On the Application of Graph Colouring Techniques in Round-Robin Sports Scheduling

R. Lewis[*] and J. Thompson
Operational Research Group, Cardiff School of Mathematics,
Prifysgol Caerdydd/Cardiff University, Cardiff CF24 4AG,
WALES.
tel: 0044 (0)2920 874856; fax: 0044 (0)2920 874199
email: {lewisR9,thompsonJM1}@cf.ac.uk

May 13, 2010

Abstract

The purpose of this paper is twofold. First, it explores the issue of producing valid, compact round-robin sports schedules by considering the problem as one of graph colouring. Using this model, which can also be extended to incorporate additional constraints, the difficulty of such problems is then gauged by considering the performance of a number of different graph colouring algorithms. Second, neighbourhood operators are then proposed that can be derived from the underlying graph colouring model and, in an example application, we show how these operators can be used in conjunction with multiobjective optimisation techniques to produce high-quality solutions to a real-world sports league scheduling problem encountered at the Welsh Rugby Union in Cardiff, Wales.

1 Introduction

Round-robin (RR) sports schedules occur in many tournaments and leagues across the globe, including the Six Nations Rugby Championships, various European and South American domestic soccer leagues, and the England and Wales County Cricket Championships. Round-robins are schedules involving n teams, where each team is required to play all other teams exactly m times within a fixed number of *rounds*. The most common types are *single* round-robins (SRRs), where $m = 1$, and *double* round-robins (DRRs), where $m = 2$ (in the latter, teams will typically be scheduled to meet once in each other's home venue).

Usually, the number of teams n in a round-robin schedule will be even. In cases where n is odd, an extra "dummy team" can be introduced, and teams assigned to play the dummy team will receive a bye in the appropriate part of the schedule. Round-robins are considered *valid* if each team in the schedule competes at most once per round. They are also described as *compact* if the number of rounds used is minimal at $m(n-1)$, thus implying $\frac{n}{2}$ matches-per-round. (All of the sports competitions mentioned above are compact, valid round-robins.)

It has been known for some time that valid, compact RR schedules can be constructed for any number of teams n [26]. However, it is also known that the number of *distinct* round-robin schedules grows exponentially with n, since this figure is monotonically related to the number of non-isomorphic

[*]Corresponding Author

Round	Matches
r_0 :	{0,1}, {2,8}, {3,7}, {4,6}, {5,9}
r_1 :	{0,2}, {1,9}, {3,8}, {4,7}, {5,6}
r_2 :	{0,3}, {1,2}, {4,8}, {5,7}, {6,9}
r_3 :	{0,4}, {1,3}, {2,9}, {5,8}, {6,7}
r_4 :	{0,5}, {1,4}, {2,3}, {6,8}, {7,9}
r_5 :	{0,6}, {1,5}, {2,4}, {3,9}, {7,8}
r_6 :	{0,7}, {1,6}, {2,5}, {3,4}, {8,9}
r_7 :	{0,8}, {1,7}, {2,6}, {3,5}, {4,9}
r_8 :	{0,9}, {1,8}, {2,7}, {3,6}, {4,5}

Figure 1: Demonstration of the greedy algorithm for producing valid, compact round-robin schedules for $n = 10$. The matches are first ordered lexicographically ($\{0,1\},\{0,2\},\{0,3\},\ldots,\{8,9\}$) and the first match is assigned to the first round. Each remaining match is then considered in turn and assigned to the next round where no clash occurs. When the final round is reached, the algorithm loops back to the first round. Note that if a DRR is required, the second half of the schedule can be produced by simply copying the first half.

one-factorisations of a complete graph with n vertices (which is also known to increase exponentially with n). Such a growth rate implies that, for non-trivial problems, enumerating all possible schedules will not be possible in reasonable time (in practice, "non-trivial" values for n would appear to be anything above 10 for SRRs [18].)

A number of constructive algorithms have previously been proposed for producing compact, valid round-robin schedules in linear time, including the *greedy algorithm* [26, 3], which operates by assigning matches to rounds in lexicographic order (see fig. 1), and the *polygon* method considered, for example, by Riberio and Urrutia [39] and Anderson [2]. In fact the solutions produced via the greedy and polygon methods are isomorphic, since schedules produced via one method can be converted into the other by simply relabelling the teams and rounds, as proved by Anderson [2]. This is also true of the solutions produced via the *canonical* algorithm, which, in addition to producing an RR schedule, also copes with the issue of minimising the number of "breaks" (a break being a situation where a team is required to play two or more home (or away) games in consecutive rounds). This latter method, proposed by de Werra [11], has previously been used in commercial sports timetabling software, such as Splendid City.[1] More recently the task of minimising breaks has also been explored by other authors. Trick [45], Miyashiro and Matsui [34], and Elf [21], for example, have examined the problem of taking an *existing* SRR and then assigning home/away values to each of the matches in order to minimise the number of breaks. Miyashiro and Matsui [32] have also shown that the problem of deciding whether a home/away assignment exists for a particular schedule such that the theoretical minimum of $n-2$ breaks can be met is computable in polynomial time. Meanwhile, the inverse of this problem – taking a fixed home/away pattern and then assigning matches consistent with this pattern – has also been studied by various other authors [41, 11, 36]. A good survey on many aspects of break minimisation can be found in the recent survey of Rasmussen and Trick [38].

Although it is straightforward to construct valid, compact RR schedules using constructive methods such as the greedy, polygon, and canonical algorithms, in practical circumstances it is often the case that the production of "high-quality" schedules will depend on additional user requirements and constraints. The minimisation of "breaks" (mentioned in the previous section) is one example of this. Other constraints, however, can include those associated with demands from broadcasters,

[1] `www.splendidcity.net`

various economical and logistical factors, inter-team politics, policing, and the perceived fairness of the league. In addition, factors such as the type of sport, the level of competition, and the country (or countries) involved will also play a part in determining what is considered "high-quality". An impression of the wide range of such constraints and requirements can be gained by considering the variety of problems that have previously been tackled in the operational research literature, including German, Austrian, and Italian soccer leagues [6, 13], New Zealand basketball leagues [49], amateur tennis tournaments [14], English county cricket fixtures [48], and American professional ice hockey [22]. A good survey on the wide range of problems and solution methods for sports scheduling can be found in the recent paper of Kendall et al. [25].

An oft-quoted example of such requirements is the issue of *carry-over* in RR schedules, where we consider the possibility of a team's performance being influenced by their opponents in previous rounds. For example, if we have a very strong team t whose opponents are often left injured or demoralised, then a team that plays t's opponents in the next round may well be seen to gain an advantage.[2] In such cases the aim is to therefore produce a round-robin schedule in which the overall effects of carry-over are minimised. This requirement was first considered by Russell [40] who also proposed a constructive algorithm able to produce provably optimal schedules in cases where n is a power of 2. Later, better results for other values of n were achieved by Anderson [4] using a sophisticated constructive algorithm. More recently still, methods for small-sized problems have also been proposed by Trick [45] and Henz [24] who both make use of constraint programming techniques.

Another set of schedule requirements, which has become somewhat of a benchmark over the past decade, is encapsulated in the Traveling Tournament Problem (TTP) [19]. In this formulation a compact DRR schedule is required (teams play each other twice, once in each other's home venue) and the overriding aim is to minimise the *distances travelled* by each team. Geographical constraints such as this are particularly relevant in large countries such as Brazil and the USA where match venues are typically far apart. Consequently, when a team is scheduled to attend a succession of away games, instead of returning to their home city after each match, the team travels directly to their next away venue. Note that in addition to the basic RR scheduling constraints, this problem also contains elements of the traveling salesman problem, as we are interested in scheduling runs of successive away matches for each team such that they occur in venues that are close to one another. An early solution method proposed for this formulation was proposed by Easton et al. [20] who used integer- and constraint-programming techniques. More recent proposals, however, have focused on metaheuristic techniques, and in particular, neighbourhood search-based algorithms. Good examples of these include the simulated annealing approaches of Anagnostopoulos et al. [1] and Lim et al. [29], and the local-search approaches of Di Gaspero and Schaerf [16] and Ribeiro and Urrutia [39]. In the latter example the authors also consider the added restriction that the DRR schedule should be "mirrored": that is, if teams t_i and t_j play in t_i's home stadium in round $r \in \{1, \ldots, n-1\}$, then t_i and t_j should necessarily play each other in t_j's home stadium in round $r + (n-1)$. In the other algorithms, this restriction is not considered. At the time of writing, the metaheuristic approaches listed here produce state of the art results for this problem.

1.1 Paper Scope and Outline

Due to the noted idiosyncratic nature of problem formulations encountered in sports scheduling, it is perhaps quite natural that heuristic search algorithms, and particularly metaheuristics, hold some promise in this research area. There are two reasons for this. Firstly, such methods by their very

[2] As an illustration, in fig. 1 we see that team 1, for instance, is scheduled to play the opponents of team 0 from the previous round on seven different occasions. This feature also exists for teams t_i and t_{i+1} for $0 \leq i \leq 7$. Thus this schedule actually contains rather a large amount of carry-over.

nature are intended to offer algorithmic frameworks that are adaptable to a range of different problem and constraint types. Secondly, if they are used correctly, metaheuristics can also provide effective ways of navigating through large search-spaces (of the sort encountered in sports scheduling problems), thus allowing good-quality solutions to be found in reasonable amounts of time.

To our knowledge, all current applications of metaheuristics in RR-scheduling have adopted a two-stage strategy in which a valid, compact round-robin schedule is first constructed, with changes then being made to this schedule via search operators that preserve these characteristics. Such a strategy would seem appropriate as it ensures, if nothing else, that the solution produced meets the most basic requirements of a round-robin schedule. In this paper our intention is to follow this approach, though we also choose to view sports timetabling as a type of graph colouring problem, enabling us to borrow various techniques from the graph-colouring literature. We note that graph colouring concepts are often used with other types of scheduling problem, particularly university timetabling problems [43, 8, 28], though this has not previously been the case with sports scheduling. In Section 2 we thus begin by describing how basic RR problems can be represented as graph colouring problems, and discuss the properties of such graphs. In Section 3 we then attempt to assess the "difficulty" of solving such graphs by applying a number of well-known, but operationally distinct graph colouring algorithms. Following this, in Section 4 we then introduce ways in which our proposed model can be extended in order to incorporate other types of "hard" (i.e. mandatory) constraint, and in Section 5 we discuss various different neighbourhood operators that can be used with this extended model for exploring the space of *feasible* solutions (that is, solutions that are compact, valid, and which also obey any imposed hard constraints). Finally in Section 6 we consider a complex real-world scheduling problem that was given to us by the Welsh Rugby Union, based in the Millennium Stadium in Cardiff, Wales, and we propose two separate algorithms that make use of our proposed algorithmic operators. The performance of these algorithms is then analysed over a number of different problem instances and conclusions are drawn.

2 Representing Basic Round-Robins as Graph Colouring Problems

Given a simple and undirected graph $G = (V, E)$, the NP-hard graph colouring problem (or "vertex colouring problem") involves assigning "colours" to each vertex $v \in V$ such that (a) no pair of adjacent vertices is assigned the same colour, and (b) the number of colours used is minimal [23]. The minimum number of colours required to colour a particular graph is called the "chromatic number" denoted χ.

Round-robin scheduling problems can be represented as graph colouring problems by considering each individual match as a vertex, with edges then being added between any pair of matches that cannot be scheduled in the same round (i.e. matches featuring a common team). The colours then represent the individual rounds of the schedule, and the task is to colour the graph using k colours, where k represents the number of available rounds. Note that for the remainder of this paper we only consider the task of producing compact schedules: thus, $k = \chi$ in all cases unless otherwise specified.

Example graphs for $n = 4$ are provided in fig. 2. For our purposes, in the SRR case we represent each vertex $v \in V$ as an unordered pair $v = \{t_i, t_j\}$, denoting a match between teams t_i and t_j for $0 \leq i \neq j < n$. The number of vertices $|V|$ in SRRs is thus $\frac{1}{2}n(n-1)$, and for a compact schedule, the number of available colours $k = n - 1$. For DRRs the number of vertices $|V| = n(n-1)$ and $k = 2(n-1)$, since teams will play each other twice. In this case, we can also represent each vertex using an ordered pair (t_i, t_j), where $i \neq j$, and can follow the sporting tradition of denoting the first team as the "home" team and the second team as the "away" team (that is, a match (t_i, t_j) will take place in t_i's home stadium).

Note that the vertices in such graphs are all equal in degree: $2(n-2)$ for SRRs, and $4(n-2)+1$

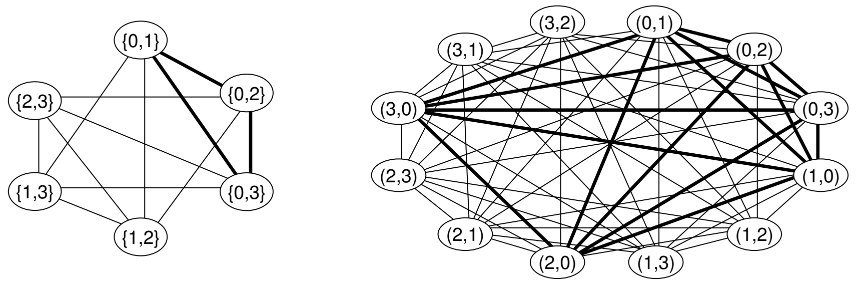


Figure 2: Graphs for single (left) and double (right) round-robin problems with $n = 4$.

for DRRs. The edge densities D are thus:

$$D_{\text{sin}} = \frac{0.5 \times |V| \times 2(n-2)}{0.5 \times |V| \times (|V|-1)} = \frac{4}{n+1} \tag{1}$$

for SRRs, and

$$D_{\text{dob}} = \frac{0.5 \times |V| \times (4(n-2)+1)}{0.5 \times |V| \times (|V|-1)} = \frac{4(n-2)+1}{|V|-1} \tag{2}$$

for DRRs. (The denominators in the leftmost formulae represent the number of pairs of vertices in the graphs; the numerators represent the total number of edges). For both SRRs and DRRs the generated graphs thus belong to the class of *k-colourable*, *equipartite* graphs, because solutions will use $k = \chi$ colours, with each colour containing exactly $\frac{n}{2} = \frac{|V|}{\chi}$ vertices-per-colour. Since the degrees of all vertices are equal, such graphs may also be considered *flat* graphs [10]. Note that vertices associated with each team also constitute *cliques*: in SRRs these cliques are of size $n-1$, and in DRRs they are of size $2(n-1)$. Cliques arising due to team 0 are shown in bold in fig. 2.

Note that many previous methods for sports timetabling have also used graphs for representing round-robins, though these have usually been of a different form [11, 31, 39]. Specifically, in such cases a complete graph K_n is used, with each vertex denoting a team and each edge denoting a match between its two endpoints. In such cases, the task is to find a proper $(n-1)$-colouring of the edges which, for a complete graph, is equivalent to the problem of achieving a one-factorisation of K_n (as alluded to earlier, this goal is always achievable for even values of n [26, 31]). In such an edge colouring, all edges of a particular colour will necessarily be a perfect matching (that is, a one-factor) and will indicate the matches that are to occur in the same round in the round-robin. On the other hand, the graphs generated using our methods are the corresponding *line graphs* of these complete graphs. Thus each vertex in our model corresponds to an edge in K_n, with pairs of these vertices being adjacent if and only if the corresponding edges in K_n share an endpoint. Of course, in practice it is easy to switch between these two representations; however, the main advantage of using our representation is that it allows the exploitation of previously developed vertex-colouring techniques, as the following sections demonstrate.

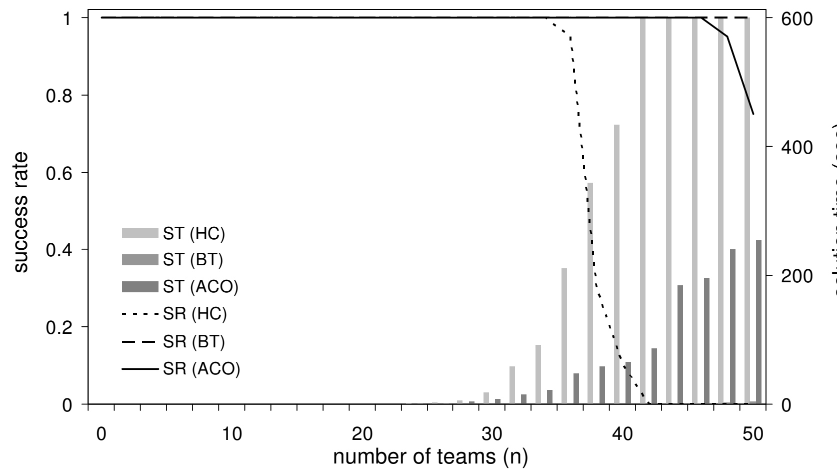
3 Generating Valid Round-Robin Schedules

Having defined the basic structures of "round-robin graphs", in this section we investigate whether such graphs actually constitute difficult-to-colour problem instances. Note that by k-colouring such graphs we are doing nothing more than producing valid, compact round-robin schedules which, as we have mentioned, can be easily constructed using the linear greedy, polygon, and canonical algorithms. However, there are a number of reasons why solving these problems from the perspective of graph colouring might be worthwhile.

- Because of the structured, deterministic way in which the linear methods operate the range of output that can be produced by these algorithms will only represent a very small part of the space of all valid RRs.
- The schedules that are produced by the linear methods also occupy very *particular* parts of the search space. For example Miyashiro and Matsui [33] have conjectured that the polygon method produces schedules in which the amount of carry-over is actually maximised. This conjecture also applies to the (isomorphic) solutions produced via the greedy and canonical algorithms.
- The specific structures present in schedules produced by the linear algorithms can also sometimes have adverse effects when applying neighbourhood search operators (we expand on this topic in Section 5).
- Finally, we are also able to modify the graph colouring model to incorporate additional constraints (see Section 4).

By using graph colouring methods, particularly those that are stochastic in nature, the hope is that we thus have a more robust mechanism able to produce RR schedules in a less biased fashion, allowing a larger range of structurally distinct schedules to be achieved. Note that this will be especially useful in the application of metaheuristics, where the production of random initial solutions is usually required.

We mentioned earlier that the specific structure of round-robin graphs (k-colourable, equipartite, no variance in vertex degrees) means that they are a type of *flat* graph. Often flat graphs are considered difficult to solve because there is little heuristic information for algorithms to exploit (i.e. all vertices "look the same"). Various works, however, have demonstrated that such difficulty only exists in flat graphs with edge densities within specific ranges, often referred to as the "phase transition region" [9, 46, 27]. In order to assess the difficulty of our round-robin graphs we attempted to solve problems of a range of different sizes using three well-known stochastic-based algorithms: the *Hill-Climbing* (HC) approach of Lewis [27], a *Backtracking* (BT) algorithm based on the Dsatur heuristic [7], and the hybrid *Ant Colony Optimisation* (ACO) algorithm of Thompson and Dowsland [44]. For our purposes, each of these methods can be considered as representing a different type of search strategy: the HC approach uses a greedy macro-search operator in conjunction with a second greedy *local*-based search operator; the BT method is a heuristic-based branch-and-bound algorithm; and the ACO approach uses a combination of global heuristic search and a more intense local search operator (provided by pheromone information and tabu search respectively). For comparative purposes we also produced an Integer Programming (IP) model for these graphs which we then attempted to solve using the commercial solver FICO Xpress (optimiser version 20.00.05), which also operates under a branch-and-bound framework. The specific IP model used in our experiments is defined in Appendix A. For all experiments, runs were specified to terminate once a solution using the minimal $k = \chi$ colours was achieved, or once an upper time limit of 600 CPU seconds was reached (all trials, except those using Xpress, were conducted on a PC under Linux with a 1.8GHz processor and 256MB RAM). Note that it is not our intention here to provide a through and critical comparison of these techniques: instead, we use these algorithms merely to provide an indication as to the difficulty of the considered graphs. Indeed, such a comparison could not be strictly accurate in any case since the algorithms



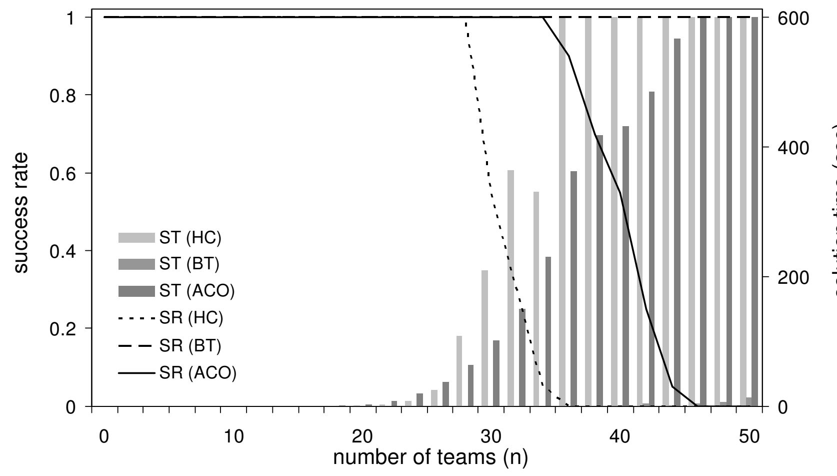


Figure 3: Success Rate (SR) and Solution Time (ST) for the HC, BT, and ACO algorithms when solving SRRs and DRRs of sizes $n \leq 50$.

were implemented by different people at different times using different languages, although we did compensate for this by compiling all algorithms under the same options using compilers from the same source (the Gnu collection under the `-O3` optimisation option).)[3]

Figure 3 summarises the results of our experiments for SRRs and DRRs of up to 50 teams. For each algorithm and for each value of n two measures are reported: the success rate (SR) and the solution time (ST) from 20 individual runs. The success rate indicates the proportion of runs in which a valid compact schedule (i.e. a feasible graph colouring solution using k colours) was achieved, and the solution time indicates the mean time that it took to find such solutions (in cases where the success rate is less than 1.0, only successful runs were considered in the latter's calculation). Note that because these algorithms operate in different ways, we were unable to compare these algorithms using metrics that reflect the accuracy of the algorithms, such as the number of extra colours or the number of uncoloured vertices.

[3]In more detail, the HC algorithm [27] was coded in C++; the BT algorithm was compiled from publicly available C code (www.cs.ualberta.ca/~joe/Coloring/); and the ACO algorithm was coded in Fortran.

The results in Figure 3 suggest that the ACO algorithm is better than the HC algorithm in solving these graphs, both from the perspective of success rates and solution times. For both of these algorithms we also see that the success rates begin to dip as n approaches 50, meaning that the algorithms cannot always achieve k-coloured solutions within the time limit. This can be expected to a certain extent since these problems are quite large (1225 and 2450 vertices for SRRs and DRRs respectively) and will obviously feature very large search spaces in which the algorithms have to navigate. In contrast the BT algorithm is far more successful, and in our tests was able to produce solutions in all cases in less than 15 CPU seconds (and less than 5 seconds in 98% of cases). For our trials these results were achieved by setting the branching factor of the algorithm to 1, which meant that when the algorithm backtracked, only one other path was inspected at each node in the search tree. Under these settings it would seem that the BT algorithm's operational characteristics and heuristics naturally exploit the underlying structures contained in these graphs, allowing k-coloured solutions to be produced in very short amounts of time. We did find, however, that if we increased the branching factor of the algorithm, then the success rates and solution times tended to worsen considerably. Finally, not shown in fig. 3 are the results that were gained using the IP model. Here we found that Xpress could only find solutions for SRRs with $n \leq 20$ and for DRRs with $n \leq 14$. This was despite the fact that these experiments were conducted on a quicker machine (Windows PC, with 2.83GHz processor and 3.2GB RAM) which meant that the 600-second time limit we imposed allowed approximately double the amount of computation compared to the stochastic graph colouring algorithms.

4 Extending the Graph Colouring Model

A particular advantage of transforming the task of round-robin construction into a type of graph colouring problem is that we can easily extend the model to incorporate other types of scheduling constraints. In this section, we specifically consider the imposition of *round-specific* constraints, which specify the rounds that matches can and cannot be assigned to. In many practical cases, round-specific constraints will be a type of hard constraint – that is, they will be mandatory in their satisfaction – and candidate solutions that violate such constraints will be considered infeasible. Encoding such constraints directly into the graph colouring model thus allows us to attach an importance to these constraints that is equal to the basic RR constraints themselves.

To impose round-specific constraints we follow the method of Balakrishnan [5]: first, k extra vertices are added to the model, one for each available round. Next, edges are then added between all pairs of these "round-vertices" forming a clique of size k, and ensuring that each round-vertex will be assigned to a different colour in any feasible solution. Having introduced these extra vertices, a variety of different round-specific constraints can then be introduced:

1. **Match Unavailability Constraints:** Often it will be impossible to assign a match to a particular round, perhaps because the venue of the match is being used for another event, or because league rules state that the associated teams should not play each other in specific rounds in the league. Such constraints are introduced by adding an edge between the relevant match-vertex and round-vertex.

2. **Preassignment Constraints:** In some cases, a match will need to be assigned to a specific round r, e.g. to increase viewing figures and/or revenue. For such constraints, edges are added between the appropriate match-vertex and all round-vertices except for round-vertex r.

3. **Concurrent Match Constraints:** In some cases it also might be undesirable for two matches (t_i, t_j) and (t_l, t_m) to occur in the same round. For example, teams t_i and t_l may share a stadium,

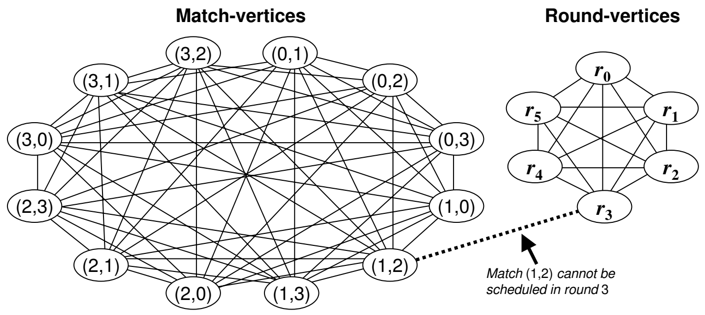


Figure 4: Method of extending the graph colouring model to incorporate round-specific constraints.

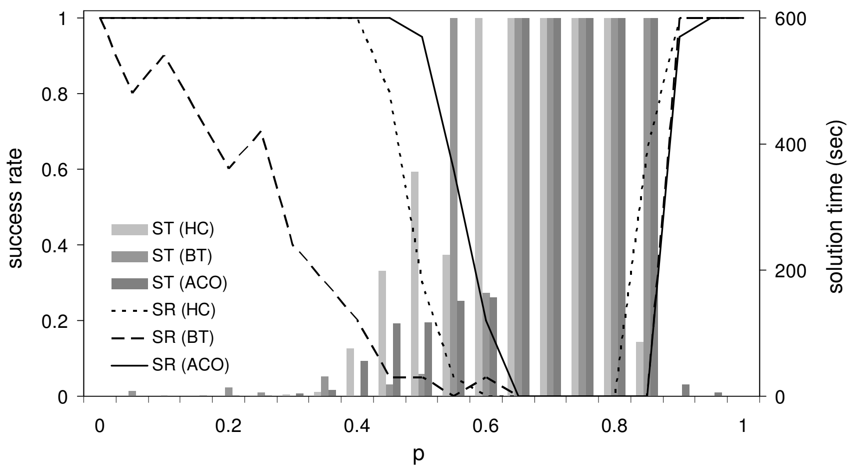
or perhaps it is forbidden for rival fans of certain teams to visit the same city during the same day. Such constraints can be introduced by assigning an edge between the appropriate pairs of match-vertices.

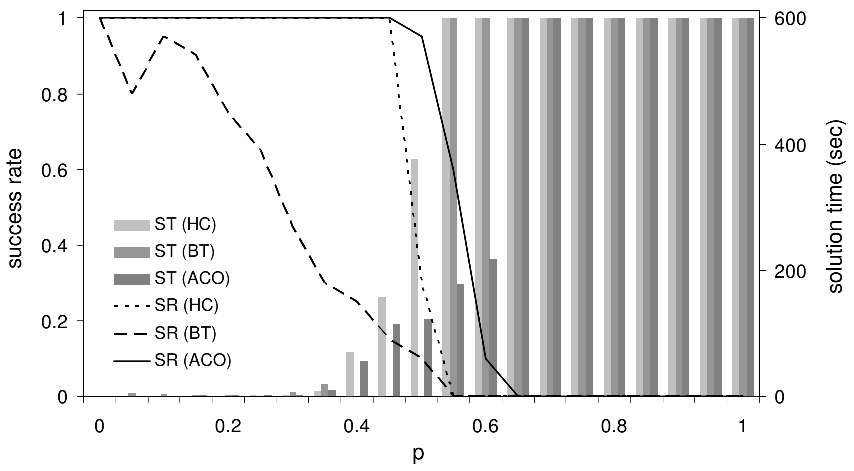
Figure 4 gives an example of how we can impose such extra constraints. It is obvious that any feasible k-coloured solution for such graphs will constitute a valid compact RR schedule that obeys the imposed round-specific constraints. As we shall see in Section 5, by incorporating constraints in this fashion we will also be able to apply various neighbourhood operators stemming from the underlying graph colouring model that ensure that these extra constraints are never re-violated.

We should note that an alternative strategy for coping with additional hard constraints such as these is to *allow* their violation within a schedule, but to then penalise their occurrence via a cost function. Anagnostopoulos et al. [1], for example, use a strategy whereby the space of all valid compact solutions is explored, with a cost function then being used that reflects the number of hard *and soft* constraint violations within a schedule (though using different hard-constraints to those mentioned above). Weights are then used to place a higher penalty on violations of the hard constraints, and it is hoped that by using such weights the search will eventually move into areas of the search space where no hard constraint violations occur. Note that the choice of which strategy to employ will largely depend on practical requirements: in instances where the hard constraints are quite easy to satisfy, or if it is acceptable for some hard constraints to be violated, then an approach such as this may well be acceptable. However, in cases where hard constraint violations are *un*acceptable, then encoding such constraints directly to the underlying graph colouring model would seem to be an appropriate alternative.

To investigate the effects that the imposition of round-specific constraints has on the difficulty of the underlying graph colouring problem, we created an instance generator that randomly added occurrences of the first two constraints above. Specifically, for match unavailability constraints each match/round pair was considered in turn, and an edge was then added between the associated vertices with probability p. For constraint 2, a similar process was also used whereby each match was considered in turn and then preassigned to a randomly selected round with probability q. To further aid our investigations, two versions of this generator were considered, one in which the presence of a k-colourable instance was always guaranteed, and one that payed no heed to this matter, perhaps producing instances where $k < \chi$ (that is, where no k-colourable solution is achievable).

The charts in fig. 5 illustrate the ability of the HC, BT and ACO algorithms for solving a range of problems produced by our generator. In this figure we specifically consider DRRs with $n = 16$, which is representative of the practical problems that we consider in Section 6. As with earlier figures a CPU-time limit off 600 seconds was imposed, and we report the success rates and solution times of





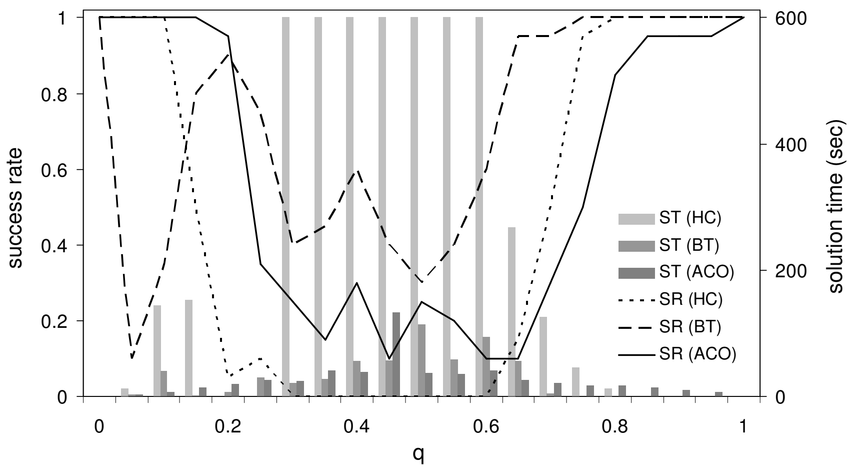


Figure 5: Success rates (SR) and solution times (ST) for the HC, BT and ACO algorithms with DRRs for $n = 16$ (therefore $k = 30$). The top and middle graphs show the results for varying values of p with $q = 0.0$ using instances that are (a) k-colourable, and (b) are not guaranteed to be k-colourable, respectively. Results for k-colourable graphs with varying values of q (with $p = 0.0$) are shown in the bottom graph (c).

each algorithm, in this case from runs on twenty randomly generated instances for each value of p and q. Note that due to performance issues, the branching factor of the BT algorithm was set to excess here, meaning that the algorithm would perform a complete search were to be granted excess time.

Considering fig. 5(a), where k-colourable solutions are guaranteed to exist, a phase-transition region is clearly visible at around $p = 0.65$ to 0.8 (that is, none of the algorithms are able to achieve k-coloured solutions within this range). Either side of this region, the HC and ACO algorithms seem the most robust, with the BT algorithm tending to be less consistent. Similar patterns are also present in fig. 5(b), though in this case we see that all algorithms fail for high values of p since these instances almost certainty feature no k-colourable solutions.

In fig. 5(c), which concerns k-colourable problems with additional preassignment constraints, a phase transition region is again visible, this time at around $q = 0.3$ to 0.6, though it is wider and shallower than fig. 5(a). Again, the HC and ACO algorithms show similar patterns on the whole, with ACO performing slightly better in the left and centre of the phase transition, but slightly worse on the right hand side. In contrast to the previous charts, however, we see that the BT algorithm is the most consistent, with its success rate never falling below 0.3 in the phase transition region. Perhaps this is due to the search spaces of such problems being too constrained for the search mechanisms of the HC and ACO algorithms whereas the presence of so many constraints provides a smaller search tree for the BT algorithm to negotiate.

In contrast to the HC and ACO algorithms, fig. 5(c) also reveals that the BT algorithm seems to experience a "mini phase-transition" in the region $q = 0.05$ to 0.15. Perhaps this is due to the solution density of the associated problems: at $q = 0.0$, for example, the number of solutions is large and the BT algorithm seems able to establish one of these quite easily; however, when q is raised a small amount, although this results in a slight reduction in the size of the search tree, there is a much larger reduction in the number of solutions, which could account for the algorithms lack of success in this region. Note that beyond this "mini phase-transition" region similar, though slightly superior, performance to the HC and ACO algorithms is witnessed in general.

Using our instance generator, we also performed similar experiments for preassignment constraints for which k-colourable solutions were not guaranteed. However in such cases, as q is raised it becomes increasingly likely that two or more conflicting matches will be preassigned to the same round, thus ensuring that $k < \chi$. For DRRs with $n = 16$, for example, we found that no k-colourable solutions could be established for $q \geq 0.1$ with any of the algorithms.

Finally, we repeated the above experiments using a range of different problem sizes. As n was increased, the same phase transitions were observed though they also tended to widen and deepen. For problems where k-colourable instances were not guaranteed, the points at which instances could not be k-coloured also tended to decrease. Thus, the algorithms find larger problems of this nature more difficult on the whole.

5 Exploring the Search Space using Neighbourhood Operators

As mentioned earlier, upon production of a valid round-robin schedule, the usual strategy (as far as metaheuristics are concerned) is to apply neighbourhood search operators that try to eliminate occurrences of any remaining constraint violations. Table 1 lists a number of neighbourhood operators that have been proposed in the literature, mostly for use in solving the TTP [1, 15, 39] (note that the information given in this table applies to DRRs, and so the number of rounds $k = 2(n - 1)$; however simple adjustments can be made for other cases). A point to note about these operators is that while all preserve the validity of a round-robin schedule, they will not be useful for optimisation in all circumstances. For example, applications of N_1, N_2, and N_3 will not affect the amount of carry-

Table 1: Description of various neighbourhood operators that preserve the validity and compactness of round-robin schedules [1, 15, 39]. "Move Size" refers to the number of matches (vertices in the graph colouring model) that are effected by the application of these operators. (This value is unspecified for N_5 as it depends on the exact details of the repair scheme used, which has tended to alter in different papers).

	Description	Move Size
N_1	Select two teams, $t_i \neq t_j$, and swap the rounds of vertices (t_i, t_j) and (t_j, t_i).	2
N_2	Select two teams, $t_i \neq t_j$, and swap their opponents in all rounds.	$2(k-2)$
N_3	Select two teams, $t_i \neq t_j$, and swap all occurrences of t_i to t_j and all occurrences of t_j to t_i.	$2k-2$
N_4	Select two rounds, $r_i \neq r_j$ and swap their contents.	n
N_5	Select a match and move it to a new round. Repair the schedule using an ejection chain repair procedure.	Variable

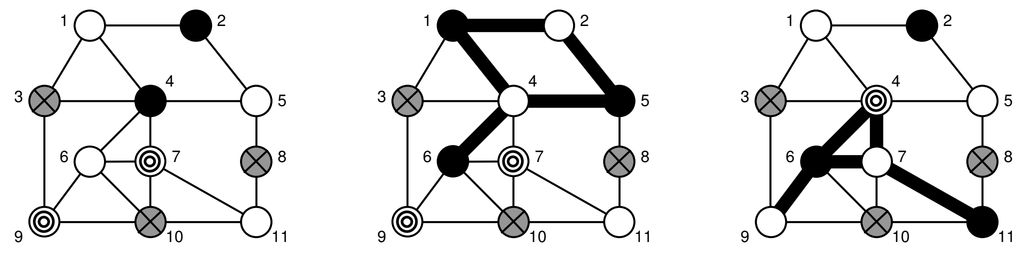
over in a schedule. Also, while applications of N_2 can have an effect on the home-away patterns on individual teams, they cannot alter the *total* number of breaks in a schedule. Finally, perhaps the most salient point from our perspective is that if extra hard constraints are being considered, such as the round-specific constraints mentioned in Section 4, then the application of such operators may lead to schedules that, while valid, are not necessarily feasible.

Pursuing the relationship with graph colouring, a promising strategy for exploring the space of solutions that are both valid *and* feasible seems to be presented by the Kempe- and S-chain operators [35, 43]. A Kempe-chain neighbourhood takes a graph G and involves identifying a connected sub-graph that contains exactly 2 colours. If G is feasibly coloured (i.e. it contains no pairs of adjacent vertices with the same colour), then swapping the colours of all vertices within such a sub-graph will lead to a new feasible colouring that uses no more colours than the original. A demonstration of this process is given in fig. 6 where we use the notation $\texttt{Kempe}(v, c)$ to denote a move that involves a randomly selected vertex v and colour c (where v is not currently assigned to colour c). Note that in the case of round-robin graphs, the number of different choices for $\texttt{Kempe}(v, c)$ is $|V|(k-1)$, though as Di Gaspero and Schaerf [16] note, some of these selections will result in the same chain being identified and thus the same move being performed. (For example, in fig. 6 the same moves will also occur if the other vertices in the chain (and opposite colours) are selected: i.e. $\texttt{Kempe}(1, \text{black})$, $\texttt{Kempe}(5, \text{black})$, $\texttt{Kempe}(6, \text{black})$, $\texttt{Kempe}(2, \text{white})$, and $\texttt{Kempe}(4, \text{white})$.)

S-chains, meanwhile, can be considered an extension of the Kempe-chain principles whereby more colours are considered. Thus $S \geq 3$ colours are identified and using an ordered list of these colours together with a recursive procedure to identify the S-chain, various sub-graphs can be identified whose colours, when rearranged, will result in a different, feasible solution (see fig. 6). Of course, because both the Kempe- and S-chain operators are known to preserve the feasibility of a graph colouring solution [35], then they are applicable to both the basic and extended versions of our graph colouring model.[4]

A notable feature of both the Kempe- and S-chain neighbourhood operators is that the number of vertices involved in a move can vary depending on the initial choices of vertex and colour. For basic (non-extended) round-robin colouring problems, the largest possible move will involve n vertices in the

[4]Note that variants of the Kempe-chain neighbourhood have also been used in some algorithms for the TTP [1, 29, 16], though different names have been used in these cases.



Kempe Chain: Select a vertex and colour randomly (e.g. vertex 1, black) ***S*-chain**: Select a vertex and two colours randomly (e.g. vertex 9, white, black).

Result of Kempe(1, black) All vertices in the associated Kempe-chain (bold lines) have had their colours interchanged

Result of S(9, white, black) All vertices in the associated *S*-chain (bold lines) have had their colours interchanged

Figure 6: Description of Kempe-chain and S-chain operators. In this case $S = 3$.

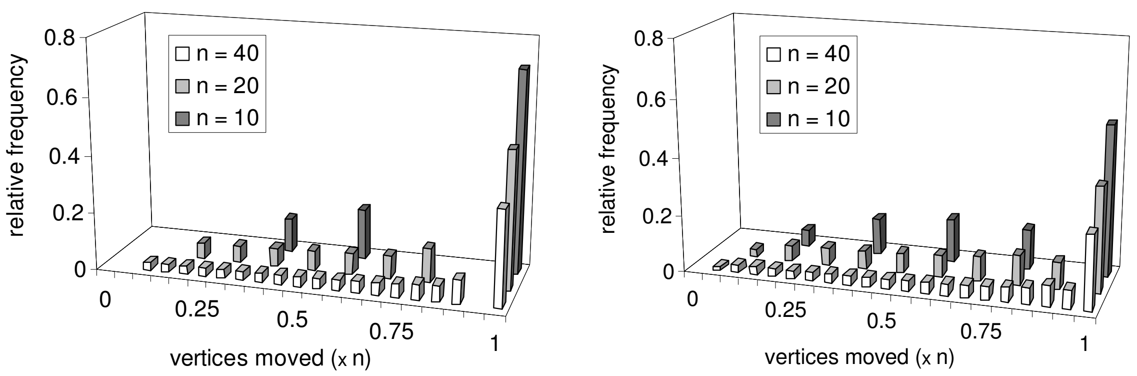


Figure 7: Distribution of different sized Kempe-chain moves for n = 10, 20, and 40 for SRRs (left) and DRRs (right).

case of Kempe-chains (i.e. two colours, with $\frac{n}{2}$ vertices in each), and $S(\frac{n}{2})$ vertices with S-chains. In fig. 7 we illustrate what we have found to be typically shaped probability distributions of the different sized Kempe-chain moves with SRR and DRR problems. These examples were gained by generating initial solutions with our graph colouring algorithms and then performing 10^6 moves, with each move being accepted automatically. We see that in the case of DRRs, the smallest moves possible involve exactly two vertices, which only occurs when a chain is formed containing the complimentary vertices (t_i, t_j) and (t_j, t_i). In this case the Kempe move is equivalent to the operator N_1 (Table 1) and it occurs with a probability $\frac{1}{k-1}$. Note that moves of size two do not occur in the SRR case since a match does not have a corresponding reverse fixture with which to be swapped. The most probable move in both cases involves n vertices – that is, all vertices in the two associated colours. Moves of this size are equivalent to a corresponding move in N_4 and occur when all vertices in the two colours are connected. Such moves appear to be quite probable due to the relatively high edge densities of the graphs, combined with the fact that a greater number of choices for v and c (in $\texttt{Kempe}(v, c)$) will result in the same chain being formed, and therefore the same move taking place. Importantly, however, we see that for larger n's the majority of moves are of sizes in-between these two extremes, resulting in moves that are outside of N_1 and N_4.

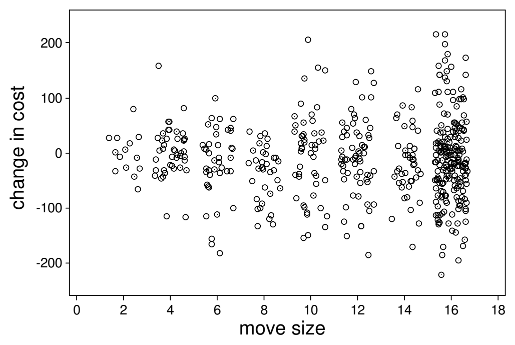
We also performed this same set of experiments using schedules generated by the linear methods mentioned in Section 1. However, due to the structured way in which these methods go about constructing a schedule, we observed that for many different values of n, SRR solutions were produced in which *all* subsequent applications of the Kempe-chain operator were of size n. This is because in these cases the underlying one-factorisations produced by the linear methods are what are known as "perfect one-factorisations" [18, 17] – that is, one-factorisations in which all pairs of one-factors form a Hamiltonian cycle.[5] For DRRs similar observations were also made, though in this case moves of size 2 were also possible and just the remaining proportion of $1 - \frac{1}{k-1}$ moves were of size n. Clearly, such features are undesirable as they do not allow the Kempe-chain operator to produce moves beyond what can already be achieved using N_1 and N_4, limiting the number of schedules accessible via the operator. We should note, however, that we found that this problem could be circumnavigated in some cases by applying neighbourhood operator N_5 to the solution. As previously noted by Ribeiro and Urrutia [39] it would seem that, unlike the other operators detailed in Table 1, N_5 has the potential of breaking up the structural properties of perfect one-factorisations, allowing the Kempe-chain distributions to return to their more "natural" shapes as seen in fig 7. However, we still found cases where this situation was not remedied.[6]

Finally, note that S-chain neighbourhoods were not considered further in our investigations. Due to the high levels of connectivity between different colours, with both types of initial solution generation we observed that over 99% of moves contained the maximum $S(n/2)$ vertices. In other words, almost all moves simply resulted in moves that are also achievable through combinations of N_4.

One of the main reasons why an analysis of moves sizes is relevant is because of the effects that the size of a move can have on the cost of a solution at different stages of the optimisation process. On the one hand, "large" moves can facilitate the exploration of wide expanses of the search space and can provide useful mechanisms for escaping from local optima. On the other hand, when relatively good candidate solutions are being considered, large moves will also be disruptive, usually worsening the quality of a solution as opposed to improving it. These effects are demonstrated in fig. 8 where we illustrate the relationship between the size of a move and the resultant change in an arbitrary cost function. In the left graph, the Kempe-chain operator has been repeatedly applied to a solution that

[5]Specifically, using n-values of between 2 and 1000 (incrementing in steps of 2), we found that 33.6% of solutions produced via these methods resulted in a perfect one-factorisation, with the rate of occurrence remaining fairly consistent as n was increased. For values of $n < 50$ this occurred with SRRs for $n =$ 8, 12, 14, 18, 20, 24, 30, 32, 38, 42, 44, and 48.

[6]Specifically for SRRs with $n =$ 12, 14, 20, 30, and 38.



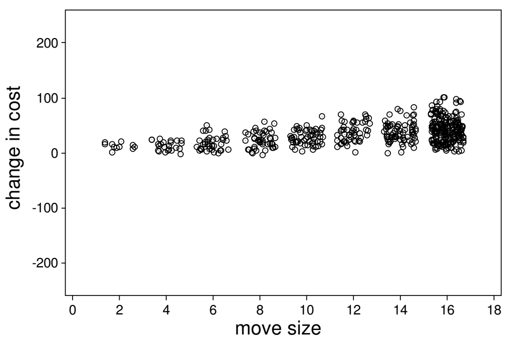


Figure 8: Demonstrating how Kempe-chain moves of different sizes influence the change in cost for a randomly generated solution (left) and a "good" solution (right). In both cases a DRR with $n = 16$ was considered using cost function c_2 defined in Section 6.1 (negative changes thus reflect an improvement).

was randomly produced by one of our graph colouring algorithms. Note that larger moves here tend to give rise to greater variance in cost, but that many moves lead to improvements (in this case 56.6%). In contrast, on the right hand side the effects of the Kempe-chain operator on a relatively "good" solution (which has a cost of approximately quarter of the previous) are demonstrated. Here, larger moves again feature a larger variance in cost, but we also witness a statistically significant medium positive correlation ($r = 0.46$) demonstrating that larger moves tend to be associated with larger decreases in solution quality.

In [16] Di Gaspero and Schaerf have also noted the latter phenomenon (albeit with different neighbourhoods and a different cost function) and suggest a modification to their neighbourhood search algorithm whereby any move above a specific size is automatically rejected with no cost evaluation taking place. Because such moves lead to a degradation in quality in the majority of cases, they find that their algorithm's performance over time is increased by skipping these mostly unnecessary evaluations. On the flip side, of course, such a strategy also eliminates the possibly of "larger" moves occurring which could diversify the search in a useful way, though this is not seen to be an issue in their work.

6 Case Study: Welsh Premiership Rugby

In this section we present an application of the graph colouring-based techniques introduced in this paper for solving a real world sports scheduling problem. The problem that we consider was provided to us by the Welsh Rugby Union (WRU), based at the Millennium Stadium in Cardiff, who are the governing body for all rugby competitions, national and international, in Wales. The particular problem that we are concerned with is the *Principality Premiership* league, which is the highest-level domestic league in the country [47].

The Principality Premiership problem is stated as follows: Each year, there are n teams involved in the league, with each team playing each other twice, once in each other's home venue. A total of $2(n-1)$ weekends throughout the year are allocated for the league, one for each round, and thus a valid, compact DRR is required. In addition, a number of round-specific (hard) constraints are also stipulated, all of which are mandatory in their satisfaction:

Hard Constraint A: Some teams in the league share a home stadium. Thus when one of these teams plays at home, the other team must play away.

Hard Constraint B: Some teams in the league also share their stadia with teams from other leagues and sports. Thus these stadia are not available in certain rounds. (In practice, the other sports teams using these venues have their matches scheduled before the Principality Premiership teams, and so unavailable rounds are known in advance.)

Hard Constraint C: Matches involving rival teams from the same region (so-called "derby matches") also need to be pre-assigned to two specific rounds in the league, corresponding to those falling on the Christmas and Easter weekends.

In addition to the above hard constraints, the league administrators also specify two soft constraints. First, they express a preference for keeping reverse fixtures (i.e. matches (t_i, t_j) and (t_j, t_i)) at least five rounds apart and, if possible, for reverse fixtures to appear in opposite "halves" of the schedule (they do not consider the stricter requirement of "mirroring" to be important, however). Second, they also express a need for all teams to have good home/away patterns, which means avoiding *breaks* (Section 1) wherever possible.

6.1 Solution Methods

Two algorithms were implemented for this scheduling problem, both that employ the strategy of first producing a feasible solution, followed by a period of optimisation via neighbourhood search in which feasibility (i.e. validity, compactness, and adherence to all hard constraints) is maintained. Specific details of these methods are given in Sections 6.1.1 and 6.1.2 below. The algorithms are then compared in Section 6.1.3.

In both cases initial feasible solutions were produced by encoding all of the hard constraints using the extended graph colouring model (Section 4), and then applying one of our graph colouring algorithms. For hard constraint A, if a pair of teams t_i and t_j was specified as sharing a stadium then edges were simply added between all match-vertices corresponding to home games of these teams. For hard constraint B, if a venue was specified as unavailable in a particular round, then edges were added between all match-vertices denoting home games of the venue's team(s) and the associated round-vertex. Finally for hard constraint C, edges were also added between the vertices corresponding to derby matches and all round-vertices except those representing derby weekends.

Details on the specific problem instance faced at the WRU are given in bold in Table 2. To aid our analysis we also generated (artificially) a further nine instances of comparable size and difficulty, details of which are also given in the table. As it turned out, we found that it was quite straightforward to find a feasible solution to the WRU problem using the three stochastic graph colouring algorithms from Section 3 (less than 10 seconds of CPU time required) and so for the purposes of our experiments we ensured that all artificially generated problems also featured at least one feasible solution. However, the minimum number of soft-constraint violations achievable in these problems is not known. Finally, note that our commercial solving software XPress was not able to achieve feasibility with any of these instances using the 600-second time limit from Section 3.

In our methods the soft constraints of the problem were captured in two cost functions, both of which need to be minimised:

Spread Cost (c_1)**:** Here, a penalty of 1 is added each time a match (t_i, t_j) and its return fixture (t_j, t_i) are scheduled to rounds r_p and r_q, such that $|p - q| \leq 5$. In addition, a penalty of $\frac{1}{0.5n(n-1)}$ is also added each time matches (t_i, t_j) and (t_j, t_i) are scheduled to occur in the same half of the schedule.

Table 2: Summary of the problem instances used. The entry in bold refers to the real-world WRU problem

#	n	$\lvert V\rvert$	Edge Density	A[a]	B[b]	C[c]
1	12	154	0.268	0	2 {5, 5}	3
2	12	154	0.292	1	3 {6, 8, 10}	6
3	12	154	0.308	2	4 {3, 6, 8, 10}	6
4	14	208	0.236	0	2 {4, 5}	4
5	**14**	**208**	**0.260**	**1**	**3 {8, 10, 10}**	**7**
6	14	208	0.271	2	5 {3, 6, 8, 10, 10}	7
7	16	270	0.219	1	3 {4, 5, 6}	5
8	16	270	0.237	2	5 {3, 6, 8, 10, 10}	8
9	18	340	0.194	1	3 {4, 5, 6}	6
10	18	340	0.212	2	6 {4, 5, 6, 7, 10, 10}	9

[a]Number of pairs of teams sharing a stadium

[b]Number of teams sharing a stadium with teams from another league/sport. The number of match-unavailability constraints for each of these teams is given in { }'s.

[c]Number of local derby pairings

Break Cost (c_2)**:** Here, the home/away pattern of each team is analysed in turn and penalties of b^l are incurred for each occurrence of l consecutive breaks. In other words, if a team is required to play two home (or away) games in succession, this is considered as one break and incurs a penalty of b^1. If a team has three consecutive home (or away) games, this is considered as two consecutive breaks and results in a penalty of b^2 being added, and so on.

The unit $\frac{1}{0.5n(n-1)}$ is used as part of c_1 to ensure that the total penalty due to match pairs occurring in the same half is never greater than one, thus placing a greater emphasis on keeping matches and their return fixtures at least five rounds apart, which was deemed appropriate in this case. The penalty unit of b^l in cost function c_2 is also used to help discourage long breaks from occurring in the schedule. (In our case, we chose to use $b = 2$: thus a penalty of 2 was incurred for single breaks, 4 for double breaks, 8 for triple breaks, and so on.)

It is notable that since the above cost functions measure different characteristics, and since they also use different penalty units and feature different growth rates, they are in some sense incommensurable. For this reason it is appropriate to treat them separately, and we therefore use the concept of *dominance* in order to distinguish between solutions. Consequently, during the execution of both algorithms a list is maintained that contains all non-dominated solutions found so far. In our case we specifically chose to use the concept of *loose dominance* [42], which works as follows: given a solution S with cost values $c_1(S)$ and $c_2(S)$, S is said to dominate another solution T if and only if:

- $c_1(S) \leq c_1(T)$ and $c_2(S) < c_2(T)$; or
- $c_1(S) < c_1(T)$ and $c_2(S) \leq c_2(T)$.

The output to both algorithms is thus a list L containing one or more solutions that are not dominated by any other solutions encountered during the search. Note that the concept of dominance is commonly used in the field of multiobjective optimisation where, in addition to incommensurability, cost functions may also be in conflict with one-another (that is, an improvement in one cost will tend to invoke the worsening of another). It is unclear whether the two cost functions used here are necessarily in conflict, however.

```
     Multi-Stage Algorithm ($S$)
(1)  while (not stopping condition) do
(2)      Apply perturbation to $S$
(3)      Reduce cost $c_1$ in $S$ via a hill climbing procedure
(4)      Reduce cost $c_2$ in $S$ via simulated annealing
(5)      Update list $L$ of non-dominated solutions
```

Figure 9: The Multi-Stage Algorithm. The procedure's takes as input a feasible solution S provided by any graph colouring method.

6.1.1 The Multi-Stage Approach

Our first method for the WRU problem operates in a series of stages, with each stage being concerned with the elimination of one type of soft constraint. A description of this method is given in fig. 9.

The multi-stage algorithm starts by taking an arbitrary initial solution S produced via our graph colouring methods. This solution is then "perturbed" by performing a series of Kempe-chain moves, paying no heed to either cost function. Next, a hill-climbing procedure is applied that attempts to make reductions in the spread cost c_1. This is done by performing a random Kempe-chain move at each iteration, and accepting the move only if the new spread cost is less than or equal to the current spread cost. On completion of the hill-climbing procedure, attempts are then made to reduce the break cost c_2 of the current schedule *without increasing the current spread cost.* This is achieved via a stage of simulated annealing using a restricted neighbourhood operator where only matches and their reverse fixtures (i.e. (t_i, t_j) and (t_j, t_i)) can be swapped. Note that the latter moves can, on occasion, violate some of the additional hard constraints of this problem, and so in these cases such moves are rejected automatically. Also note that moves in this restricted neighbourhood do not alter the spread cost of the schedule and therefore do not undo any work done in the previous hill-climbing stage. On completion, the best solution found during the round of simulated annealing is used to update the list of non-dominated solutions, if applicable, before the entire process is repeated.

Our choice of a hill-climbing procedure for reducing c_1 arose simply because in initial experiments we observed that, in isolation, the associated soft constraints seemed quite easy to satisfy. Thus a simple hill-climbing procedure proved effective for making quick and significant gains in quality (for all instances spread costs of < 1 (and often $= 0$) were nearly always achieved within our imposed cut-off point for this procedure, which was $10,000$ evaluations). In addition to this we also noticed that only short execution times were needed for the simulated annealing stage due to the relatively small search space resulting from the restricted neighbourhood operator, which meant that the search would tend to converge quite quickly at a local minimum. In preliminary experiments we also found that if we lengthened the simulated annealing process by allowing the temperature variable to be reset (thus allowing the search to escape these optima), then the very same optimum would be achieved after another period of search, perhaps suggesting that the convergence points in these searches are the true optima in these particular spaces.[7]

Finally, our use of a perturbation operator in the multi-stage algorithm is intended to encourage a diversification in the search. In this case a balance needs to be struck between applying enough changes to the current solution to cause the search to enter a different part of the search space, but not applying too many changes so that the operator becomes nothing more than a random restart

[7]In all cases we used an initial temperature $t_0 = 20$, and a temperature update rule of $t_{i+1} = \alpha t_i$ with $\alpha = 0.99$. A total of $\frac{|V|}{2}$ moves were attempted at each temperature, and the annealing process ended when no move was accepted for 20 successive temperatures. Such parameters were decided upon in preliminary testing and were not seen to be critical in dictating algorithm performance.

```
     **Multiobjective Algorithm** $(S, \Delta w)$
(1)  Set reference costs $r_1$ and $r_2$
(2)  Specify initial weights $w_1$ and $w_2$
(3)  Calculate weighted cost of solution, $f(S) = w_1c_1(S) + w_2c_2(S)$
(4)  $B \leftarrow f(S)$
(5)  while (not stopping condition) do
(6)      Form new solution $S'$ by applying Kempe-chain move to $S$
(7)      if $(f(S') \leq f(S)$ or $(f(S') \leq B)$ then
(8)          $S \leftarrow S'$
(9)          Update list $L$ of non-dominated solutions
(10)     Find $i$ corresponding to $\max_{i\in\{1,2\}}(\frac{c_1(S)}{r_1}, \frac{c_2(S)}{r_2})$
(11)     Increase weight $w_i \leftarrow w_i(1 + \Delta w)$
```

Figure 10: Multiobjective Algorithm with Variable Weights [37]. In all experiments reported here, a setting of $\Delta w = 10^{-6}$ was used.

mechanism. In our case, we chose to simply apply the Kempe-chain operator five times in succession, which proved sufficient for our purposes.

6.1.2 A Multiobjective Optimisation Approach

In contrast to the multi-stage approach, our second method attempts to eliminate violations of both types of soft constraint simultaneously. This is done by combining both cost functions into a single weighted objective function $f(S) = w_1c_1(S) + w_2c_2(S)$, used in conjunction with the Kempe chain neighbourhood operator.

An obvious issue with the function f is that suitable values need to be assigned to the weights. Such assignments can, of course, have large effects on the performance of an algorithm, but they are not always easy to determine as they depend on many factors such as the size and type of problem instance, the nature of the individual cost functions, the user requirements, and the amount of available run time. To deal with this issue, we choose to adopt the multiobjective optimisation technique of Petrovic and Bykov [37], which was originally proposed for use with exam timetabling. The basic strategy of this approach is to alter weights dynamically during the search based on the quality of solutions that have been found so far, thus directing the search into specific regions of the search space. This is achieved by providing two *reference costs* to the algorithm, r_1 and r_2. Using these values, we can then imagine a reference point (r_1, r_2) being plotted in a two-dimensional Cartesian space, with a straight *reference line* then being drawn from the origin $(0, 0)$ and through the reference point (see fig. 11). During the search, all solutions encountered are then also represented as points in this Cartesian space and, at each iteration, the weights are adjusted automatically to encourage the search to move towards the origin while remaining close to the reference line. It is hoped that eventually solutions will be produced that feature costs less than the original reference costs.

A pseudocode description of this approach is given in fig. 10. Note that the weight update mechanism used here (line (11)) means that weights are gradually increased during the run. Since, according to line (7), changes to solutions are only permitted if (a) they improve the cost, or (b) if the weighted cost is kept below a constant B, this implies that worsening moves become increasingly less likely during execution. Thus the search process is analogous to metaheuristics such as the great deluge algorithm and simulated annealing. Note that different acceptance criteria, and methods of updating the weights could also be used in practice, though we choose to keep our methods similar to those proposed in the original publication. We also decided to follow the authors' guidelines of determining the initial settings for the weights, using the rule $w_i = \frac{c_i(S_{\text{init}})}{r_i}$ for $i \in \{1, 2\}$.

6.1.3 Experimental Analysis

To compare the performance of the two approaches outlined above, we performed 100 runs of each algorithm on each of the ten problem instances. In all cases a cut off point of 20,000,000 evaluations (of either cost function) was used, resulting in run times in the range of 450 to 1300 CPU seconds. For simplicity's sake, all initial (feasible) solutions were generated using the HC algorithm, though this was essentially an arbitrary choice. Both algorithms were also coded in C++ and were compiled and executed under the same conditions as our previous experiments.

Note that, unlike the multi-stage approach, reference costs r_1 and r_2 need to be supplied to the multiobjective approach. Petrovic and Bykov [37] suggest that such values might be determined by evaluating a solution from another source, such as a manually produced solution, or a solution found by some other automated method. In our case we chose suitable reference costs by performing one run using the multi-stage approach. We then used the costs of the resultant solution as the reference costs for the multiobjective algorithm. In runs where more than one non-dominated solution was produced, solutions in L were sorted according to their costs, and the costs of the median solution were used. Note that because problem instance 5 is based on real world data, in this case we were also able to use the costs of the (manually produced) league schedule used in the competition. However, runs using the reference costs from either source turned out to be similar in practice, and so we only report results achieved using the former method here.

Before assessing the quality of solutions produced by the two algorithms, we first examine the general behaviour of each method. Figure 11 displays the costs of a sample of solutions encountered in an example run with each algorithm. From the initial solution at point $(22.4, 686)$ we see that the multi-stage approach quickly finds solutions with very low values of cost c_1. It then spends the remainder of the run attempting to make reductions to c_2, which is the cause of the bunching of the points in the left of the graph. Due to the position of the reference line (the dotted line in the figure), the multiobjective approach follows a similar pattern to this, though initial progress is considerably slower. However, as the search nears the reference line, the algorithm then considers c_1 to have dropped to an appropriate level and so weight w_2 (used in evaluation function f (fig. 10)) begins to increase such that reductions to cost c_2 are also sought. In this latter stage, unlike the multi-stage approach, improvements to both cost functions are being made simultaneously. As both algorithms' searches converge, we see from the projection (inset) that the multiobjective approach produces the best solutions in this case.

In order to give a more clear sense of time, fig. 12 also demonstrates how c_1 and c_2 alter during execution according to the number of evaluations. Here, we see that the multi-stage algorithm obtains good solutions rapidly, but that the search also quickly stagnates. This stagnation is due to the way in which initial solutions are generated for the simulated annealing phase (step 4 (fig. 9)), which are effectively random with regards to cost function c_2, having been produced by the previous hill-climbing phase. Note that due to the restrictiveness of the search space, the quality of result achieved by each phase of simulated annealing seems dependent on the starting solution; thus, further improvements to c_2 only occur when favourable features occur, by chance, in the initial solutions of the simulated annealing phase. This results in slow progress in general. In contrast to the multi-stage approach, the multiobjective algorithm makes much slower progress in initial stages (indeed, significant reductions to c_2 are only made in the second half of the run), but in the latter stages improvements are made that result in a superior solution.

A summary of the results over all ten test problems is provided in Table 3. The "best" column here shows the costs of the best solution found in any of the runs with either algorithm, which we use

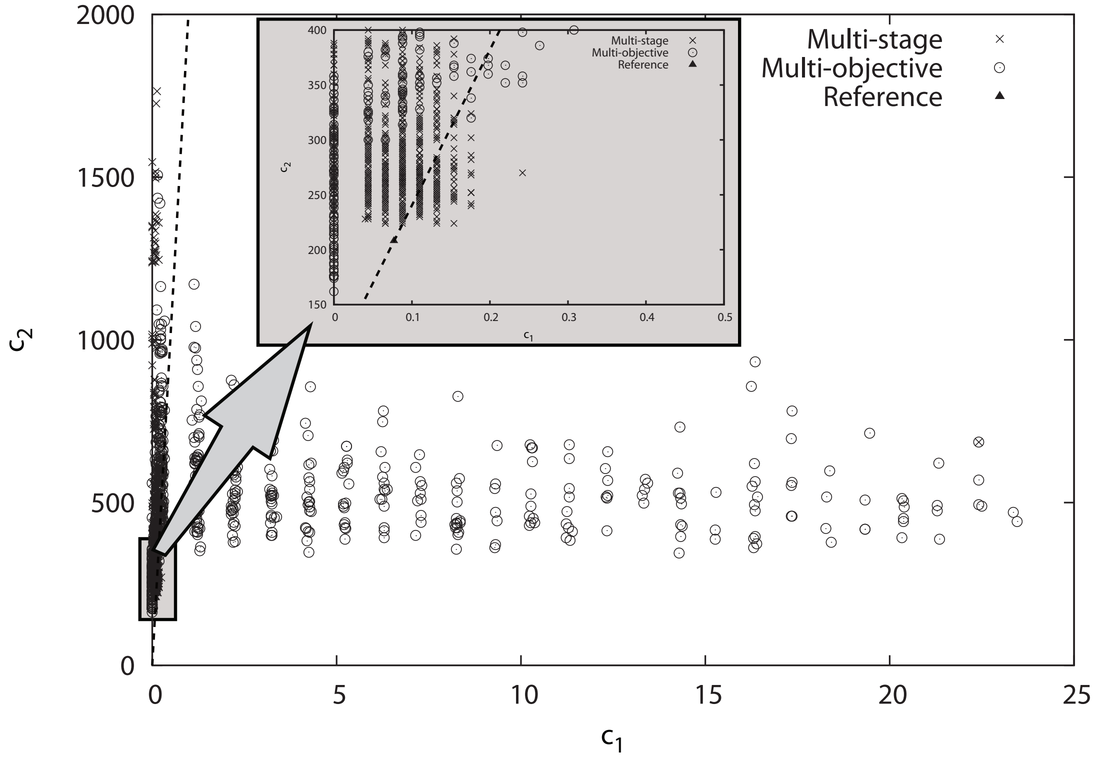


Figure 11: Costs of solutions encountered by the multi-stage and multiobjective approaches in one run with problem instance 5. Solution costs were recorded every 10,000 evaluations. The dotted line (in both the main graph and the projection) represents the reference line used by the multiobjective approach and is drawn from the origin and through the reference point.

for comparative purposes. Note that in all instances except problem 2 only one best result has been produced – that is, we have not witnessed any obvious trade-off in the two cost functions. The values in the column labelled "maximum" are used for normalisation purposes, and represent the highest cost values seen in any solution returned by either algorithm. For both algorithms, the table then displays three statistics: the mean CPU time of the runs, the mean size of the non-dominated solution lists L that were returned, and the distances between each of the solutions in L and the "best" solutions. This latter performance measure is based on the metric suggested by Deb et al. [12], which assesses the performance an algorithm by considering the distance between the costs of each solution in L and the costs of some optimal (or near-optimal) solution to the problem. Because the costs of the global optima are not known for these instances, we choose to use the values given in the "best" column as approximations to this. In our case distances were calculated as follows. First, the costs of all solutions returned by the algorithms, in addition to the costs of the "best" solutions, were normalised to values in $[0, 1]$ by dividing by the maximum cost values specified in the table. Next, for each solution, the (Euclidean) distance between the normalised best costs and all normalised solution costs was then calculated. The mean, standard deviation and median of all distances returned for each algorithm was then recorded.

The results in Table 3 can be split into two cases. The first involves the larger problem instances 3 to 10 (which, we remember, includes the real-world league scheduling problem). Here, we see that the multiobjective approach consistently produces better results than the multi-stage approach, which is

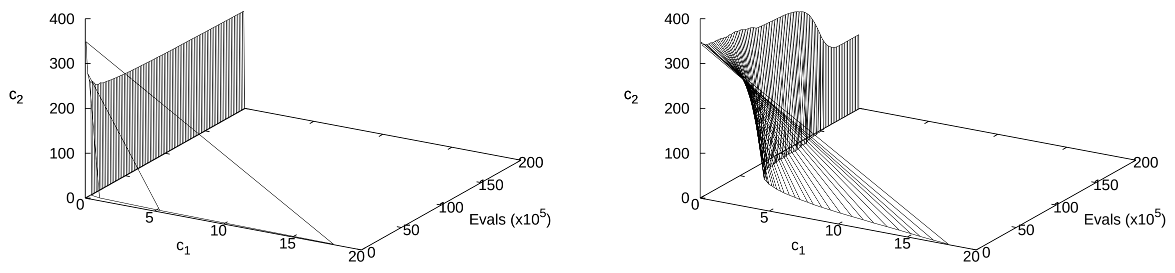


Figure 12: Demonstration of the changes in costs according to the number of evaluations for the multi-stage (left) and multiobjective (right) algorithms. The lines in both figures represent the costs of the median solution from the non-dominated set of solutions, averaged across 100 different runs with problem instance 5

Table 3: Summary of the Results achieved by the Multi-Stage and Multiobjective Algorithms. An asterisk (*) in the "best" column indicates that the solution with the associated costs was found using the Multi-stage approach (otherwise it was found by the multiobjective approach).

				Multi-Stage				**Multiobjective**			
	Best	Maximum				Distance from Best				Distance from Best	
#	(c_1, c_2)	c_1	c_2	CPU	$\|L\|$	$\mu \pm \sigma$	Med.	CPU	$\|L\|$	$\mu \pm \sigma$	Med.
1	(0, 104)*	4.21	352	557.7	1.81	0.15 ± 0.22	0.04	524.7	2.57	0.27 ± 0.10	0.27
2	(0, 134)*	11.3	394	533.8	2.81	0.06 ± 0.05	0.05	531.7	3.82	0.20 ± 0.11	0.17
	(2.15, 132)*										
3	(0, 102)	8.24	560	467.8	4.32	0.24 ± 0.17	0.17	515.9	3.32	0.13 ± 0.10	0.12
4	(0, 112)	2.17	186	749.2	2.72	0.28 ± 0.09	0.26	655.1	1.00	0.14 ± 0.06	0.13
5	(0, 134)	3.29	294	711.8	3.61	0.34 ± 0.11	0.31	662.4	1.00	0.11 ± 0.04	0.11
6	(0, 148)	4.26	444	646.0	4.44	0.32 ± 0.11	0.30	666.4	1.04	0.08 ± 0.05	0.08
7	(0, 156)	2.18	816	994.0	4.51	0.19 ± 0.11	0.16	830.2	1.04	0.05 ± 0.04	0.04
8	(0, 186)	3.27	786	916.6	4.71	0.26 ± 0.12	0.22	822.8	1.53	0.08 ± 0.06	0.06
9	(0, 204)	1.26	940	1290.0	4.32	0.21 ± 0.10	0.19	1007.3	1.08	0.05 ± 0.06	0.04
10	(0, 234)	2.29	1398	1226.0	4.63	0.20 ± 0.10	0.16	1007.0	1.30	0.09 ± 0.05	0.07

reflected in the lower means, medians, and deviations in the Distance from Best column. Note that the best results for these instances have also come from the multiobjective approach. We also see that the multi-stage approach has produced larger solution lists for these problem instances, which could be useful if a user wanted to be presented with a choice of solutions, though the solutions in these lists are of lower quality in general. Also note that the differences between the mean and median values here reveal that the distributions of distances with the multi-stage approach feature larger amounts of positive skew, reflecting the fact that this method produced solutions of very low quality on occasion. Meanwhile, for the smaller problem instances 1 and 2 we see that these patterns are more-or-less reversed with the multi-stage approach producing solutions with costs that are consistently closer (or equal to) to the best known costs. One feature to note in this case is the relatively large difference between the mean and median with the multi-stage approach for instance 1, where the we saw about 60% of produced solutions produced being very close to the best, with the remaining 40% having much larger distances. Finally, note that for all problem instances, the non-parametric Mann-Witney test indicates that the median distances of each algorithm are significantly different with significance level $\leq 0.01\%$.

In summary, the results in Table 3 suggest that the strategy used by the multi-stage algorithm of employing many rounds of short intensive searches seems more fitting for smaller, less constrained instances, but for larger instances, including the real world problem instance, better solutions are achieved by using the multiobjective approach where longer, less intensive searches are performed.

7 Conclusions

In this paper we have shown that in many cases solutions to round-robin scheduling problems can be produced via the utilisation of well-known graph colouring principals. In particular, we have seen that three existing, heuristic-based graph colouring methods have been able to construct valid compact round-robin schedules, often in the presence of a large number of additional hard constraints. However, we have also noted areas in which some of these methods seem to struggle, such as when tackling very large instances, or those in the noted phase transition regions. In future work it would be interesting to investigate how higher success rates (and lower computation times) could be achieved: for example, modifications could be made to these methods that allow them to better exploit the known equipartite structure of the graphs.

In the second half of this paper, we have exploited the links between round-robin sports scheduling and graph colouring by proposing two algorithms for a real-world sports scheduling problem. In both cases the three graph colouring algorithms were able to achieve the main aim of producing initial feasible solutions to these problems in short amounts of time. In contrast, an IP model of these problems used in conjunction with a commercial solver could not achieve feasibility within a generous time limit. The underlying graph colouring model was then used to help define neighbourhood operators that guarantee never to re-violate the hard constraints of the considered problems. One of these approaches also constitutes what is, to our knowledge, the first application of multiobjective techniques to a sports scheduling problem. In the case of the real-world problem instance (number 5), we found that more than 98% of all solutions generated by our multiobjective approach dominated the solution that was manually produced by the WRU's league administrators. For the multi-stage approach this figure was just 0.02%. We must note, however, that these particular statistics should be interpreted with care, firstly because the manually produced solution was actually for a slightly different problem (the exact specifications of which we were unable to obtain from the league organisers), and second because our specific cost functions were not previously used by the league organisers for evaluating their solutions.

We believe that the methods proposed here are quite generic and in the future we plan to apply these techniques to other practical sports scheduling problems. We also plan to try and gain deeper insights into the connectivity of the underlying search spaces of such problems, allowing us to better understand the overall potential of neighbourhood search-based approaches. To aid further investigation we have posted the ten problem instances used in Section 6 online[8] and we invite other researchers interested in this area to use them in their own research.

References

[1] A. Anagnostopoulos, L. Michel, P. van Hentenryck, and Y. Vergados. A simulated annealing approach to the traveling tournament problem. *Journal of Scheduling*, 9(2):177–193, 2006.

[2] I. Anderson. Kirkman and GK_2n. *Bulletin of the I.C.A*, 3:111–112, 1991.

[3] I. Anderson. *Combinatorial Designs and Tournaments.* Oxford University Press, 1997.

[4] I. Anderson. Balancing carry-over effects in tournaments. In *Combinatorial designs and their applications*, volume 403, pages 1–16, Boca Ranton, FL, 1999. Chapman and Hall/CRC Res. Notes Math.

[5] N. Balakrishnan. Examination scheduling: a computerized application. *Omega*, 19(1):37–41, 1991.

[6] T. Bartsch, A. Drexl, and S. Kröger. Scheduling the professional soccer leagues of austria and germany. *Computers and Operations Research*, 33(7):1907–1937, 2006.

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

[8] E. Burke, D. de Werra, and J. Kingston. *Handbook of Graph Theory.*, chapter Applications to timetabling, pages 445–474. CRC Press, 2003.

[9] P. Cheeseman, B. Kanefsky, and W. Taylor. Where the really hard problems are. In *Proceedings of IJCAI-91*, pages 331–337, 1991.

[10] J. Culberson and F. Luo. Exploring the k-colorable landscape with iterated greedy. In D. S. Johnson and M. A. Trick, editors, *Cliques, Coloring, and Satisfiability: Second DIMACS Implementation Challenge*, volume 26, pages 245–284. American Mathematical Society, 1996.

[11] D. de Werra. Some models of graphs for scheduling sports competitions. *Discrete Applied Mathematics*, 21:47–65, 1988.

[12] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast elitist multi-objective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197, 2000.

[13] F. della Croce and D. Oliveri. Scheduling the Italian football league: an ILP-based approach. *Computers and Operations Research*, 33(7):1963–1974, 2006.

[14] F. della Croce, R. Tadei, and P. Asioli. Scheduling a round-robin tennis tournament under courts and players unavailability constraints. *Annals of Operational Research*, 92:349–361, 1999.

[15] L. Di Gaspero and A. Schaerf. Neighborhood portfolio approach for local search applied to timetabling problems. *Journal of Mathematical Modeling and Algorithms*, 5(1):65–89, 2006.

[16] L Di Gaspero and A. Schaerf. A composite-neighborhood tabu search approach to the traveling tournament problem. *Journal of Heuristics*, 13(2):189–207, 2007.

[17] J. Dinitz and D. Garnick. There are 23 non-isomorphic perfect one-factorisations of K_{14}. *Journal of Combinatorial Designs*, 4(1):1–4, 1996.

[18] J. Dinitz, D. Garnick, and B. McKay. There are 526,915,620 nonisomorphic one-factorizations of K_{12}. *Journal of Combinatorial Designs 2*, 2:273–285, 1994.

[8] `www.rhydlewis.eu`

[19] K. Easton, G. Nemhauser, and M. Trick. The traveling tournament problem: description and benchmarks. In T. Walsh, editor, *Principles and Practice of Constraint Programming*, volume 2239 of *Lecture Notes in Computer Science*, pages 580–585. Springer, Berlin/Heidelberg, 2001.

[20] K. Easton, G. Nemhauser, and M. Trick. Solving the traveling tournament problem: A combined integer programming and constraint programming approach. In E. Burke and P. De Causmaecker, editors, *Practice and Theory of Automated Timetabling (PATAT) IV*, volume 2740 of *Lecture Notes in Computer Science*, pages 100–109. Springer, 2003.

[21] M. Elf, M. Jünger, and G. Rinaldi. Minizing breaks by maximizing cuts. *Operations Research Letters*, 31(5):343–349, 2003.

[22] C. Fleurent and J. Ferland. Allocating games for the NHL using integer programming. *Operations Research*, 41(4):649–654, 1993.

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

[24] M. Henz, T. Müller, and S. Thiel. Global constraints for round robin tournament scheduling. *European Journal of Operational Research*, 153:92–101, 2004.

[25] G. Kendall, S. Knust, C. Ribeiro, and S. Urrutia. Scheduling in sports, an annotated bibliography. *Computers and Operations Research*, 37(1):1–19, 2010.

[26] T. Kirkman. On a problem in combinations. *Cambridge Dublin Math Journal*, 2:191–204, 1847.

[27] R. Lewis. A general-purpose hill-climbing method for order independent minimum grouping problems: A case study in graph colouring and bin packing. *Computers and Operations Research*, 36(7):2295–2310, 2009.

[28] R. Lewis, Paechter B., and O. Rossi-Doria. Metaheuristics for university course timetabling. In K. Dahal, K. Tan, and P. Cowling, editors, *Evolutionary Scheduling*, volume 49 of *Studies in Computational Intelligence*, pages 237–272. Springer-Verlag, Berlin, 2006.

[29] A. Lim, B. Rodrigues, and X. Zhang. A simulated annealing and hill-climbing algorithm for the traveling tournament problem. *European Journal of Operational Research*, 174(3):1459–1478, 2006.

[30] A. Mehrotra and M. Trick. A column generation approach for graph coloring. *INFORMS Journal on Computing*, 8:344–354, 1996.

[31] E. Mendelsohn and A. Rosa. One-factorizations of the complete graph – a survey. *Journal of Graph Theory*, 9:43–65, 1985.

[32] R. Miyashiro and T. Matsui. A polynomial-time algorithm to find an equitable home-away assignment. *Operations Research Letters*, 33:235–241, 2005.

[33] R. Miyashiro and T. Matsui. Minimizing the carry-over effects value in a round-robin tournament. In E. Burke and H. Rudova, editors, *PATAT '06 Proceedings of the 6th International Conference on the Practice and Theory of Automated Timetabling*, pages 460–464, 2006.

[34] R. Miyashiro and T. Matsui. Semidefinite programming based approaches to the break minimization problem. *Computers and Operations Research*, 33(7):1975–1992, 2006.

[35] C. Morgenstern. *Algorithms for General Graph Coloring.* PhD thesis, University of New Mexico, 1989.

[36] G. L. Nemhauser and M. A. Trick. Scheduling a major college basketball conference. *Operations Research*, 46:1–8, 1998.

[37] S. Petrovic and Y. Bykov. A multiobjective optimisation approach for exam timetabling based on trajectories. In E Burke and P. De Causmaecker, editors, *The Paractice and Theory of Automated Timetabling (PATAT) IV*, volume 2740, pages 181–194. Springer-Verlag, Berlin, 2003.

[38] R. Rasmussen and A. Trick. Round robin scheduling – a survey. *European Journal of Operational Research*, 188:617–636, 2008.

[39] C. Ribeiro and S. Urrutia. Heuristics for the mirrored travelling tournament problem. *European Journal of Operational Research*, 179(3):775–787, 2007.

[40] K. Russell. Balancing carry-over effects in round-robin tournaments. *Biometrika*, 67(1):127–131, 1980.

[41] R. Russell and J. Leung. Devising a cost effective schedule for a baseball league. *Operations Research*, 42(4):614–625, 1994.

[42] R. Steuer. *Multiple Criteria Optimization: Theory, Computation and Application.* John Wiley & Sons, 1986.

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

[44] J. Thompson and K. Dowsland. An improved ant colony optimisation heuristic for graph colouring. *Discrete Applied Mathematics*, 156:313–324, 2008.

[45] M. A. Trick. A schedule-then-break approach to sports timetables. In E. Burke and W. Erben (editors), editors, *The Practice and Theory of Automated Timetabling III*, volume 2079 of *Lecture Notes in Computer Science*, pages 242–253. Springer-Verlag, 2001.

[46] J. S. Turner. Almost all k-colorable graphs are easy to color. *Journal of Algorithms*, 9:63–82, 1988.

[47] The Welsh Rugby Union. http://www.wru.co.uk/.

[48] M. Wright. Timetabling county cricket fixtures using a form of tabu search. *Journal of the Operational Research Society*, 47(7):758–770, 1994.

[49] M. Wright. Scheduling fixtures for basketball New Zealand. *Computers and Operations Research*, 33(7):1875–1893, 2006.

A IP Formulation used for Colouring Round-Robin Graphs

The following gives the IP formulation for graph colouring used in our experiments, which is based on a model outlined by Mehrotra and Trick [30]. Given an undirected graph $G = (V, E)$ and a fixed number of colours k, let x_{vj}, $v \in V$, $1 \leq j \leq k$ be a binary variable such that:

$$x_{vj} = \begin{cases} 1 & \text{if vertex } v \text{ is assigned to colour } j \\ 0 & \text{otherwise.} \end{cases} \tag{3}$$

The objective is to thus find an assignment to each binary variable such that:

$$x_{uj} + x_{vj} \leq 1 \quad \forall (u, v) \in E, \forall j \tag{4}$$

$$\textstyle\sum_{j=1}^{k} x_{vj} = 1 \quad \forall v \in V. \tag{5}$$

$$\textstyle\sum_{v \in V} x_{vj} = \frac{|V|}{k} \quad \forall j. \tag{6}$$

Thus the aim is to achieve a solution that ensures (a) that no pair of adjacent vertices are assigned to the same colour (eq. (4)), and (b) that all vertices are assigned to exactly one colour (eq. (5)). For the round-robin graphs, which are equipartite in nature, we are also able to impose eq. (6), which specifies that exactly $|V|/k$ vertices should be assigned to each colour. In these cases, we are also able to impose a tight lower bound of $k = \chi$.

Finally, in our experiments, we also tried a second IP model similar to the above, but which aimed at maximising the number of vertices to which a colour had been assigned (i.e. the model allowed vertices to be "uncoloured"). This involved replacing the equals signs in eqs. (5) and (6) by less-than-or-equal signs and then trying to maximise the following objective function f:

$$f = \sum_{v \in V} \sum_{j=1}^{k} x_{vj}. \tag{7}$$

Note however, that when input into Xpress, we found this latter model to exhibit inferior performance than the previous model. Thus all IP experiments mentioned in the paper refer to the first model.