Creating Seating Plans: A Practical Application

Rhyd Lewis[1] and Fiona Carroll[2]
[1]Cardiff School of Mathematics, Cardiff University, WALES.
[2]Faculty of Computing, Engineering and Science, University of South Wales, WALES.
Email: lewisR9@cf.ac.uk, fiona.carroll@southwales.ac.uk.

April 28, 2016

Abstract

This paper examines the interesting problem of designing seating plans for large events such as weddings and gala dinners where, amongst other things, the aim is to construct solutions where guests are sat on the same tables as friends and family but, perhaps more importantly, are kept away from those they dislike. This problem is seen to be $\mathcal{NP}$-complete from a number of different perspectives. We describe the problem model and heuristic algorithm that is used on the commercial website www.weddingseatplanner.com. We present results on the performance of this algorithm, demonstrating the factors that can influence run time and solution quality, and also present a comparison with an equivalent IP model used in conjunction with a commercial solver.

Keywords: Seating Plans; Graph Colouring; Combinatorial Optimisation; Metaheuristics; Integer Programming.

1 Introduction

Consider an event such as a wedding or gala dinner where, as part of the formalities, the n guests need to be divided on to $k \leq n$ dining tables. To ensure that guests are sat at tables with appropriate company, it is often necessary for organisers to specify a seating plan, taking into account the following sorts of factors:

- Guests belonging to groups, such as couples and families, should be sat at the same tables, preferably next to each other.
- If there is any perceived animosity between different guests, these should be sat on different tables. Similarly, if guests are known to like one another, it may be desirable for them to be sat at the same table.
- Some guests might be required to sit at a particular table. Also, some guests might be prohibited from sitting at other tables.
- Since tables may vary in size and shape, each table should be assigned a suitable number of guests, and these guests should be arranged around the table in an appropriate manner.

A naive method for producing a seating plan best fitting these sorts of criteria might be to consider all possible plans and then choose the one perceived to be the most suitable. However, for non-trivial values of n or k, the number of possible solutions is in fact prohibitively large for this to be possible. To illustrate, consider a simple example where we have 48 guests using six tables, with exactly eight guests per table. For the first table we need to choose eight people from the 48, for which there are $\binom{48}{8} = 377,348,994$ possible choices. For the next table, we then choose eight further people from the remaining 40, giving $\binom{40}{8} = 76,904,685$

further choices, and so on. Assuming that n is a multiple of k (allowing equal sized tables), the number of possible plans is thus calculated:

$$\begin{aligned}
&\prod_{i=0}^{k-2} \binom{n-\frac{in}{k}}{n/k} \\
&= \frac{n!}{\left(\frac{n}{k}\right)!\left(n-\frac{n}{k}\right)!} \cdot \frac{\left(n-\frac{n}{k}\right)!}{\left(\frac{n}{k}\right)!\left(n-\frac{2n}{k}\right)!} \cdot \frac{\left(n-\frac{2n}{k}\right)!}{\left(\frac{n}{k}\right)!\left(n-\frac{3n}{k}\right)!} \cdot \ldots \cdot \frac{\left(n-\frac{(k-2)n}{k}\right)!}{\left(\frac{n}{k}\right)!\left(n-\frac{(k-1)n}{k}\right)!} \\
&= \frac{n!}{\left(\left(\frac{n}{k}\right)!\right)^{k-1}\left(n-\frac{(k-1)n}{k}\right)!} \\
&= \frac{n!}{\left(\left(\frac{n}{k}\right)!\right)^{k}} \qquad (1)
\end{aligned}$$

This function clearly features a growth rate that is subject to a combinatorial explosion—even for the modestly sized example above, the number of distinct solutions to check is approximately 2.9×10^{33}, which is far beyond the capabilities of any state of the art computing equipment. Furthermore, if we were to relax the problem by allowing tables of any size, the task would now be to partition the n guests into k non-empty subsets, meaning that the number of solutions would be equal to a Stirling number of the second kind $S(n, k)$. These numbers feature even higher growth rates than Equation (1), giving even larger solution spaces.

Such arguments demonstrate that this sort of naive method for producing a desirable seating plans is clearly infeasible in most cases. However, the problem of constructing seating plans is certainly important since (a) it is regularly encountered by event organisers and (b) the quality of the proposed solution could have a significant effect on the success (or failure) of the gathering.

Currently there is a small amount of commercial software available for constructing seating plans [1, 2, 3]. The first of these, for example, allows users to input a list of guest names and then specify preferences between these guests (such as whether they need to be sat apart or together). It then allows users to define table shapes, sizes and locations, before assisting the user in placing the guests at these tables via drag and drop functionality and also an auto assign tool. The details of the underlying algorithm used with the auto assign tool are commercially sensitive, though the software's documentation states that a genetic algorithm is used, with different penalty costs being applied for different types of constraint violation. The fitness function of the algorithm is simply an aggregate of these penalties.

In its simplest form the problem of constructing a seating plan might be defined using an $n \times n$ binary matrix $\mathbf{W}$, where element $W_{ij} = 1$ if guests i and j are required to be sat at different tables and $W_{ij} = 0$ otherwise. We can also assume that $W_{ij} = W_{ji}$. Given this input matrix, our task might then be to partition the n guests into k subsets $\mathcal{S} = \{S_1, \ldots, S_k\}$, such that the objective function:

$$f(\mathcal{S}) = \sum_{t=1}^{k} \sum_{\forall i,j \in S_t : i<j} W_{ij} \qquad (2)$$

is minimised. The problem of confirming the existence of a zero cost solution to this problem is clearly equivalent to the $\mathcal{NP}$-complete graph k-colourability problem [8, 10], in which each guest corresponds to a vertex, and two vertices v_i and v_j are considered adjacent if and only if $W_{ij} = 1$. This $\mathcal{NP}$-completeness obviously suggests that there exists no polynomially bounded algorithm for exactly solving all instances of the problem [8].

From an alternative perspective, consider the situation where we are again given the binary matrix $\mathbf{W}$, and where we are now given a subset S of guests that have been assigned to a particular circular shaped table. Here, we might be interested in arranging the guests onto the table such that, for all pairs of guests $i, j \in S$, if $W_{ij} = 1$ then i and j are not sat in adjacent seats. This problem can also be described by a graph $G = (V, E)$ for which the vertex set $V = \{v_i : i \in S\}$ and the edge set $E = \{\{v_i, v_j\} : W_{ij} = 1 \ \wedge \ v_i, v_j \in V\}$. A Hamiltonian cycle of the compliment graph $\bar{G}$ defines a seating arrangement satisfying this criterion, as illustrated in the example in Figure 1. However, determining the existence of a Hamiltonian cycle in an arbitrary graph is also an $\mathcal{NP}$-complete problem.

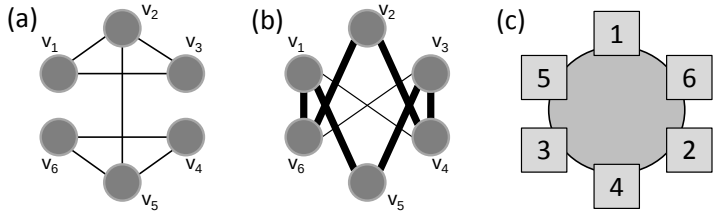

Figure 1: (a) A graph G in which edges specify pairs of guests who should not be sat in adjacent seats; (b) the compliment graph $\bar{G}$, together with a Hamiltonain cycle (shown in bold); and (c) the corresponding seating arrangement around a circular table.

In practical situations it might be preferable for $\mathbf{W}$ to be an integer or real-valued matrix instead of binary, allowing users to place greater importance on some of their seating preferences compared to others. Assuming lower values for W_{ij} indicate an increased preference for guests i and j to be sat together, the problem of partitioning the groups on to k tables now becomes equivalent to the edge-weighted graph colouring problem, while the task of arranging people on to circular tables in the manner described above becomes equivalent to the travelling salesman problem. Of course, both of these problems are also $\mathcal{NP}$-complete since they generalise the graph k-colorability problem and the Hamiltonian cycle problem respectively [8]. In addition, the problem of arranging guests around tables can become even more complicated when tables of different shapes are used. For example, with rectangular tables we might also need to take account of who is sat opposite a guest in addition to their neighbours on either side.

At the time of writing, there seems to be very little research into the problem of constructing seating plans. In fact, we have come across just one paper on the subject written in 2012 by Bellows and Luc Peterson [4] who proposed an approach using IP methods. In their case, the authors made use of an $n \times n$ integer matrix as described above. They also specified a limit on the number of guests allowed per table, and stipulated a parameter γ such that a solution was considered feasible only if each guest was friends with at least γ guests on his or her table. In their paper, the authors then went on to consider a practical problem involving 107 guests and 11 tables (actually from their own wedding day!) and ran this on the commercial solver CPLEX for 36 hours. A feasible solution to their problem was produced in this time, but no certificate of optimality was obtained by the software.

In this current paper we also focus our attention on creating seating plans for wedding parties making use of the problem model used with our commercial website `www.weddingseatplanner.com`. This website contains a free tool, embedded within a webpage, for inputting and solving instances of the problem. It is stated by Nielsen [13] that users tend to leave a website in less than two minutes if it is not understood or perceived to fulfil their needs. Consequently, our particular problem model is designed to strike what we feel to be the right balance between being quickly accessible to users, while still being useful in practice. This means that some of the issues discussed above are deliberately left out. The main focus of this paper is to describe the heuristic algorithm that is used for calculating seating plans based on the users' input. Since users will typically have little knowledge of optimisation algorithms and the implications of $\mathcal{NP}$-hardness, this algorithm is specifically designed to supply the user with high-quality (though not necessarily optimal) solutions in very short amounts of run time (typically less than three seconds). In particular, our approach seeks to exploit the underlying graph-based structures of this problem, encouraging an effective navigation of the solution space via specialised neighbourhood operators.

The next section of this paper contains a formal definition of our problem model, and also demonstrates its underlying complexity. Section 3 then describes our algorithm and online tool in detail, before looking at its salient run characteristics in Section 4. Section 5 then examines more closely the speed and quality of solutions produced by our algorithm by comparing against a commercial solver using an integer programming (IP) formulation of the same problem. Finally, Section 6 concludes the paper and provides a discussion into how our model might be extended to incorporate further constraints and deal with other types of event.

2 Problem Description

In our definition of this problem, we choose to first partition the guests into a number of *guest groups*. Each guest group refers to a subset of guests who are required to sit together (couples, families with young children, etc.) and will usually be known beforehand by the user. In addition to making the problem smaller, specifying guest groups in this way also means that users do not have to subsequently input preferences between pairs of people in the same families etc. to ensure that they are sat together in a solution.

Having done this, the Wedding Seating Problem (WSP) can now be formally stated as type of graph partitioning problem. Specifically, we are given a graph $G = (V, E)$ comprising a vertex set V and edge set E. Each vertex $v \in V$ represents a guest group, with the size of each guest group denoted s_v. The total number of guests in the problem is thus $n = \sum_{v \in V} s_v$.

In G, each edge $\{u, v\} \in E$ defines the relationship between vertices u and v according to a weighting w_{uv} (where $w_{uv} = w_{vu}$). If $w_{uv} > 0$ this is interpreted to mean that we would prefer the guests associated with vertices u and v to be sat on different tables. Larger values for w_{uv} reflect a strengthening of this requirement. Similarly, negative values for w_{uv} mean that we would rather u and v were assigned to the same table.

A solution to the WSP is consequently defined as a partition of the vertices into k subsets $\mathcal{S} = \{S_1, \ldots, S_k\}$, such that:

$$\bigcup_{i=1}^{k} S_i = V, \text{ and} \tag{3}$$

$$S_i \cap S_j = \emptyset \;\; \forall i, j \in \{1, \ldots, k\},\, i \neq j \tag{4}$$

The requested number of tables k is defined by the user, with each subset S_i defining the guests assigned to a particular table.

Under this basic definition of the problem, the quality of a particular candidate solution might be calculated according to various different metrics. In our case we use two objective functions, both that are to be minimised. The first of these is analogous to Equation (2):

$$f_1 = \sum_{i=1}^{k} \sum_{\forall u,v \in S_i : \{u,v\} \in E} (s_v + s_u) w_{uv} \tag{5}$$

and reflects the extent to which the rules governing who sits with whom are obeyed. In this case the weighting w_{uv} is also multiplied by the total size of the two guest groups involved $s_v + s_u$. This is done so that violations involving larger numbers of people contribute more to the cost (i.e., it is assumed that s_v people have expressed a seating preference in relation to guest group u, and s_u people have expressed a preference in relation to guest group v).

The second objective function used in our model is intended to encourage equal numbers of guests being assigned to each table. In practice some weddings may have varying sized tables, and nearly all weddings will have a special "top table" where the bride, groom and their associates sit. In practice the top table and its guests can be ignored in this particular formulation because they can easily be added to a table plan once the other guests have been arranged. We also choose to assume that the remaining tables are be equi-sized, which seems to be a very common option, particularly for large venues. Consequently, the second objective function measures the degree to which the number of guests per table deviates from the required number of either $\lfloor n/k \rfloor$ or $\lceil n/k \rceil$:

$$f_2 = \sum_{i=1}^{k} \left(\min \left(|\tau_i - \lfloor n/k \rfloor| , |\tau_i - \lceil n/k \rceil| \right) \right). \tag{6}$$

Here $\tau_i = \left(\sum_{\forall v \in S_i} s_v \right)$ denotes the number of guests assigned to each table i. Obviously, if the number of guests n is a multiple of k, then Equation (6) simplifies to $f_2 = \sum_{i=1}^{k} \left(|\tau_i - n/k| \right)$.

2.1 Problem Complexity

It is evident that our problem model is $\mathcal{NP}$-hard since it generalises two classical $\mathcal{NP}$-hard problems:

(a)

	Top table?	Guest name	Companion 1	Companion 2	Companion 3
1	✓	Cath	Michael	Kurt	Rosie
2		John	Sarah	Jack	Jill
3		Bill	June		
4		Pat	Susan		
5		Una	Tom		
6		Ruth	Kevin	Gareth	
7		Ken	Frank	Bobby	
8		Rod	Dereck	Freddy	
9		Jane			

(b)

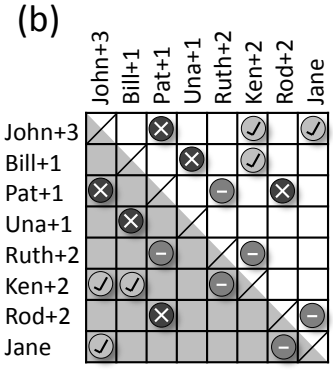

Figure 2: Specification of guest groups (a) and seating preferences (b).

- If $E = \emptyset$ then f_1 (Equation (5)) will always equate to zero. This means that the only goal is to ensure that the number of guests per table is as equal as possible. Hence, the problem reduces to the $\mathcal{NP}$-hard k-partition problem.[1]
- If $s_v = 1\ \forall v \in V$, then all guest groups contain just one person, and the number of guests assigned to a table i will equal the number of vertices in the group $|S_i|$. Hence, the problem will now be equivalent to the $\mathcal{NP}$-hard equitable weighted k-colouring problem, itself a generalisation of the $\mathcal{NP}$-hard equitable graph k-colouring problem [8].

3 Problem Interpretation and Tabu Search Algorithm

On entering the website, the user is first asked to input (or import) the names of all guests into an embedded interactive table. Guest groups that are to be seated together (families etc.) are placed on the same rows of the table, thus defining the various values for s_v. Guests to be sat at the top table are also specified. At the next step the user is then asked to define seating preferences between different guest groups. Since guests to be sat at the top table have already been given, constraints only need to be considered between the remaining guest groups (guests at the top table are essentially ignored from this point onwards).

Figure 2 shows a small example of this process. Here, nine guest groups ranging in size from 1 to 4 have been input, though one group of four has been allocated to the top table. Consequently, only the remaining eight groups ($n = 20$ guests) are considered. The right-hand grid then shows the way in which the seating preferences (values for w_{uv}) are defined between these. On the website this is done interactively by clicking on the relevant cells in the grid. In our case, users are limited to three options: (1) "Definitely Apart" (e.g., Pat and John); (2) "Rather Apart" (Pat and Ruth); and (3) "Rather Together" (John and Ken). These are allocated weights of ∞, 1, and -1 respectively for reasons that will be made clear below. Note that it would have been possible to allow the user to input their own arbitrary weights here; however, while being more flexible, it was felt by the website's interface designers that this ran the risk of bamboozling the user while not improving the effectiveness of the tool [6].

Once the input to the problem has been defined by the user, the overall strategy of our algorithm is to classify the requirements to the problem as either hard (mandatory) constraints or soft (optional) constraints. In our case we consider just one hard constraint, which we attempt to satisfy in Stage 1—specifically the constraint that all pairs of guest groups required to be "Definitely Apart" are assigned to different tables. In Stage 2, the algorithm then attempts to reduce the number of violations of the remaining constraints via specialised neighbourhood operators that do not allow any of the hard constraints satisfied in Stage 1 to be re-violated. The two stages of the algorithm are now described in more detail.

[1] Note that the k-partition problem is also variously known as the load balancing problem, the equal piles problem, or the multiprocessor scheduling problem.

3.1 Stage 1

In Stage 1 the algorithm operates on the sub-graph $G' = (V, E')$, where each vertex $v \in V$ represents a guest group, and the edge set $E' = \{\{u, v\} \in E : w_{uv} = \infty\}$. That is, G' contains only those edges from E that define the "Definitely Apart" requirement.[2] Using this sub-graph, the problem of assigning all guests to k tables (while not violating the "Definitely Apart" constraint) is equivalent to the graph k-colouring problem, as noted in Section 1.

The graph k-colouring problem is a widely studied combinatorial optimisation problem for which a multitude of different approximation algorithms have been proposed [10]. In our case, we use a variant of the DSATUR heuristic [5] to produce an initial solution. DSATUR is a constructive algorithm that seeks to colour the most constrained vertices first. Specifically, at each iteration DSATUR selects the uncoloured vertex v that currently has the largest number of distinct colours assigned to adjacent vertices. Any ties are broken by selecting the vertex amongst these with the largest degree. In our case the selected vertex v is then assigned to the colour $i \in \{1, \ldots, k\}$ that (a) currently features no other vertices adjacent to v, and (b) contains the fewest vertices, encouraging an equitable spread of the vertices amongst the colours. If no colour satisfying (a) exists, then v is kept to one side and is assigned to a random colour at the end of the construction process, thereby introducing hard constraint violations.

If the solution produced by the above constructive process contains hard constraint violations, an attempt is then made to eliminate these. This is achieved using the TABUCOL algorithm of Hertz and de Werra [9, 7] which, while perhaps not as powerful as more contemporary graph colouring algorithms, does have the advantage of being very fast (see [10] for further details). In essence, TABUCOL uses the tabu search metaheuristic to make a series of small changes (moves) to the candidate solution, attempting to find a solution for which the cost function $f_1 = 0$ (using G'). A move in the search space is performed via a neighbourhood operator that takes a vertex v whose assignment to colour i is currently contributing to the cost, and then assigns it to a new colour $j \neq i$. The tabu list is stored in a matrix $\mathbf{T}_{|V| \times k}$ and, upon performing this move, the element T_{vi} is marked as tabu for the next t iterations (meaning that v cannot be transferred back to colour i during this time). In each iteration of TABUCOL, all possible moves from the current solution are considered, and the one that is chosen is the move seen to invoke the largest decrease (or failing that, the smallest increase) in cost of any non-tabu move. Ties are broken randomly and, as an aspiration criterion, tabu moves are also permitted if they are seen to improve on the best solution observed so far by the algorithm.

Because speed is a major issue with our online tool, TABUCOL is only run for a fixed number of iterations.[3] In the online tool, if at the end of the process a solution with cost $f_1 = 0$ (using G') has not been achieved, k is incremented by one, and Stage 1 of the algorithm is repeated. This situation might occur because the user has specified a k-value for which no k-colouring exists (that is, k is smaller that the chromatic number of G'); however, it might also be the case that feasible solutions *do* exist, but that the algorithm has been unable to find one within the given computation limit. This process of incrementing k and reapplying DSATUR and TABUCOL continues until all of the hard constraints have been satisfied, resulting in a proper k-colouring of G'.

3.2 Stage 2

At the start of Stage 2 we will have achieved a k-colouring of $G' = (V, E')$ for which no pair of adjacent vertices is assigned the same colour. We call such a solution *feasible* because it obeys all of the imposed hard constraints of the problem. In the second stage of this algorithm we now try to eliminate violations of the soft constraints by exploring the space of feasible solutions: that is, we will attempt to make alterations to our seating plan in such a way that no violations of the "Definitely Apart" constraint are reintroduced. Note that movements in this space might be restricted—indeed the space might even be disconnected—and so it is necessary to use neighbourhood operators that provide as much solution space connectivity as possible. In our case this is achieved using tabu search in conjunction with two neighbourhood operators, both that attempt to exploit the structure of the underlying graph colouring problem. These operators are defined as follows:

[2] For example, using the problem shown in Figure 2, $V = \{v_2, v_3, \ldots, v_9\}$ and $E' = \{\{v_2, v_4\}, \{v_3, v_5\}, \{v_4, v_8\}\}$.

[3] Specifically, TABUCOL is executed for $20n$ iterations, using a tabu tenure t proportional to the current cost ($t = 0.6f_1 + r$, where r is randomly selected from the set $\{1, 2, \ldots, 9\}$), as recommended in [7]).

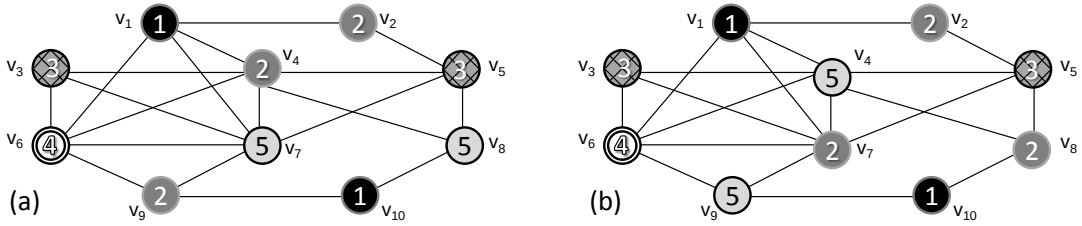

Figure 3: Two examples of a feasible five-colouring of graph $G' = (V, E')$.

Kempe-Chain Interchange: Given an arbitrary vertex v currently assigned to colour i, and given a second colour $j \neq i$, a Kempe-chain is defined as a connected subgraph in G' that contains v and that only comprises vertices coloured with i and j. Such a chain can be denoted KEMPE(v, i, j).

A Kempe-chain interchange involves taking a particular Kempe-chain and swapping the colours of all vertices contained within it. For example, in Figure 3(a), KEMPE$(v_7, 5, 2)$ results in a chain involving vertices $\{v_4, v_9\}$ (assigned to colour 2) and $\{v_7, v_8\}$ (assigned to colour 5). The colours of these vertices are then interchanged, as with Figure 3(b). With regards to seating plans, a Kempe-chain move therefore involves interchanging two sets of guest groups between their respective tables.

It is worth noting that, given a feasible solution $\mathcal{S} = \{S_1, \ldots, S_k\}$, the application of a Kempe-chain interchange will always result in a new solution $\mathcal{S}'$ that is also feasible. To prove this, consider the contrary situation in which $\mathcal{S}$ is feasible but where $\mathcal{S}'$ is not. Since a Kempe-chain interchange involves two colours i and j, this means $\mathcal{S}'$ features a pair of adjacent vertices u and v that are assigned the same colour. Without loss of generality, assume this to be colour i. Moreover, u and v must have both been in the Kempe chain used for the interchange since they are adjacent. This now implies that u and v were both assigned to colour j in $\mathcal{S}$, which is impossible since $\mathcal{S}$ is known to be feasible. ■

It is also worth noting that Kempe-chains can vary in size (in terms of the number of vertices they contain), and and that some combinations for KEMPE(v, i, j) will *overlap* in that they result in the same Kempe-chain being identified (e.g., KEMPE$(v_7, 5, 2)$ = KEMPE$(v_9, 2, 5)$ in Figure 3). Indeed, as the number of colours k is reduced, or the density of G' is increased, then so will the size of the chains and the amount of overlap. This feature is relevant with applications of tabu-search such as ours because, with appropriate book-keeping, we can avoid evaluating the effects of a particular interchange more than once when scanning the entire neighbourhood in each iteration.

Swaps: The swap operator is used for performing further moves that are not contained within the Kempe-chain neighbourhood. Specifically, when scanning the set of Kempe-chain interchanges, we might also identify pairs of non-adjacent vertices u, v that both feature Kempe-chains KEMPE(u, i, j) and KEMPE(v, j, i) that are of size 1 (see, for example KEMPE$(v_1, 1, 3)$ and KEMPE$(v_5, 3, 1)$ in Figure. 3(a).) In these particular cases u and v can have their colours swapped, but no hard constraint violations will occur as a result.

In each iteration of Stage 2, all possible moves from both of the above neighbourhoods are evaluated and the same acceptance criteria as Stage 1 are applied. Once a move is performed, all relevant parts of the tabu list **T** are updated to reflect the changes made to the solution.[4]

The cost function used in this stage is simply $(f_1 + f_2)$. This will always evaluate to a value less than ∞ since violations of the hard constraints cannot occur. Although such an aggregate function is not wholly ideal (because it involves adding together two different forms of measurement) it is considered acceptable in our case because in some sense both metrics relate to the number of people effected by the violations—that is, a table that is considered too have x too many (or too few) people will garner the same penalty cost as x violations of the "Rather Apart" constraint.

[4]That is, all vertices and colours involved in the move are marked as tabu in **T**. For speed's sake, in our application a fixed-size tabu tenure of 10 is used along with an iteration limit of $10n$.

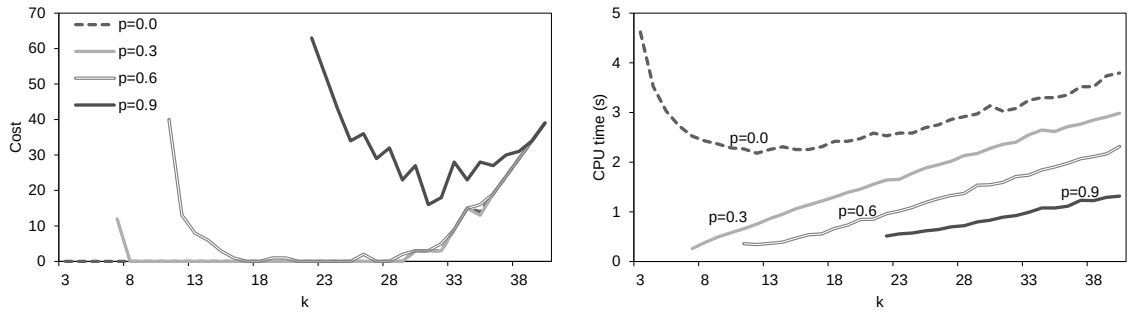

Figure 4: Solution costs (left) and run times (right) for four instances using various k-values. For most of the left figure, the line for $p = 0.0$ is obscured by the line for $p = 0.3$ (i.e., the same results were achieved).

4 Algorithm Performance

The algorithm and interface described above has been implemented in ActionScript 3.0 and can be accessed at `www.weddingseatplanner.com` (note that an installation of Adobe Flash Player is required). The optimisation algorithm is therefore run at the client side. To ensure run times are kept relatively short, and to also allow the interface to be displayed clearly on the screen, problem size has been limited to $|V| = 50$ guest groups of up to 8 people, allowing a maximum of $n = 400$ guests.

To gain an understanding of the performance characteristics of this algorithm, a set of maximum-sized problem instances of $|V| = 50$ guest groups were constructed, with the size of each group chosen uniform randomly in the range 1 to 8 giving $n \approx 50 \times 4.5 = 225$. These instances were then modified such that a each pair of vertices was joined by an ∞-weighted edge with probability p, meaning that a proportion of approximately p guest group pairs would be required to be "Definitely Apart". Tests were then carried out using values of $p = \{0.0, 0.3, 0.6, 0.9\}$ with numbers of tables $k = \{3, 4, \ldots, 40\}$.

Figure 4 shows the results of these tests with regard to the costs and run times that were achieved by our algorithm at termination. Note that for $p \geq 0.3$, values are not reported for the lowest k's because feasible k-colourings were not achieved (possibly because they do not exist). Also note that all runs took less than five seconds on a fairly standard desktop computer (all results reported in this paper were achieved using a 3.0 GHz Windows 7 machine with 3.18 GB RAM).

Figure 4 (left) shows that, with no hard constraints ($p = 0.0$), balanced table sizes have been achieved with all k-values up to 30. Beyond this point, however, it seems there are simply too many tables (and too few guests per table) to spread the groups evenly. Higher costs are also incurred for larger values of p because, in these cases, many guest group combinations (including many of those required for achieving low cost solutions) will now contain at least one hard constraint violation, meaning they cannot be assigned to the same table. That said, in these trials similar solutions have been achieved for $p = 0.0$, 0.3, and 0.6 for various values of k, suggesting that the cost of the best solutions found is not unduly affected by the presence of moderate levels of hard constraints. The exception to this pattern, as shown in Figure 4 (left), is for the lowest achievable values for k. Here, the larger number of guest groups per table makes it more likely that any particular combination will be deemed infeasible, reducing the number of possible feasible solutions, and making the presence of a zero-cost solution less likely.

We also note that for more constrained problems (lower k's and/or larger p's), the Kempe-chains in the underlying graph colouring model tend to become larger and less numerous (i.e., there is more overlapping). In practice this means that the size of the neighbourhoods scanned in each iteration of tabu search is reduced, resulting in shorter run times as demonstrated in Figure 4 (right). The exception to this pattern is for low values of k using $p = 0.0$, where the larger numbers of guests per table requires more overheads in the calculation of Kempe chains and the cost function, resulting in an increase in run times.

5 Comparison to an IP Model

To further analyse the performance of our algorithm in terms of both solution quality and run time, we also choose to compare our results to those achieved by a commercial solver (using a branch and bound framework) applied to an equivalent integer programming (IP) version of this problem. One of the advantages of an IP approach is that, given excess time, we can determine with certainty the optimal solution or, indeed, whether a feasible solution actually exists to the problem. In contrast to our tabu search-based method, the IP solver is therefore able to provide the user with a certificate of optimality and/or infeasibility, at which point it can be halted. Of course, due to the underlying complexity of the WSP these certificates will not always be produced in reasonable time, but given the relatively small problem sizes being considered here, it is still pertinent to ask how often this is the case, and to also compare the quality of the IP solver's solutions to our tabu search approach under similar time limits.

The WSP can formulated as an IP problem as follows. Recall that there are $|V|$ guest groups that we seek to partition on to k tables. Accordingly, the seating preferences of guests can be expressed using a symmetric $|V| \times |V|$ matrix $\mathbf{W}$, where:

$$W_{ij} = \begin{cases} \infty & \text{if we require guest groups } i \text{ and } j \text{ to be "definitely apart";} \\ 1 & \text{if we would prefer } i \text{ and } j \text{ to be on different tables ("rather apart");} \\ -1 & \text{if we would prefer } i \text{ and } j \text{ to be on the same table ("rather together");} \\ 0 & \text{otherwise.} \end{cases} \tag{7}$$

As before, we also let s_i define the size of each guest group $i \in \{1, \ldots, |V|\}$. A solution to the problem can then be represented by a $|V| \times |V|$ binary matrix $\mathbf{X}$, where:

$$X_{it} = \begin{cases} 1 & \text{if guest group } i \text{ is assigned to table } t, \\ 0 & \text{otherwise,} \end{cases} \tag{8}$$

and a binary vector $\mathbf{Y}$ of length $|V|$, where:

$$Y_t = \begin{cases} 1 & \text{if at least one guest group is assigned to table } t, \\ 0 & \text{otherwise,} \end{cases} \tag{9}$$

subject to the following constraints:

$$\sum_{t=1}^{|V|} X_{it} = 1, \forall i \in \{1, \ldots, |V|\} \tag{10}$$

$$X_{it} + X_{jt} \leq Y_t, \forall i, j : W_{ij} = \infty, \forall t \in \{1, \ldots, |V|\} \tag{11}$$

$$X_{it} = 0, \forall i \in \{1, \ldots, |V|\}, \forall t \in \{k+1, \ldots, |V|\} \tag{12}$$

$$Y_t = 0, \forall t \in \{k+1, \ldots, |V|\} \tag{13}$$

$$X_{it} = 0, \forall i \in \{1, \ldots, |V|\}, \forall t \in \{i+1, \ldots, |V|\} \tag{14}$$

$$X_{it} \leq \sum_{j=t-1}^{i-1} X_{jt-1}, \forall i \in \{2, \ldots, |V|\}, \forall t \in \{2, \ldots, i-1\}. \tag{15}$$

Here, Constraints (10) to (15) stipulate the hard constraints of the problem as before; hence, a solution satisfying these can be considered feasible in the same sense as defined in Section 3.2. These constraints are essentially an IP formulation of the graph k-colouring problem. Constraint (10) states that each guest group should be assigned to exactly one table, while (11) specifies that no pair of guest groups should be assigned to the same table if they are subject to a "Definitely Apart" constraint, with $Y_t = 1$ when at least one guest group has been assigned to table t. Constraints (12) and (13) then ensure that a maximum of k tables are used. Finally, Constraints (14) and (15) are used to eliminate symmetries in the IP model, as suggested by Méndéz-Diaz and Zabala [12]. Without these, any particular solution using k tables could be expressed in $k!$ ways by simply permuting the first k columns of $\mathbf{X}$, making the solution space far larger than it needs to be. Constraints (14) and (15) deal with this issue by specifying a unique permutation of the first k columns

for each possible feasible solution (that is, the columns are sorted according to the minimally labelled guest group in each table).

As before, the quality of a feasible candidate solution is quantified using the sum of the two previously defined objective functions $(f_1 + f_2)$. For this IP model, f_1 is rewritten:

$$f_1 = \sum_{t=1}^{k} \sum_{i=1}^{|V|-1} \sum_{j=i+1}^{|V|} X_{it} X_{jt} (s_i + s_j) W_{ij} \tag{16}$$

in order to cope with the binary matrix method of solution representation; however, it is equivalent in form to Equation (5). Similarly, f_2 in the IP model is defined in the same manner as Equation (6), except that τ_i is now calculated as $\tau_i = \sum_{j=1}^{|V|} X_{jt} s_j$. Again, this is equivalent to Equation (6).

It is worth noting here that the objective function defined in Equation (16) actually contains a quadratic term, making our proposed mathematical model a binary *quadratic* integer program. Although modern commercial IP solvers such as XPressMP and CPLEX can cope with such formulations, the use of quadratic objective functions is often known to hinder performance. One way to linearise this model is to introduce an additional auxiliary binary variable:

$$Z_{ijt} = \begin{cases} 1 & \text{if guest groups } i \text{ and } j \text{ are both assigned to table } t, \\ 0 & \text{otherwise,} \end{cases} \tag{17}$$

together with the following additional constraints:

$$Z_{ijt} = 0, \forall i \in \{1, \ldots, |V|\}, \forall j \in \{1, \ldots, i\}, \forall t \in \{1, \ldots, k\} \tag{18}$$

$$Z_{ijt} = 0, \forall i \in \{1, \ldots, |V|\}, \forall j \in \{1, \ldots, |V|\}, \forall t \in \{k+1, \ldots, |V|\} \tag{19}$$

$$Z_{ijt} \leq \frac{1}{2}(X_{it} + X_{jt}), \forall i \in \{1, \ldots, |V|-1\}, \forall j \in \{i+1, \ldots, |V|\}, \forall t \in \{1, \ldots, k\} \tag{20}$$

$$Z_{ijt} \geq (X_{it} + X_{jt}) - 1, \forall i \in \{1, \ldots, |V|-1\}, \forall j \in \{i+1, \ldots, |V|\}, \forall t \in \{1, \ldots, k\}. \tag{21}$$

These constraints ensure that the correct values are assigned to the variables Z_{ijt}. They also allow us to restate the objective function f_1 in the linear form:

$$f_1 = \sum_{t=1}^{k} \sum_{i=1}^{|V|-1} \sum_{j=i+1}^{|V|} Z_{ijt} (s_i + s_j) W_{ij}. \tag{22}$$

In our experiments, both IP formulations were tested using the commercial software XPressMP (version 7.7). We repeated the experiments of Section 4 using two time limits: five seconds, which was approximately the longest time required by our tabu search algorithm (see Figure 4); and 600 seconds, to gain a broader view of the IP solver's capabilities with these formulations. Across the 152 combinations of p and k, under the five second limit the linear model produced feasible solutions for just 11 cases compared to the quadratic model's 112. Similarly, the number of cases where certificates of infeasibility were returned were 24 and 26 respectively. The underperformance of the linear model in these cases may well be due to the much larger number of variables and constraints involved, which seems to present difficulties under this very strict time limit. That said, even though the models' results became more similar under the 600 second time limit, the costs returned by the linear model were still consistently higher than the quadratic model's. Consequently, only the results from the quadratic model are considered for the remainder of this section.

The results of the trials are summarised in Figure 5. The circled lines in the left of the graphs indicate values of k where certificates of infeasibility were produced by the IP solver under the two time limits. As might be expected, these certificates are produced for a larger range of k-values when the longer time limit is used; however, for $p \in \{0.3, 0.6\}$ there remain values of k for which feasible solutions have not been produced (by any algorithm) and where certificates of infeasibility have not been supplied. Thus we are none the wiser as to whether feasible solutions exist for these particular k-values. Also note that certificates of *optimality* were not provided by the IP solver in any of the trials conducted.

Figure 5 reveals that, under the five second limit, the IP approach has produced solutions of inferioir quality compared to tabu search in all cases. In addition, the IP method has failed to achieve feasible solutions

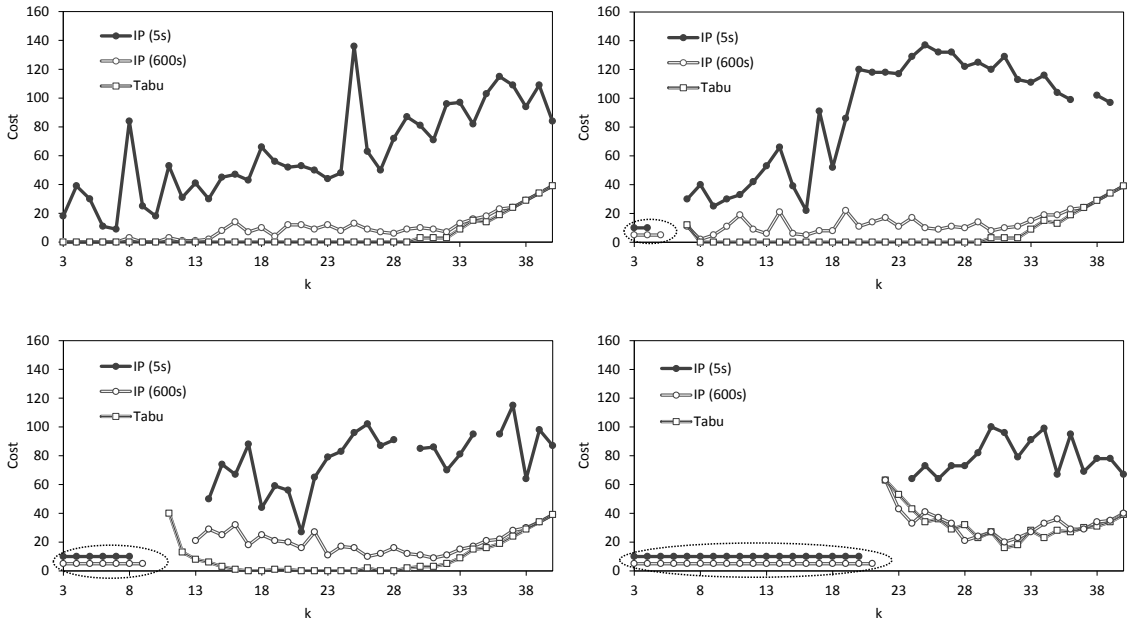

Figure 5: Comparison of solution costs achieved using the IP solver (using 2 different time limits) and our tabu search-based approach for $p = 0.0$ (top-left), $p = 0.3$ (top-right), $p = 0.6$ (bottom-left), and $p = 0.9$ (bottom-right).

in seven of the 121 cases where tabu search has been successful. When the extended run time limit is applied, this gap in performance diminishes, but similar patterns still emerge. We see that the tabu search algorithm has produced feasible solutions whenever the IP approach has, plus three further cases. In addition, in the 119 cases where both algorithms have achieved feasible solutions, tabu search has produced superior quality solutions in 94 cases, compared to the IP method's six. However, we must bear in mind that the IP solver has required more than 400 times the CPU time of tabu search to achieve these particular solutions, making it much less suitable for an online tool.

Finally, for completeness we also chose to re-implement the IP model of Bellows and Luc-Petersen [4] in our experiments. Recall from Section 1 that instead of using guest groups, their method uses the larger matrix $\mathbf{W}_{n\times n}$ in order to specify seating preferences between guests, with elements W_{ij} receiving particularly small values if the associated guests belong to families/couples etc.. However, their use of this larger problem matrix seems to increase run times quite dramatically. Indeed, in our tests using the same problem instances as Sections 4 and 5, no integer solutions were returned within the five second time limit for $p \geq 0.3$.

6 Conclusions and Discussion

This paper has discussed the interesting combinatorial problem of constructing seating plans. We have looked at this problem in broad terms and have then gone on to propose a formulation that generalises both the weighted graph colouring problem and the k-partition problem. An effective two-stage heuristic algorithm making use of concepts taken from the underlying graph colouring problem has been described for this model, and experiments have demonstrated the factors that influence solution quality and run times. The algorithm has also shown to produce solutions that are generally superior to those achieved by an IP approach under similar and extended time limits.

The algorithm and interface described here has been available online at `www.weddingseatplanner.-com` since mid-2011 and currently receives approximately 2,000 hits per month. In addition to weddings, it can also be used in other situations where we need to divide people into groups, such as birthday parties, gala dinners, team building exercises, group projects, and poker tournaments. In the latter example, we are

typically interested in dividing competitors into equal sized groups at the start of a tournament; however, we may also require seeded competitors to be assigned to separate tables. The vertices associated with such competitors would therefore form a clique in the underlying graph colouring model.

As we have noted, our algorithm design choices have been heavily influenced by the need of our online tool to produce good solutions in short amounts of time. If more computing power/time is available, further improvements to Stage 1 might be achieved using some of the more elaborate algorithms for graph colouring appearing in the literature, such as the evolutionary-based methods of Galinier and Hao [7] or Malaguti et al. [11] (see [10] for a survey). On the other hand, other neighbourhood operators or search methodologies (such as simulated annealing or tabu search with diversification techniques) might show promise in Stage 2, or we may see some benefit in choosing not to aggregate the objective functions f_1 and f_2 and instead optimise them separately using multiobjective-based techniques.

As we have seen, our problem formulation has been chosen to try to strike the right balance between being useful to users, while also being easy to understand. As part of this, in our online application we have chosen to allow only three different weights on edges: -1, 1, and ∞, corresponding to the constraints "Rather Together", "Rather Apart", and "Definately Apart" respectively. In practice however, our proposed algorithm could be applied to any set of weights. For example, we might choose to define a threshold value c, and then consider any edge $\{u, v\}$ with weight $w_{uv} \geq c$ as a hard constraint, with the remaining edges then being treated as soft constraints.[5] On the other hand, if it is preferable to treat all constraints as soft constraints, we might simply abandon Stage 1 and use the optimisation process of Stage 2 to search through the solution space comprising all k-partitions of the guest groups.

An additional advantage of our graph colouring-based model is that it can be easily extended to incorporate "table specific" constraints that specify which table each guest group can and cannot be assigned to. To impose such constraints we first need to add k additional vertices to the model, one for each available table. Next, edges of weight ∞ then need to be added between each pair of these "table-vertices", thereby forming a clique of size k and ensuring that each table-vertex is assigned to a different colour in any feasible solution. Having introduced these extra vertices we are now able to introduce other types of constraints into the model:

- If guest group v is not permitted to sit at table i, then an edge is imposed between vertex v and the ith table vertex.
- If a guest group v must be assigned to table i, then edges are imposed between vertex v and all table vertices except the ith table vertex.

Note that if our model is extended in this way, we will now be associating each subset of guest groups S_i in a solution $\mathcal{S} = \{S_1, \ldots, S_k\}$ with a particular table number i. Hence, we might also then permit tables of different sizes and shapes in the model, perhaps also incorporating constraints concerning these factors into the objective function. Further constraints, such as making sure that a pair of guests are "at least x tables apart" might then also be considered [14].

Acknowledgement

The authors would like to thank the anonymous referee whose comments and suggestions helped to improve this manuscript.

References

[1] www.perfecttableplan.com.
[2] www.seatingarrangement.com.
[3] www.toptableplanner.com.
[4] M. Bellows and J. Luc Peterson. Finding an optimal seating chart. *Annals of Improbable Research*, 2012.
[5] D. Brelaz. New methods to color the vertices of a graph. *Commun. ACM*, 22(4):251–256, 1979.
[6] F. Carroll and R. Lewis. The 'engaged' interaction: Important considerations for the hci design and development of a web application for solving a complex combinatorial optimization problem. *World Journal of Computer Application and Technology*, 1(3):75–82, 2013.

[5] That is, the graph G' used in Stages 1 and 2 would comprise edge set $E' = \{\{u, v\} \in E : w_{uv} \geq c\}$.

[7] P. Galinier and J-K. Hao. Hybrid evolutionary algorithms for graph coloring. *Journal of Combinatorial Optimization*, 3:379–397, 1999.

[8] M. R. Garey and D. S. Johnson. *Computers and Intractability - A guide to NP-completeness*. W. H. Freeman and Company, San Francisco, first edition, 1979.

[9] A. Hertz and D. de Werra. Using tabu search techniques for graph coloring. *Computing*, 39(4):345–351, 1987.

[10] R. Lewis, J. Thompson, C. Mumford, and J. Gillard. A wide-ranging computational comparison of high-performance graph colouring algorithms. *Computers and Operations Research*, 39(9):1933–1950, 2012.

[11] E. Malaguti, M. Monaci, and P. Toth. A metaheuristic approach for the vertex coloring problem. *INFORMS Journal on Computing*, 20(2):302–316, 2008.

[12] I. Méndez-Díaz and P. Zabala. A cutting plane algorithm for graph coloring. *Discrete Applied Mathematics*, 156:159–179, 2008.

[13] J. Nielsen. The need for web design standards. Online Article: `www.nngroup.com/articles/the-need-for--web-design-standards`, September 2004.

[14] C. M. Valenzuela. A study of permutation operators for minimum span frequency assignment using an order based representation. *Journal of Heuristics*, 7:5–21, 2001.