Constructing Wedding Seating Plans: A Tabu Subject

Rhyd Lewis*

Cardiff School of Mathematics,
Cardiff Univerity, CF10 4AG, WALES.
Email: lewisR9@cf.ac.uk; Tel: +44 (0)2920 874856.

Abstract

This paper examines an interesting combinatorial optimisation problem that generalises both the graph colouring and k-partition problems. The problem has an interesting practical application in the construction of wedding seating plans, where we seek to assign equal numbers of guests to tables such that they are sat near friends and, perhaps more importantly, kept away from their enemies. We describe an effective two-stage metaheuristic-based approach for this problem which is currently used with the online tool on the commercial website www.weddingseatplanner.com. We also present results on the performance of this algorithm, indicating what factors can influence run time and solution quality.

Keywords: Combinatorial Optimisation; Metaheuristics; Graph Colouring; Partitioning.

1 Introduction

The Wedding Seating Problem (WSP) involves taking a set of wedding guests and assigning them to tables so that the following constraints are met:

- Guests belonging to groups, such as couples and families, should be sat at the same tables;
- The number of guests per table should be equal;
- If there is any perceived animosity between different guests, these should be sat on different tables; and similarly,
- If guests are known to like one another, they should be sat at the same table.

*Corresponding author. Submitted to GEM'13 - The 2013 International Conference on Genetic and Evolutionary Methods

The WSP can be formally stated as type of graph partitioning problem. Specifically, we are given a graph $G = (V, E)$ comprising a vertex set V and an edge set E. Each vertex $v \in V$ is used to represent a group of guests who are required to sit together (couples, families, etc.), with the size of each guest group denoted s_v. The total number of guests n is thus $\sum_{v \in V} s_v$. Each edge $\{u, v\} \in E$ then defines the relationship between vertices u and v according to a weighting $w_{u,v}$ (where $w_{u,v} = w_{v,u}$). If $w_{u,v} > 0$ we interpret this to mean that we would prefer the guests associated with vertices u and v to be sat on different tables. Larger values for $w_{u,v}$ reflect a strengthening of this requirement. Similarly, negative values for $w_{u,v}$ mean that we would rather u and v were assigned to the same table.

A solution to the WSP is defined as a partition of the vertices into k subsets $\mathcal{U} = \{U_1, \ldots, U_k\}$, such that $\bigcup_{i=1}^{k} U_i = V$ and $U_i \cap U_j = \emptyset$, $i, j \in \{1, \ldots, k\}$, $i \neq j$. The requested number of tables k is defined by the user, with each subset U_i defining the guests assigned to a particular table i.

The quality of a candidate solution for the WSP can be evaluated according to two objective functions, both which we seek to minimise. The first of these, calculated:

$$f_1 = \sum_{i=1}^{k} \sum_{\forall u,v \in U_i : \{u,v\} \in E} w_{u,v} \tag{1}$$

reflects the extent to which the rules governing who sits with who are obeyed. The second objective function then measures the degree to which the number of guests per table deviates from the required number of either $\lfloor n/k \rfloor$ or $\lceil n/k \rceil$:

$$f_2 = \sum_{i=1}^{k} \left(\min \left(\left| \left(\sum_{\forall v \in U_i} s_v \right) - \left\lfloor \frac{n}{k} \right\rfloor \right|, \left| \left(\sum_{\forall v \in U_i} s_v \right) - \left\lceil \frac{n}{k} \right\rceil \right| \right) \right). \tag{2}$$

It is evident that the WSP is intractable since it generalises two classical NP-hard problems:

- If $E = \emptyset$ (or $\forall \{u, v\} \in E$, $w_{u,v} = 0$) then f_1 (Equation (1)) will always equal 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 k-partition problem.[1]
- If $\forall v \in V$, $s_v = 0$ then f_2 (Equation (2)) will always equal zero, implying that the task of balancing the number of guests per table is no longer relevant. In this case the problem becomes equal to the weighted k-colouring problem, which is itself a generalisation of the NP-hard graph k-colouring problem [3].

In this paper we describe a two stage approximation algorithm for the wedding seating problem. This algorithm is currently used on the commercial website www.weddingseatplanner.com. In particular, this approach exploits the underlying graph structures of the problem allowing effective neighbourhood operators to be defined that are able to quickly identify high-quality solutions. In the next section we

[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.

	Guest	Companion 1	Companion 2	Companion 3
1	John	Sarah	Jack	Jill
2	Bill	June		
3	Pat	Susan		
4	Una	Tom		
5	Ruth	Kevin	Gareth	
6	Ken	Frank	Bobby	
7	Rod	Dereck	Freddy	
8	Jane			

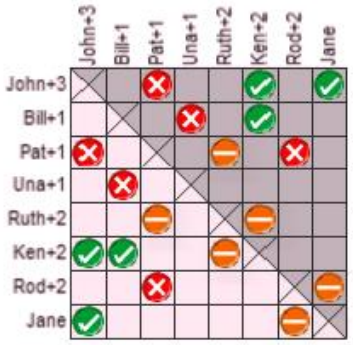

Figure 1: Specification of guest groups (left) and seating preferences (right).

describe the online interactive tool and algorithm in detail, before looking at some its run characteristics in Section 3. Section 4 concludes the paper.

2 Problem Interpretation and Algorithm

On entering the website, the user is asked to input (or import) the names of all guests into an embedded interactive table. Groups of guests that need to be seated together (families etc.) are placed on the same rows of the table, thus defining the various values for s_v. At the next step the user is then asked to define seating preferences between different guest groups.

Figure 1 shows a small example of this process. Here, eight guest groups ranging in size from 1 to 4 have been input, giving 20 guests in total. The right-hand grid then shows how the seating preferences (values for $w_{u,v}$) are defined. If the user wants to define a preference, they do so by clicking on the relevant cells in the grid. 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). In our case these are allocated weights of ∞, 1, and -1 respectively. Note that it would have been possible to allow the user to input their own weights here; however it was felt by the website designers that, while perhaps more flexible, this ran the risk of bamboozling the user while not improving the usability of the tool.

The overall strategy of our algorithm is classify the requirements of 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. We now describe the two stages of the algorithm in more detail.

2.1 Stage 1

In Stage 1 the algorithm operates on the sub-graph $G' = (V, E')$, where $E' = \{\{u, v\} \in E : w_{u,v} = \infty\}$. 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.

The graph k-colouring problem is a widely studied combinatorial optimisation problem for which a multitude of different algorithms are available [5]. In our case, we use a variant of the DSatur heuristic [1] to produce an initial solution. DSatur is a constructive algorithm that, at each iteration, selects the uncoloured vertex v that currently has the largest number of distinct colours assigned to adjacent vertices. Ties are broken by selecting the vertex with the largest degree. In our case the selected vertex v is then assigned to any colour $i \in \{1, \ldots, k\}$ that currently features no other vertices adjacent to v. If no such colour exists, v is kept to one side and is assigned to a random colour at the end of the construction process, thereby introducing violations of the hard constraint.

If the solution produced by the above process contains hard constraint violations, attempts are next made to eliminate these. This is done using the TabuCol algorithm of Hertz and de Werra [4, 2] which, while perhaps not as powerful as more contemporary graph colouring algorithms, does have the advantage of being very fast [5]. 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 using 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 $|V| \times k$ matrix T and, upon performing this move, the element $T_{v,i}$ is marked as tabu for the next t iterations. In each iteration of TabuCol all possible moves from the current solution are considered, and the move that is chosen is the one that is seen to invoke the largest decrease (or failing that, the smallest increase) in cost of any non-tabu move. Ties are broken randomly, and tabu moves are also permitted if they are seen to improve on the best solution found so far.

Because speed is an issue with our online tool, TabuCol is only run for a fixed number of iterations.[3] If at the end of the process a solution with cost $f_1 = 0$ has not been achieved, then it is possible that the user has specified a k-value for which a k-colouring is not achievable (i.e. the number of tables is too small to meet all of the "Definately Apart" constraints). In this case, k is incremented by one and Stage 1 of the algorithm is repeated. This process continues until all hard constraints are satisfied.

[2] For example, using the problem shown in fig. 1, $E' = \{\{1, 3\}, \{2, 4\}, \{3, 7\}\}$.

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

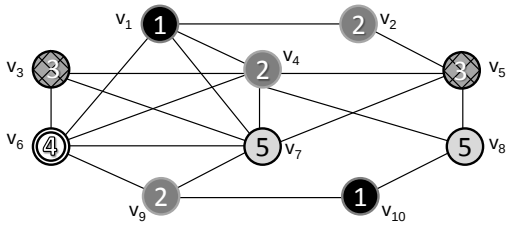

Figure 2: Example of a feasible 5-colouring of graph $G' = (V, E')$.

2.2 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* as it obeys all of the imposed hard constraints. In this stage we now try to eliminate violations of the soft constraints by exploring the space of feasible solutions. Note that movements in this space might be restricted – indeed the space might not even be contiguous – and so it is necessary to use neighbourhood operators that allow as much freedom-of-movement as possible. In our case this is achieved using tabu search in conjunction with two operators, both that exploit the underlying structure of the graph colouring problem. These are defined as follows:

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 starting from v that only contains vertices coloured with i and j. Such a chain can be denoted $\textsc{Kempe}(v, i, j)$.

A Kempe-chain interchange involves taking a particular Kempe-chain and swapping the colours of all vertices contained within it. For example, in fig. 2, $\textsc{Kempe}(v_7, 5, 2)$ results in a chain involving vertices $\{v_4, v_9\}$ (assigned to colour 2) and $\{v_7, v_8\}$ (assigned to colour 5). When the colours of these vertices are interchanged, we observe that the resultant solution will still be feasible. This is actually the case with all Kempe-chain interchanges [6].

It is also worth noting that Kempe-chains can vary in size, and some combinations for $\textsc{Kempe}(v, i, j)$ will *overlap* in that they result in the same Kempe-chain being identified (e.g. $\textsc{Kempe}(v_7, 5, 2) = \textsc{Kempe}(v_9, 2, 5)$). 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.

Swaps: The swap operator is used for performing further moves not contained within the Kempe-chain neighborhood. Specifically, when scanning the set of Kempe-chain interchanges, we can also identify pairs of non-adjacent vertices u, v that both feature Kempe-chains $\textsc{Kempe}(u, i, j)$ and $\textsc{Kempe}(v, j, i)$ of size 1 (e.g.

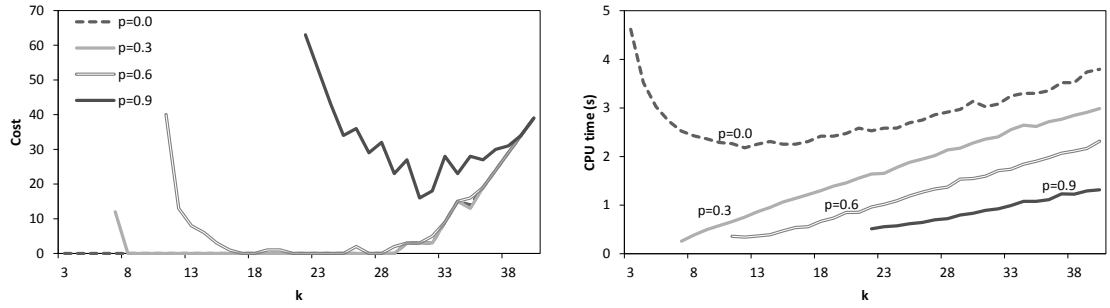

Figure 3: Solution costs (left) and run times (right) for four instances using various k-values. For most of the left figure, the line for $p = 0.0$ is obscured by the line for $p = 0.3$ (i.e. the same results were achieved).

> KEMPE$(v_1, 1, 3)$ and KEMPE$(v_5, 3, 1)$ in fig. 2.) 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 this stage of the algorithm all possible moves from both 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 is simply $(f_1 + f_2)$ which will always evaluate to a value less than ∞ (since violations of the hard constraints cannot occur).

3 Algorithm Performance

The algorithm and interface described above has been coded in ActionScript 3.0 and can be run via a web browser at www.weddingseatplanner.com. To ensure run times are kept relatively short, and also to allow the interface to be displayed clearly on the screen, problem size is 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 large problem instance of $|V| = 50$ guest groups was constructed, with the size of each group chosen uniform randomly in the range 1 to 8. This instance was 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 $p = \{0.0, 0.3, 0.6, 0.9\}$ and number of tables $k = \{3, 4, \ldots, 40\}$.

Figure 3 shows the results of these tests with regard to the costs and run times that were achieved at termination. Note that for $p > 0.0$, values are not reported for the lowest k's as feasible k-colourings were not achieved (quite possibly because they

[4]That is, all nodes and colours effected are marked as tabu in T. For our application a tabu tenure of 10 is used along with an iteration limit of $10n$.

do not exist). Also note that all runs took less than 5 seconds on our machine (3.0 GHz Windows XP machine with 3.18 GB RAM).

Figure 3 (left) demonstrates that with no hard constraints, balanced table sizes can be 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 equally. Higher costs are also realised when $p > 0.0$ and the lowest achievable values for k are used. This is because many guest group combinations (including many of those that are required for achieving low cost solutions) will now contain at least one hard constraint violation, meaning they will not be considered by the algorithm. This also explains why low cost solutions are not achieved using $p = 0.9$.

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. 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 shown in Figure 3 (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.

4 Conclusions

An effective two stage heuristic algorithm has been proposed for the wedding seating problem, making use of concepts taken from the observed underlying graph colouring model. Experiments have been presented to demonstrate factors that influence solution quality and run times.

The described tool has been live at www.weddingseatplanner.com since mid-2011 and receives approximately 1000 hits per month at the time of writing. We have also found that it can be a useful tool for introducing students to many of the issues surrounding combinatorial optimisation.

References

[1] D. Brelaz. New methods to color the vertices of a graph. *Commun. ACM*, 22(4):251–256, 1979.

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

[3] 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.

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

[5] 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.

[6] J. Thompson and K. Dowsland. A robust simulated annealing based examination timetabling system. *Computers and Operations Research*, 25(7/8):637–648, 1998.