 (11.1) [941.2]	30 (31.3) [846.8]	50 (51.4) [760.1]	100 (101.8) [606.3]	200 (201.9) [514.2]	300 (301.9) [476.1]
8	10 (10.5) [931.9]	30 (30.5) [823.6]	50 (50.6) [732.6]	100 (100.7) [564.7]	200 (200.7) [469.9]	300 (300.7) [434.4]
9	10 (10.2) [924.0]	30 (30.3) [807.6]	50 (50.3) [704.4]	100 (100.4) [525.1]	200 (200.4) [431.9]	300 (300.3) [397.5]
10	10 (10.1) [915.7]	30 (30.1) [788.1]	50 (50.1) [677.4]	100 (100.2) [491.0]	200 (200.2) [401.8]	300 (300.1) [372.8]
20	10 (10) [837.5]	30 (30) [626.2]	50 (50) [469.5]	100 (100) [279.7]	200 (200) [232.7]	
30	10 (10) [764.5]	30 (30) [500.8]	50 (50) [336.7]	100 (100) [194.4]		
40	10 (10) [702.3]	30 (30) [390.9]	50 (50) [238.2]	100 (100) [148.0]		
50	10 (10) [637.0]	30 (30) [321.4]	50 (50) [179.6]	100 (100) [119.2]		
100	10 (10) [396.3]	30 (30) [115.5]	50 (50) [74.8]			
200	10 (10) [147.1]					
350	10 (10) [32.1]					
$k = 50$						
1			50 (892.1) [994.9]	100 (869.7) [987.7]	200 (855.2) [971.2]	300 (853.1) [954.8]
2			50 (262.6) [935.3]	100 (320.0) [907.2]	200 (421.3) [873.8]	300 (505.9) [839.8]
3			50 (112.9) [884.2]	100 (167.7) [816.8]	200 (272.6) [769.5]	300 (371.0) [729.7]
4			50 (72.9) [838.4]	100 (124.1) [744.3]	200 (226.8) [680.5]	300 (327.7) [639.1]
5			50 (58.2) [798.6]	100 (109.0) [679.4]	200 (211.4) [611.7]	300 (311.4) [567.6]
6			50 (53.6) [760.1]	100 (103.9) [627.4]	200 (204.5) [554.2]	300 (304.2) [511.0]
7			50 (51.4) [723.4]	100 (101.8) [578.0]	200 (201.8) [502.0]	300 (301.7) [462.5]
8			50 (50.6) [689.4]	100 (100.6) [532.0]	200 (200.6) [458.3]	300 (300.5) [421.4]
9			50 (50.3) [658.6]	100 (100.4) [488.5]	200 (200.4) [420.2]	300 (300.2) [385.2]
10			50 (50.2) [626.9]	100 (100.2) [456.3]	200 (200.2) [390.0]	300 (300.1) [358.7]
20			50 (50) [394.4]	100 (100) [260.1]	200 (200.0) [224.6]	
30			50 (50) [253.2]	100 (100) [182.6]		
40			50 (50) [174.6]	100 (100) [138.0]		
50			50 (50) [130.2]	100 (100) [109.9]		
100			50 (50) [64.3]			

Table 1: Number of happy (precoloured) vertices in the initial problem instance, after the preprocessing step (in parentheses), and in the best observed solution (in square braces), for random graphs with differing k-values, densities, and proportions of precoloured vertices. Each figure is the mean from twenty problem instances using $n = 1000$.

every pair of colours. Larger values of k therefore lead to more constraints, leading to very large memory requirements and processing times.

Phase transition regions were also seen to exist in the same areas with scale-free and d-regular graphs. For d-regular graphs these regions also seemed to be wider and more pronounced, with dips in success rates even occurring with 3-colour problem instances, as shown in Figure 6. By definition, in d-regular graphs, the variance across vertex degrees is zero, so from an algorithmic point-of-view this might make all vertices "look the same", reducing the amount of heuristic information to be exploited. Similarly, scale-free graphs feature much higher levels of degree variance, and their phase transition regions were seen to be the least drastic of the three topologies [3].

	Proportion of Vertices Precoloured					
Mean Deg.	0.01	0.03	0.05	0.1	0.2	0.3
$k = 3$						
1, 2	1.00 (0)	1.00 (0)	1.00 (0)	1.00 (0)	1.00 (0)	1.00 (0)
3	1.00 (1)	1.00 (0)	1.00 (1)	1.00 (1)	1.00 (0)	1.00 (0)
4	1.00 (1)	1.00 (1)	1.00 (1)	1.00 (1)	1.00 (0)	1.00 (0)
5	1.00 (2)	1.00 (1)	1.00 (1)	1.00 (1)	1.00 (1)	1.00 (0)
6	1.00 (2)	1.00 (2)	1.00 (1)	1.00 (2)	1.00 (3)	1.00 (0)
7	1.00 (4)	1.00 (2)	1.00 (2)	1.00 (2)	1.00 (2)	1.00 (0)
8	1.00 (4)	1.00 (3)	1.00 (2)	1.00 (1)	1.00 (2)	1.00 (0)
9	1.00 (4)	1.00 (3)	1.00 (1)	1.00 (2)	1.00 (0)	1.00 (0)
10	1.00 (5)	1.00 (3)	1.00 (2)	1.00 (2)	1.00 (0)	1.00 (0)
20	1.00 (1)	1.00 (0)	1.00 (1)	1.00 (1)	1.00 (0)	
30	1.00 (2)	1.00 (1)	1.00 (0)	1.00 (0)		
40	1.00 (3)	1.00 (1)	1.00 (0)	1.00 (0)		
50	1.00 (3)	1.00 (1)	1.00 (0)	1.00 (1)		
100	1.00 (6)	1.00 (1)	1.00 (1)			
200	1.00 (4)					
350	1.00 (6)					
$k = 10$						
1	1.00 (0)	1.00 (0)	1.00 (0)	1.00 (0)	1.00 (0)	1.00 (0)
2	1.00 (64)	1.00 (67)	1.00 (47)	1.00 (90)	1.00 (1)	1.00 (0)
3	1.00 (122)	1.00 (177)	1.00 (173)	**0.10 (284) [0.006]**	1.00 (18)	1.00 (0)
4	1.00 (128)	1.00 (202)	1.00 (281)	**0.20 (245) [0.007]**	1.00 (103)	1.00 (0)
5	1.00 (81)	1.00 (134)	**0.95 (238) 0.073**	**0.20 (285) [0.009]**	1.00 (120)	1.00 (0)
6	1.00 (57)	1.00 (78)	1.00 (207)	**0.20 (235) [0.010]**	**0.75 (77) [0.003]**	1.00 (1)
7	1.00 (73)	1.00 (71)	1.00 (161)	**0.15 (259) [0.016]**	**0.95 (35) [0.006]**	1.00 (1)
8	1.00 (88)	1.00 (78)	1.00 (103)	**0.20 (361) [0.022]**	1.00 (23)	1.00 (1)
9	1.00 (109)	1.00 (65)	1.00 (96)	**0.10 (170) [0.024]**	1.00 (2)	1.00 (1)
10	1.00 (134)	1.00 (71)	1.00 (85)	**0.10 (61) [0.033]**	1.00 (2)	1.00 (2)
20	1.00 (169)	1.00 (40)	1.00 (54)	**0.10 (15) [0.047]**	1.00 (4)	
30	1.00 (237)	1.00 (31)	**0.95 (40) [0.027]**	1.00 (10)		
40	1.00 (272)	1.00 (26)	**0.90 (34) [0.039]**	1.00 (9)		
50	1.00 (290)	1.00 (21)	**0.75 (39) [0.026]**	1.00 (11)		
100	1.00 (247)	1.00 (25)	1.00 (23)			
200	1.00 (69)					
350	1.00 (82)					
$k = 50$						
1			1.00 (23)	1.00 (12)	1.00 (8)	1.00 (6)
2			**0.00 (-) [0.028]**	**0.25 (361) [0.017]**	1.00 (49)	1.00 (16)
3			**0.00 (-) [0.056]**	**0.00 (-) [0.037]**	**0.90 (92) [0.002]**	1.00 (21)
4			**0.00 (-) [0.078]**	**0.00 (-) [0.053]**	**0.60 (117) [0.003]**	1.00 (25)
5			**0.00 (-) [0.098]**	**0.00 (-) [0.048]**	**0.60 (265) [0.004]**	1.00 (29)
6			**0.00 (-) [0.115]**	**0.00 (-) [0.043]**	**0.65 (223) [0.004]**	1.00 (33)
7			**0.00 (-) [0.133]**	**0.00 (-) [0.043]**	**0.85 (119) [0.003]**	1.00 (39)
8			**0.00 (-) [0.143]**	**0.00 (-) [0.052]**	**0.95 (59) [0.002]**	1.00 (43)
9			**0.00 (-) [0.164]**	**0.00 (-) [0.060]**	1.00 (58)	1.00 (49)
10			**0.00 (-) [0.181]**	**0.00 (-) [0.067]**	1.00 (59)	1.00 (55)
20			**0.00 (-) [0.118]**	**0.00 (-) [0.067]**	1.00 (112)	
30			**0.00 (-) [0.144]**	1.00 (195)		
40			**0.00 (-) [0.406]***	**0.00 (-) [0.181]***		
50			**0.00 (-) [0.434]***	**0.00 (-) [0.057]***		
100			**0.00 (-) [0.182]***			

Table 2: Success rates of the IP optimiser for random graphs. For non-zero success rates, figures in parenthesis give the mean CPU time (to the nearest second) of successful runs. For success rates of less than one, figures in square braces indicate the mean optimality gap in unsuccessful runs at the time limit (calculated as the difference between the IP optimiser's upper and lower bounds, divided by the upper bound). Each figure is the mean from twenty problem instances using $n = 1000$.

5.4 IP and Tabu Search Comparison

We now consider the relative performance on the IP and tabu algorithms. All trials with the latter were conducted using a setting of $\tau = 2$. This value was determined using a test set comprising the first instance of every set of twenty considered in the previous subsection (giving $12840/20 = 642$ instances in total). Tabu search was then run on this set for 600 seconds using values of $\tau \in \{0, 0.1, 0.2, 0.4, 0.6, 0.8, 1, 2, 3, 4, 5, 8, 10\}$. Our results indicated that $\tau = 2$ resulted in the largest number of happy vertices across these instances, and that movements away from $\tau = 2$ resulted in fewer happy vertices overall. Note, however, that these falls in performance were only marginal: the difference between the worst result ($\tau = 0$) and the best was fewer than one vertex in a thousand, suggesting that the algorithm is not overly sensitive to changes in this

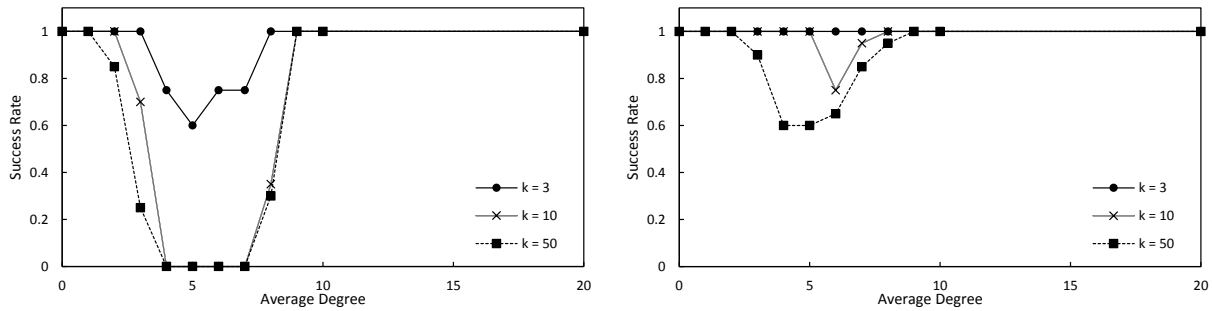

Figure 6: Success rates achieved by the IP optimiser with various d-regular (left) and random (right) graphs in which 20% of vertices are precoloured. All points are means across twenty problem instances.

	Time (seconds)			
Algorithm	<1	10	60	600
IP	41.2%	75.7%	84.7%	94.2%
TABU	41.2%	72.9%	73.9%	75.3%
TABU+	41.2%	72.9%	76.2%	80.8%

Table 3: Percentage of problem instances where the best observed solution was achieved by each algorithm at the stated time.

parameter.

An obvious advantage of using IP techniques with this problem is its ability to decrease the upper bound over time and therefore often report with certainty when an optimal solution has been achieved. On the other hand, applications of our tabu search algorithms can only report optimality when a solution with $H(G)^* = \bar{H}(G)$ happy vertices has been achieved, which may not even be possible for many problem instances. Nevertheless, for our analysis we use the optimal solutions determined by the IP optimiser to help gauge the accuracy of the tabu search algorithms.

As noted earlier, our trials with IP optimiser saw 10,744 of the 12,840 instances being solved to optimality. Of these solved instances, the TABU algorithm found solutions of equal quality for 8,514 (79.2%). For TABU+ this figure increased to 8,907 (82.9%). On the other hand, of the 2,096 instances not solved by the IP optimiser, TABU was seen to return better solutions for 673 (32.1%), worse solutions for 793 (37.9%), with the remainder tied. The corresponding figures for TABU+ were slightly better at 740 (35.3%) and 630 (30.0%) respectively.

Considering the set of 1000-vertex instances as a whole, Table 3 shows, for each algorithm and various times, the percentage of problem instances where the best observed solution[4] was achieved. Here, the column labelled < 1 refers to the initial solutions produced using our heuristic method from Section 4.1; hence, in 41.2% of instances, these initial solutions are equal in quality to the best observed solutions. Table 3 clearly demonstrates the superiority of the IP optimiser with these problems, even after just ten seconds of run time. TABU, on the other hand, seems to make progress in the first ten seconds, but only marginal improvements after that. As shown, this feature is alleviated to a certain extent by the diversification operator used with TABU+, though its results are still inferior to the IP optimiser with these instances overall.

Table 4 uses random graphs to indicate the types of problem instance for which the IP optimiser and TABU+ outperform each other. Consistent findings were again seen with the other graph topologies considered [3]. For smaller numbers of colours ($k \in \{3, 10\}$) the IP optimiser is consistently superior to TABU+, with its benefits usually being seen in and around the phase transition areas. This is despite the fact that the IP optimiser has often been unable to achieve optimality with these instances. The quality of the algorithms' solutions are usually quite similar over these regions however, with differences in the number of happy vertices usually fewer than three. The exception to these patterns occurs with $k = 50$ and low proportions of precoloured vertices, where TABU+ consistently produces superior results. Indeed, for mean degrees of greater than 40 these differences in the number of happy vertices were often seen to be more than twenty, again highlighting the difficulties that the IP optimiser experiences with these particular problems.

5.5 Influence of Problem Size

Our next set of experiments compares the performance of the IP optimiser and TABU+ on problem instances with varying numbers of vertices. Results are summarised in Table 5 where we consider random graphs with

[4]Defined in Section 5.2.

	Proportion of Vertices Precoloured					
Mean Deg.	0.01	0.03	0.05	0.1	0.2	0.3
$k = 3$						
1	0.00 (0.00)	0.00 (0.00)	0.00 (0.00)	0.00 (0.00)	0.00 (0.00)	0.00 (0.05)
2	0.00 (0.00)	0.00 (0.15)	0.00 (0.25)	0.00 (0.65)	0.00 (0.75)	0.00 (1.00)
3	0.00 (0.00)	0.00 (0.00)	0.00 (0.10)	0.00 (0.25)	0.00 (0.80)	0.00 (0.95)
4	0.00 (0.00)	0.00 (0.10)	0.00 (0.00)	0.00 (0.00)	0.00 (0.45)	0.00 (0.80)
5	0.00 (0.00)	0.00 (0.00)	0.00 (0.00)	0.00 (0.05)	0.00 (0.45)	0.00 (0.50)
6	0.00 (0.00)	0.00 (0.00)	0.00 (0.00)	0.00 (0.20)	0.00 (0.20)	0.00 (0.35)
7	0.00 (0.00)	0.00 (0.00)	0.00 (0.00)	0.00 (0.05)	0.00 (0.25)	0.00 (0.20)
8	0.00 (0.00)	0.00 (0.00)	0.00 (0.00)	0.00 (0.00)	0.00 (0.10)	0.00 (0.15)
9	0.00 (0.00)	0.00 (0.00)	0.00 (0.00)	0.00 (0.00)	0.00 (0.30)	0.00 (0.00)
10	0.00 (0.00)	0.00 (0.00)	0.00 (0.00)	0.00 (0.00)	0.00 (0.15)	0.00 (0.00)
≥ 20	0.00 (0.00)	0.00 (0.00)	0.00 (0.00)	0.00 (0.00)	0.00 (0.00)	
$k = 10$						
1	0.00 (0.00)	0.00 (0.00)	0.00 (0.00)	0.00 (0.05)	0.00 (0.20)	0.00 (0.05)
2	0.00 (0.10)	0.00 (0.10)	0.00 (0.40)	0.00 (1.00)	0.00 (0.90)	0.00 (0.80)
3	0.00 (0.00)	0.00 (0.00)	0.00 (0.20)	0.00 (1.00)	0.00 (1.00)	0.00 (0.95)
4	0.00 (0.00)	0.00 (0.00)	0.00 (0.20)	0.00 (0.60)	0.00 (1.00)	0.00 (0.85)
5	0.00 (0.00)	0.00 (0.00)	0.00 (0.00)	0.05 (0.15)	0.00 (1.00)	0.00 (0.45)
6	0.00 (0.00)	0.00 (0.00)	0.00 (0.10)	0.00 (0.20)	0.00 (1.00)	0.00 (0.25)
7	0.00 (0.00)	0.00 (0.00)	0.00 (0.00)	0.00 (0.05)	0.00 (0.95)	0.00 (0.05)
8	0.00 (0.00)	0.00 (0.00)	0.00 (0.00)	0.00 (0.10)	0.00 (0.90)	0.00 (0.00)
9	0.00 (0.00)	0.00 (0.00)	0.00 (0.00)	0.00 (0.10)	0.00 (0.60)	0.00 (0.00)
10	0.00 (0.00)	0.00 (0.00)	0.00 (0.00)	0.00 (0.10)	0.00 (0.45)	0.00 (0.00)
20	0.00 (0.00)	0.00 (0.00)	0.00 (0.00)	0.00 (0.55)	0.00 (0.00)	
30	0.00 (0.00)	0.00 (0.00)	0.00 (0.00)	0.00 (0.40)		
≥ 40	0.00 (0.00)	0.00 (0.00)	0.00 (0.00)	0.00 (0.00)		
$k = 50$						
1			0.00 (0.00)	0.00 (0.10)	0.00 (0.35)	0.00 (0.20)
2			0.70 (0.10)	0.10 (0.75)	0.00 (1.00)	0.00 (0.90)
3			0.95 (0.00)	0.05 (0.95)	0.00 (1.00)	0.00 (0.90)
4			0.95 (0.00)	0.10 (0.65)	0.00 (1.00)	0.00 (0.85)
5			1.00 (0.00)	0.20 (0.40)	0.00 (1.00)	0.00 (0.60)
6			0.85 (0.00)	0.05 (0.55)	0.00 (1.00)	0.00 (0.20)
7			0.90 (0.00)	0.10 (0.25)	0.00 (0.95)	0.00 (0.25)
8			0.75 (0.00)	0.15 (0.20)	0.00 (0.95)	0.00 (0.10)
9			0.75 (0.00)	0.30 (0.25)	0.00 (0.80)	0.00 (0.05)
10			0.60 (0.00)	0.15 (0.05)	0.00 (0.60)	0.00 (0.00)
20			0.05 (0.00)	0.35 (0.55)	0.00 (0.00)	
30			0.00 (0.00)	0.00 (0.65)		
40			0.85 (0.00)	1.00 (0.00)		
≥ 50			1.00 (0.00)	1.00 (0.00)		

Table 4: Comparison of the IP and TABU+ algorithms with random graphs. Figures in the table give the proportion of problem instances where TABU+ returned a better solution and, in parenthesis, the proportion where the IP optimiser gave better solutions. All figures are taken from runs across twenty problem instances.

mean degrees of five in which 10% of vertices have been precoloured. These can be considered difficult-to-solve instances according to our results from Section 5.3. As previously, CPU time limits of 600 seconds are used.

For lower values of k and n we see that the IP optimiser has high success rates and low computation times. The solutions of TABU+ are quite close in terms of the number of happy vertices, but are consistently inferior in quality. On the other hand, the advantages of TABU+ become apparent for higher values of k and n, indicating that the IP optimiser is struggling with larger instances of this problem. The asterisks in Table 5 also highlight the places where, with all instances, the IP optimiser was even unable to complete its pre-solve routines within the time limit. The results recorded here therefore correspond to the lower bound given by our initial solution (Section 4.1) and the upper bound of $\bar{H}(G)$ (Section 3.3). As can be seen in the table, these results are substantially worse than those of TABU+.

5.6 Choice of Precoloured Vertices

In our previous experiments, problem instances have been generated by precolouring randomly selected vertices. However, we might also ask whether biasing these choices towards certain vertices will influence the types of solution that are produced. In particular, if only the low-degree vertices are precoloured in a graph, we should expect solutions to contain fewer uncoloured vertices overall because there will be fewer paths between precoloured vertices. Table 6 summarises some results in this regard for random and scale-free

	IP Optimiser		Tabu+	
n	Num. Happy	SR (Secs) [Gap]	Num. Happy	Secs
$k = 3$				
250	196.20	1.00 (0)	196.20	(0)
500	**385.85**	1.00 (0)	385.70	(0)
1000	**768.65**	1.00 (2)	768.55	(2)
2500	**1903.60**	0.90 (79) [0.004]	1903.45	(30)
5000	**3795.05**	0.70 (228) [0.002]	3794.60	(28)
7500	5674.85	0.15 (431) [0.049]	**5679.85**	(56)
10000	7487.80	0.05 (474) [0.057]	**7496.05**	(66)
$k = 10$				
250	177.15	1.00 (6)	177.15	(22)
500	**354.80**	0.80 (59) [0.012]	354.35	(41)
1000	**703.95**	0.25 (301) [0.010]	703.65	(45)
2500	**1752.70**	0.00 (-) [0.036]	1752.30	(160)
5000	3460.80	0.00 (-) [0.121]	**3478.75**	(79)
7500	5176.20	0.00 (-) [0.123]	**5205.85**	(191)
10000	6816.80	0.00 (-) [0.127]	**6851.35**	(229)
$k = 50$				
500	**335.25**	0.00 (-) [0.022]	334.15	(85)
1000	658.35	0.00 (-) [0.159]*	**678.70**	(29)
2500	1644.80	0.00 (-) [0.160]*	**1692.00**	(85)
5000	3276.85	0.00 (-) [0.164]*	**3374.55**	(147)
7500	4906.95	0.00 (-) [0.165]*	**5057.15**	(174)
10000	6447.35	0.00 (-) [0.171]*	**6645.35**	(249)

Table 5: Number of happy vertices in solutions returned by the IP and Tabu+ algorithms with random graphs of differing values of n and k. For the IP optimiser we also report success rates, CPU times of successful runs, and optimality gaps of unsuccessful runs. For Tabu+, we report the time after which no further improvements were achieved. All figures are means taken across twenty problem instances.

		Choice of Precoloured Vertices		
Graph Type	n	Lowest Degrees	Random	Highest Degrees
Random	1000	84.1 ± 0.93	65.5 ± 1.33	54.3 ± 0.80
	5000	82.8 ± 0.43	64.0 ± 0.40	52.6 ± 0.35
	10000	82.8 ± 0.31	63.9 ± 0.43	52.3 ± 0.28
Scale-free	1000	81.3 ± 0.50	75.9 ± 1.28	56.9 ± 1.79
	5000	83.2 ± 0.36	77.5 ± 0.58	55.1 ± 2.01
	10000	87.3 ± 0.15	80.4 ± 0.37	54.1 ± 2.02

Table 6: Percentage of vertices seen to be happy in solutions returned by Tabu+ after 600 seconds using graphs of differing topologies, methods for selecting the precoloured vertices, and values of n. All figures are means taken across twenty problem instances using $k = 10$, 10% of precoloured vertices, and an average vertex degree of six.

graphs (regular graphs are not considered here since, by definition, all vertices have the same degree). In these cases, instances were generated by simply ordering vertices by degree before randomly assigning them to colours as before.

The results in Table 6 confirm that, for both topologies, precolouring the lowest degree vertices leads to more happy vertices, while precolouring the high-degree vertices leads to fewer. (These differences equate to approximately 30% of vertices in the instances tested). We also see a slight difference in results between random and scale-free graphs when vertices are randomly assigned, with scale-free graphs usually featuring more happy vertices. This is due to the way in which edges are distributed in scale-free graphs, which results in a small number of high-degree vertices, but many more low-degree vertices. Hence, compared to random graphs, a random selection of vertices in a scale-free graph is less likely to contain vertices of high degree.

5.7 Comparison with MNMC Benchmarks

In Section 2.1 we noted the equivalence of the MHIS problem and the minimum node multi-way cut problem. This allows the opportunity of comparing our results to those reported by Cornaz et al. [10]. Table 7 summarises the random instances used in their paper.[5] These were constructed by creating a random graph of density 39% before adding some terminal vertices together with a small number of edges between these terminals and the original graph. Further details are provided in [10]. As noted in Section 2.1, instances of the

[5]These were provided to us by the authors of [10] on request

Instance	n	Number of terminals	Sum of terminal degrees	Maximum degree among terminals	m	Optimal (from [10])	Lower bound ($n - \bar{H}(G)$)	Our results ($n - f(\mathcal{S})$)
R_50	57	7	22	4	511	18	10	18
R_70	77	7	23	4	993	19	11	19
R_100	107	7	22	4	1985	18	11	18
R_300	307	7	23	4	17,792	19	11	19
R_600	607	7	23	4	70,673	19	11	19
R_700	707	7	22	4	96,432	18	11	18
R_800	810	10	51	7	125,978	44	25	44
R_900	910	10	58	7	159,331	51	28	51
R_1000	1010	10	54	9	196,771	45	27	45
R_1200	1210	10	51	7	283,386	44	25	44
R_1300	1310	10	51	7	332,861	44	25	44
R_1500	1510	10	54	7	442,444	47	26	47
R_1800	1815	15	86	8	637,307	78	43	78
R_2000	2015	15	78	10	786,639	68	39	68
R_2100	2115	15	90	8	867,430	82	44	82
R_2300	2315	15	78	8	1,041,158	70	39	70
R_3000	3015	15	85	10	1,770,773	75	42	75
R_3300	3315	15	88	8	2,142,370	80	43	80
R_3800	3815	15	80	8	2,841,805	72	40	72
R_4200	4215	15	89	8	3,472,117	81	44	81
R_4500	4515	15	89	8	3,987,339	81	44	81
R_4800	4815	15	82	8	4,537,794	74	41	74
R_5000	5015	15	74	7	4,922,710	67	37	67

Table 7: Comparison of results using the random problem instances of [10].

MNMC problem can be converted into an equivalent MHIS problem by simply precolouring each terminal vertex with a different colour. The optimal (minimum) cut size is then calculated using $n - H(G)^*$.

Table 7 shows the quality of solutions achieved by TABU with these instances. As shown, they are optimal in all cases. Surprisingly, these values were actually determined using TABU's initial solutions—i.e., solutions produced using our heuristic from Section 4.1, where all appropriate free vertices are assigned to the same colour. From the perspective of the MNMC problem, this reveals that optimal solutions to these instances are those in which the cut vertices are simply the neighbours of all terminals except the terminal with the highest degree. This fact can also be seen by noting that, for each instance in the table, the optimal value is equal to the sum of the terminal degrees minus the maximum terminal degree.

Although these instances appear to be quite trivial note that, on its own, our heuristic does not prove the optimality of these solutions. Table 7 also reveals that the bound $\bar{H}(G)$ is not particularly accurate in these cases. We also applied our IP optimiser to these instances. Since it is warm-started using our heuristics, the solutions found by the IP optimizer are at least always as good as the heuristic. Furthermore, we find that the IP optimizer does not improve upon any of these solutions (naturally, since the solutions are already optimal); hence, the execution time is spent improving the bounds in order to prove optimality.

6 Conclusions and Further Work

This paper has analysed a number of features of the maximum happy induced subgraph problem, including methods for determining bounds and for breaking up problems. We have seen that difficult-to-solve instances of the MHIS problem seem to occur only when there is a suitable balance between the proportion of pre-coloured vertices and the density of the graph. Instances outside of these phase transition regions tend to be under- or over-constrained and, consequently, are solved quite easily. These patterns have been seen to be fairly consistent across random, scale-free, and d-regular topologies, though the latter do seem to present some extra difficulties for the IP optimiser in and around the regions.

On the whole, the IP optimiser performed very well with our set of 1000-vertex instances, though it does seem to experience difficulties with the instances featuring larger numbers of colours. On the other hand, the advantages of the TABU+ algorithm are apparent with instances with higher numbers of vertices and/or colours. That said, even with its in-built diversification operator our results have also suggested that TABU+ does not seem to make significant gains during the latter stages of runs. In some cases this could be because the algorithm has found an optimal solution but is unable to detect this fact; in other cases it suggests that the algorithm has stagnated in a particular sub-optimal region of the solution space and that further actions may be needed to help reinvigorate the search.

One direction of future research would be to seek improvements to our proposed optimisation methods. On the mathematical programming side, it would be useful to research new IP formulations with the aim

of improving scalability [10]. Decompositions, such as a Lagrangian relaxation might also prove beneficial in some cases. It is also likely that matheuristics such as the Construct, Merge, Solve & Adapt method of Blum et al. [8] might show some promise, particularly for larger problem instances.

Other graph topologies are also of interest. For example, the "small-world" graphs of Watts and Strogatz [28] seek to emulate situations where networks have small average path lengths and local clusters—features that are often seen in real-world social networks. In preliminary experiments with this topology we have found that an increased number of colours generally causes a decrease in the number of happy vertices available in a solution. This matches the results seen with the other topologies. We have also observed that solutions to graphs with higher clustering coefficients tend to feature more happy vertices than those with little clustering, particularly in comparison to random and d-regular graphs. This stands to reason because vertices within each cluster will often be able to assume the same colours as one another, making them happy. Further research in this area is desirable.

Another avenue of research would be to consider versions of this problem in which graphs are known to evolve over time through the addition and deletion of vertices, edges, and/or precolourings. If the amount of change occurring at each time step is known to be quite small, then we may find that a candidate solution produced for a previous time step, with slight modifications, may also be sufficient for the current time step; however, with larger changes other approaches might be necessary. If future changes to a problem instance can be modelled probabilistically, it might also be possible to use this information to produce robust solutions that, with high probability, remain feasible even when these modifications are made. Some relevant work in this area with regards to the graph colouring problem is due to Hardy et al. [17].

It would also be interesting to consider decentralised versions of this problem. As we have seen, the optimisation methods considered here assume that graphs can be stored centrally and that all of its features are visible to all operators. However, this might not always be the case in practice, either because the graph is too large to be stored in one location (as with large social networks such as Facebook), or because the vertices represent independent real-world entities (such as people [18] or radio transmitters [15]) that are responsible for choosing their own colours and that are only aware of features in certain parts of a graph, such as their immediate neighbourhood. Clearly, different optimisation schemes will be required for such problems.

Acknowledgements

This research was partially supported by the Cardiff University International Collaboration Seedcorn Fund. It was also supported in part by the Monash eResearch Centre and eSolutions-Research support services through the use of the MonARCH HPC cluster.

References

[1] Problem generator. `http://www.rhydlewis.eu/resources/happygen.zip`. Accessed: 2020-02-21.

[2] Source code and results data set. `http://www.rhydlewis.eu/resources/MHIS.zip`. Accessed: 2020-02-21.

[3] Supplimentary material. `http://www.rhydlewis.eu/papers/MHISa.pdf`. Accessed: 2020-02-21.

[4] N. Aravind, S. Kalyanasundaram, and A. Swami Kare. Linear time algorithms for happy vertex coloring problems for trees. In V. Mäkinen, S. Puglisi, and L. Salmela, editors, *Combinatorial Algorithms: IWOCA 2016*, volume 9843 of *Lecture Notes in Computer Science*, pages 281–292. Springer Cham., 2016.

[5] N. Aravind, S. Kalyanasundaram, A. Swami Kare, and J. Lauri. Algorithms and hardness results for happy coloring problems. *CoRR*, abs/1705.08282, 2017.

[6] A. Barabási. *Network Science*. Cambridge University Press, 2016.

[7] I. Blöchliger and N. Zufferey. A graph coloring heuristic using partial solutions and a reactive tabu scheme. *Computers and Operations Research*, 35:960–975, 2008.

[8] C. Blum, P. Pinacho, and J. López-Ibáñez, M. amd Lozano. Construct, merge, solve & adapt a new general algorithm for combinatorial optimization. *Computers and Operations Research*, 68:75–88, 2016.

[9] P. Cheeseman, B. Kanefsky, and W. Taylor. Where the really hard problems are. In *Proceedings of the 12th International Joint Conference on Artificial Intelligence - Volume 1*, IJCAI'91, pages 331–337, San Francisco, CA, USA, 1991. Morgan Kaufmann Publishers Inc.

[10] D. Cornaz, Y. Magnouche, A. Mahjoub, and S. Martin. The multi-terminal vertex separator problem: Polyhedral analysis and branch-and-cut. *Discrete Applied Mathematics*, 256:11–37, October 2019.

[11] P. Dev. Homophily and community structure in networks. *Journal of Public Economic Theory*, 18(2):268–290, 2016.

[12] A. E. Eiben, J. K. van der Hauw, and J. I. van Hemert. Graph coloring with adaptive evolutionary algorithms. *Journal of Heuristics*, 4(1):25–46, 1998.

[13] B. Everitt, S. Landau, M. Leese, and D. Stahl. *Cluster Analysis*. John Wiley & Sons, 5th edition, 2011.

[14] Y. Fan and J. Shen. On the phase transitions of random k-constraint satisfaction problems. *Artificial Intelligence*, 175(3):914–927, 2011.

[15] I. Finocchi, A. Panconesi, and R. Silvestri. An experimental analysis of simple, distributed vertex colouring algorithms. *Algorithmica*, 41:1–23, 2005.

[16] N. Garg, V. Vazirani, and M. Yannakakis. Multiway cuts in node weighted graphs. *Journal of Algorithms*, 50:49–61, 2004.

[17] B. Hardy, R. Lewis, and J. Thompson. Tackling the edge dynamic graph colouring problem with and without future adjacency information. *Journal of Heuristics*, 24(3):321–343, 2017.

[18] M. Kearns, S. Suri, and N. Montfort. An experimental study of the coloring problem on human subject networks. *Science*, 313(5778):824–827, 2006.

[19] R. Lewis. Metaheuristics can solve sudoku puzzles. *Journal of heuristics*, 13(4):387–401, 2007.

[20] R. Lewis. *A Guide to Graph Colouring: Algorithms and Applications*. Springer International Publishing, 2016.

[21] R. Lewis and F. Carroll. Creating seating plans: A practical application. *Journal of the Operational Research Society*, 67(11):1353–1362, 2016.

[22] R. Lewis, D. Thiruvady, and K. Morgan. Finding happiness: An analysis of the maximum happy vertices problem. *Computers and Operations Research*, 103:265–276, 2019.

[23] A. Li and P. Zhang. Algorithmic aspects of homophyly of networks. *Theoretical Computer Science*, 593:117–131, 2015.

[24] D. Marx. Parameterized graph separation problems. *Theoretical Computer Science*, 351:394–406, 2006.

[25] P. Ross, D. Corne, and H. Terashima-Marin. The phase-transition niche for evolutionary algorithms in timetabling. In E. Burke and P. Ross, editors, *Practice and Theory of Automated Timetabling (PATAT) I*, volume 1153 of *Lecture Notes in Computer Science*, pages 309–325. Springer-Verlag, Berlin, 1996.

[26] S. Simard, C. Defrenne, and K. Beiler. Talking trees. *National Geographic*, pages 26–27, June 2018.

[27] A. Steger and N. Wormald. Generating random regular graphs quickly. *Combinatorics, Probability and Computing*, 8(4):377–396, July 1999.

[28] D. Watts and S. Strogatz. Collective dynamics of 'small-world' networks. *Nature*, 6684:440–442, 1998.