A Real-World Employee Scheduling and Routing Application

Emma Hart
Institute for Informatics and Digital Innovation
Edinburgh Napier University
Edinburgh, Scotland, UK
e.hart@napier.ac.uk

Kevin Sim
Institute for Informatics and Digital Innovation
Edinburgh Napier University
Edinburgh, Scotland, UK
k.sim@napier.ac.uk

Neil Urquhart
Institute for Informatics and Digital Innovation
Edinburgh Napier University
Edinburgh, Scotland, UK
n.urquhart@napier.ac.uk

ABSTRACT

We describe a hyper-heuristic application developed for a client to find quick, acceptable solutions to Workforce Scheduling and Routing problems. An interactive fitness function controlled by the user enables five different objectives to be weighted according to client preference. The application uses a real road network in order to calculate driving distances between locations, and is designed to integrate with a web-based application to access employee calendars.

Categories and Subject Descriptors

Computing Methodologies [**Artificial Intelligence**]: Problem Solving, Control Methods and Search

Keywords

Hyper-heuristics; interactive fitness; scheduling and routing

1. INTRODUCTION

The Workforce Scheduling and Routing Problem (WSRP) was defined in [1] as a scenario that involves the mobilisation of personnel in order to perform work related activities at different locations. Personnel are considered *flexible* in that they have variable working patterns, and *mobile* in that travelling is required between their home base and jobs, and in that travelling may involve significant times. The problem thus combines features of vehicle routing problems (VRP) that attempt to reduce distance travelled, and personnel scheduling problems that allocate work to staff to ensure that all required tasks are completed whilst respecting working patterns.

We describe software developed for a client who sell a range of software products to large companies. The client wishes to 'add value' to existing products by improving their current software to include a cheap method of creating schedules in typical WRSP situations that include time-windows, service-times and employees based from home. Rather than provide a specialised WRSP algorithm customised to each company, they wish to develop a generic algorithm that will be widely applicable in accouting for common constraints and practices, whilst recognising that some companies might operate practices that fall outside of the specification. Specifically, the aim of the software is not to search for *global* optima but to provide a simple tool that on the one hand automates the scheduling process, and on the other, provide a decision support tool for an operator. The software we developed makes use of a hyper-heuristic approach to developing a solution, and utilises an interactive fitness function to produce a single solution.

GECCO'14, July 12–16, Vancouver, BC, Canada.

http://dx.doi.org/10.1145/2598394.2605447.

2. PROBLEM DESCRIPTION

The brief required that the algorithm scheduled a set of engineers to servicing jobs that occurred at locations across Scotland. Schedules were planned for a period of one month at a time. Each of the jobs to be serviced has a planned schedule date, a *pre-schedule* tolerance and a *post-schedule* tolerance defining the earliest and latest dates the job might be scheduled, typically up to three days either side of the planned date. A company can specify whether it is possible to 'break' time-window constraints by specifying an allowable number of days. Each job has a processing time, typically between 1 and 4 hours. Each job also has a driving time which is dependent on the last position of the worker assigned to the job.

The company employs n workers, each based at their home locations which can be anywhere in Scotland. Each worker has an associated calendar that specifies their working hours over a period of one month, which may include holidays. Individual employee availability is designed to be read via a web interface that uses the Calendaring Extensions to WebDAV (CalDAV) format. This is an Internet standard allowing a client to access scheduling information on a remote server. It extends WebDAV (HTTP-based protocol for data manipulation) specification and uses iCalendar format for the data, allowing multiple client access to the same information thus allowing cooperative planning and information sharing. Availability may vary significantly across employees.

Workers always start working from home and should return to their home base at the end of each day. Some companies have a policy that a working day does not include travel time to/from the home location, while others require that travel time is factored in, therefore the algorithm must account for individual preferences. Some companies allow overtime, specifying the maximum number of overtime hours allowed.

3. ROUTING

Routing is performed using the GraphHopper library [2]. This application offers memory efficient algorithms written in Java for routing between two locations on a graph (e.g. Dijkstra and A*) using road network data provided by OpenStreetMap [3]. The OSM XML data is pre-processed into a street graph that takes account of turn and direction restrictions within the road network. GraphHopper runs on a wide variety of platforms and due to its use of Apache license can be embedded within products, making it an ideal choice for a client.

We make use of the inbuilt bidirected A* algorithm to find routes through the street graph from one latitude/longitude point to another. The client supplies data in the form of geocodes (longitude and latitude) — however, postcodes can also be used if required by looking up data released by Ordnance Survey (Crown Copyright) which contains a complete list of Postcodes and their latitute/longitude position. GraphHopper returns both the driving time and driving distance between the two locations, although we only make use of the time in the described application.

4. OBJECTIVE FUNCTION

As the application is designed to be flexible to cater for a range of client requirements, we define five different objectives that can be optimised. The rationale behind each of the objectives is explained below:

- *Idle Time - I* For a single employee, the idle time describes the ratio of time that the employee is neither working nor driving. I defines the average idle time over the complete schedule. Optimising this criteria is relevant in maximising employee efficiency.
- *Driving Time - D* For a single employee, this criteria measures the proportion of an employees working day that is spent driving as opposed to servicing, and thus is a measure of the productivity of an employee.
- *OverTime - O* The number of minutes worked beyond the end of the working day for each employee is calculated, and recorded as the ratio of the number of available working hours. Some clients do not allow overtime to be worked while others prefer to minimise this to reduce extra costs.
- *Unscheduled Job Time - O* Some jobs might not be able to be scheduled given the number of available employees and particularly if no overtime is allowed. This may also occur if the total time to complete a job (including driving and servicing) is greater than the working day of any employee.
- *Broken Minutes - W* For some jobs, scheduling within the tolerance period is critical. For other clients, some deviation may be tolerated, however may incur a cost.

The five objectives are combined using a weighted sum. Each of the weights can be specifed by the user according to their own preferences and can be altered interactively during the course of a run as show in in figure 1.

5. ALGORITHM

We utilise a hyper-heuristic approach that aims to reduce the amount of knowledge required in terms of the search methodology to make it easier for the client to maintain and/or develop in the future. Hyper-heuristics are well known to produce quick, acceptable solutions to problems (though rarely optimal) and our own previous work with a hyper-heuristic system called NELLI [4] suggested that combining heuristic components into a suitable sequence can lead to promising new heuristics that generalise well across sets of problems.

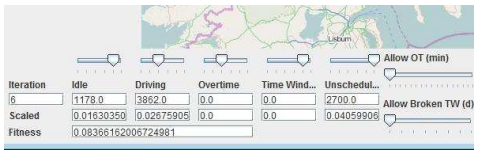

Figure 1: The sliders allow the relative weights to be adjusted during the course of a run depending on the criteria defined by the customer. The Customer can also specify the amount of allowed overtime and the maximum period by which time windows can be broken

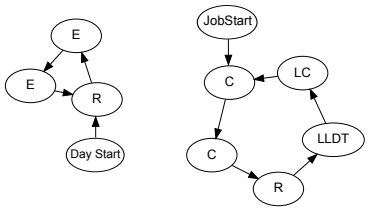

Figure 2: 2 lists of rules are maintained by each individual as a cyclic list with a predefined start point; one for the employee heuristics and the other for the job heuristics

We define a set of low-level heuristics, shown in table 1, using intuituve knowledge of how a human might solve the problem. Using a straightforward evolutionary algorithm, we evolve two cyclic lists with pre-defined start points that define a sequence of heuristics to be used in solving the problem, each of which has a pointer indicating the current heuristic to be used.

A wrapper, defined in figure 1, controls application of heuristics. The wrapper first chooses an employee according to the current heuristic in the employee list. A job is then selected according to the heuristic defined by the pointer in the job list. The pointer is then advanced, and jobs repeatedly selected in this manner until no more jobs can be placed into the selected employees day. The pointer in the employee list is then advanced and a new employee day selected. This process is repeated until no more jobs can be scheduled. In each list, the pointer cycles back to the start of the list once it reaches the end.

A standard steady-state evolutionary algorithm is used to evolve lists, using a randomly initialised population of chromosomes. A messy crossover operator produces two new children with probability p_c, from parents selected via tournament selection. A mutation operator selects at random one of nine possible mutation operators that add, delete or swap heuristics within a sequence, or concatenate two sequences together.

Table 1: Employee Rules and Job Rules

Job Heuristic	Abreviation	Description
ClosestJob	C	based on current location of an employee
EarliestJob	E	returns job with the earliest start date (no relaxation of the time window)
EarliestJobTW	ETW	returns job with the earliest start date (including any time window relaxation)
LargestJob	L	returns job with largest processing time
LargestJobClosest	LC	returns job closest to last location from set of jobs with largest processing time
LargestJobLDT	LLDT	returns job with largest (processing time + total driving time)
RandomJob	R	returns a random job
SmallestJob	S	returns job with smallest processing time
SmallestJobClosest	SC	returns job closest to last location from set of jobs with smallest processing time
SmallestJobSDT	SSDT	returns job with the smallest (processing time + total driving time)
SmallestTwJob	STW	returns job with the smallest total time window duration

Employee Heuristic	Abreviation	Description
EarliestDayEvent	E	returns earliest available day from all employees
RandomDayEvent	R	returns a random day from a random employee
RandomNextDayEvent	RN	returns the next free day from a randomly chosen employee

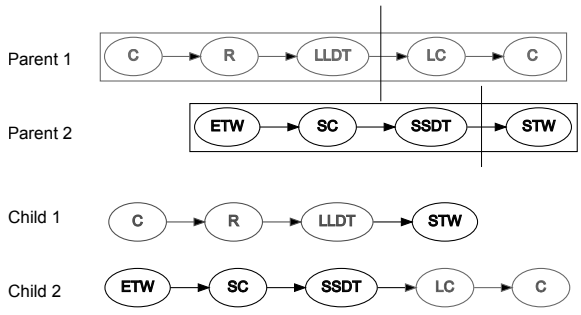

Figure 3: Crossover selects with equal probability one of the 2 heuristic lists and performs a "messy" one point crossover as shown (Shown not to benefit the EA)

Algorithm 1 Wrapper Pseudo Code

Require: EmployeeRuleList : The list of employee rules
Require: JobRuleList : The list of job rules
1: **repeat**
2: select next employee rule: returns an employees free time for a day
3: advance employee pointer (return to beginning at end)
4: **repeat**
5: select next job rule: returns a job that meets the constraints (overtime and time window)
6: advance job pointer (return to beginning at end)
7: schedule job
8: **until** no more jobs can be placed into the selected employees day
9: **until** employees days full OR no more jobs can be placed

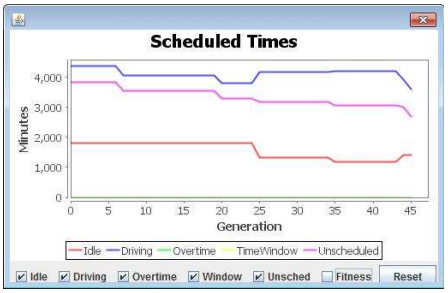

Figure 4: The plot shows the actual times for idle time, driving time, overtime, broken time windows and unscheduled jobs. The display can be switched to show the normalised weighted fitness used to drive evolution.

6. PRESENTATION OF RESULTS

In order to enable the client to interpret the results (and for them to be refined by a human scheduler if preferred) then the results need to be presented in a format that is easy to intepret.

Objectives.

Progress of the algorithm against each of the five objectives is shown in a display as in figure 4 which shows the change in each objective over time. As the sliders controlling the weighting of each objective can be controlled interactively during the course of the run, a scheduler is able to investigate alternative solutions and intuitively guide the algorithm towards their preferred objective. Note that some objectives are antaganistic, e.g. as driving time increases, idle time decreases and vice versa.

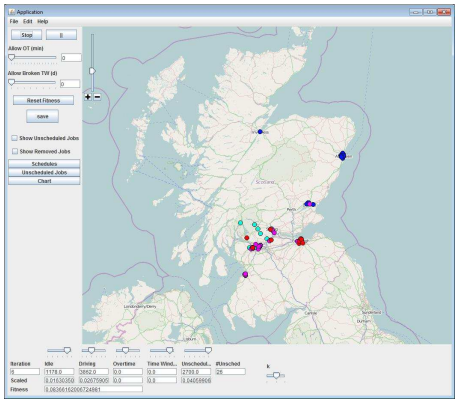

Figure 5: The Map displays the position of each employee jobs colour coded for each employee

Assignment.

Jobs assigned to each engineer are shown using a colour-coded map 5 (each colour represents an individual engineer). This gives an easy to interpret view in which it can be clearly be seen that jobs are clustered in particular geographic areas, located close to each engineers home base. Jobs tend to be clustered around the major population centres, however some jobs occur in regions that are located a considerable distance from any engineer, and require significant travelling.

Scheduling.

Schedules are displayed as the algorithm runs as indicated in Figure 6. Each vertical bar shows an employees calendar colour coded by employee. Coloured blocks indicate a jobs duration and the grey blocks indicate the corresponding driving time. In the example shown some employees have full or part days where they are not scheduled to work. This gives a visual overview of the relative amounts of driving time, servicing time, and idle time (indicated by gaps at the end of each day) for each employee. This view is for the purpose of the scheduler while running the algorithm. As previously mentioned, for each employee, calendar availability and assigned jobs are read and written using the *calDav* format from the employees own calendar, thus the scheduling information can also be viewed in calendar format as shown in figure 7 and can be manually manipulated using standard industry calendar tools if required.

7. CONCLUSION

We have described an approach that was taken to develop a WRSP application for a client. Although we are unable to present results across multiple problems due to client confidentiality, the paper illustrates a number of important points regarding the development of real-world applications. We have noted the emphasis on developing a practical solution, rather than seeking global optima. In contrast to taking a multi-objective approach, we have used a weighted objective function in order to present the client with a single solution. A map of the real road-network is used to calculate driving distances and times, and the application is able to access availability of employee calendars via a web interface. A simple search algorithm that combines heuristics is both effective and simple to understand and maintain from the client perspective. Finally, the algorithm can be extended in future by adding additional heuristics if required.

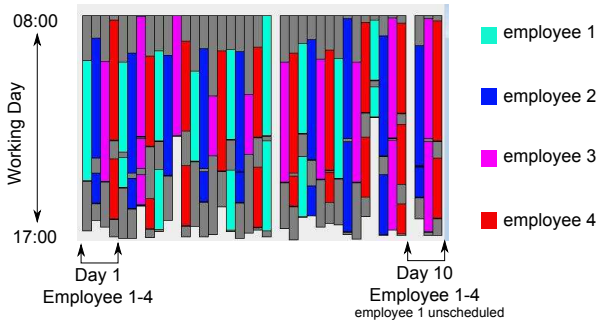

Figure 6: Each vertical bar shows an employees calendar colour coded by employee. Coloured blocks indicate a jobs duration and the grey blocks indicate the corresponding driving time. In the example shown some employees have full or part days where they are not scheduled to work. This shows the same information as is shown in the calendar application depicted in Figure 7 but is updated continuously during the course of a run.

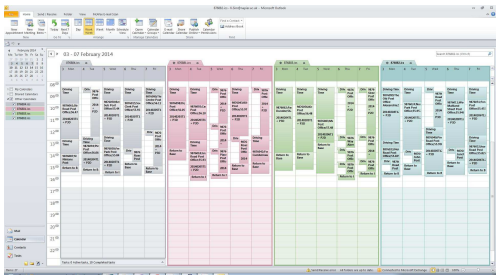

Figure 7: Output from the application is in the format of an individual ics file for each employee which can be imported into a number of Calendar applications allowing the customer to easily visualise and alter schedules manually if desired

8. REFERENCES

[1] A Castillo-Salazar, D Landa-Silva, and R QuA. Survey on Workforce Scheduling and Routing Problems. The 9th International Conference on the Practice and Theory of Automated Timetabling (PATAT 2012), 28-31 August 2012, Son, Norway

[2] GraphHopper `http:www.graphhopper.com` applications. Ubiquity 2012 3:1–3:13.

[3] Open Street Map `http:www.openstreemap.org`

[4] Sim, K., Hart, E.: An improved immune inspired hyper-heuristic for combinatorial optimisation problems. In: GECCO '14: Proceeding of the sixteenth annual conference on Genetic and evolutionary computation conference (In Press 2014)