A choice function hyper-heuristic framework for the allocation of maintenance tasks in Danish railways

Shahrzad M.Pour[a], John H. Drake[b], Edmund K. Burke[b]

[a]*DTU Management Engineering, Technical University of Denmark, Produktionstorvet, 2800 Kgs. Lyngby, Denmark*
[b]*Operational Research Group, Queen Mary University of London, Mile End Road, London E1 4NS, UK*

Abstract

A new signalling system in Denmark aims at ensuring fast and reliable train operations, however imposes very strict time limits on recovery plans in the event of failure. As a result, it is necessary to develop a new approach to the entire maintenance scheduling process. In the largest region of Denmark, the Jutland peninsula, there is a decentralised structure for maintenance planning, whereby the crew start their duties from their home locations rather than starting from a single depot. In this paper, we allocate a set of maintenance tasks in Jutland to a set of maintenance crew members, defining the sub-region that each crew member is responsible for. Two key considerations must be made when allocating tasks to crew members. Firstly a fair balance of workload must exist between crew members and secondly, the distance between two tasks in the same sub-region must be minimised, in order to facilitate quick response in the case of unexpected failure. We propose a perturbative selection hyper-heuristic framework to improve initial solutions by reassigning outliers, those tasks that are far away, to another crew member at each iteration, using one of five low-level heuristics. Results of two hyper-heuristics, using a number of different initial solution construction methods are presented over a set of 12 benchmark problem instances.

Keywords: Hyper-heuristics, Maintenance Scheduling, Combinatorial Optimisation, European Rail Traffic Management System

1. Introduction

European Railway Traffic Management System (ERTMS) [2] is the newest signalling standard to systematise train control and communication system within railway networks. The motivation behind ERTMS has been to enhance the signalling communication amongst various train systems, to improve connectivity and allow for faster travel between European countries. Although ERTMS was initially presented by the European Union for the scope of European countries, it rapidly was discerned as a worldwide signalling standard. As ERTMS is still in the primary stages of operation, there is very limited research pertinent to the maintenance processes and other aspects in ERTMS [33, 28, 24, 10, 2].

Denmark will be the first country in Europe to upgrade its entire signalling system to ERTMS. Railway track and signalling systems are complex and highly interdependent. Unlike when a failure happens on a track segment, failure of one component in the signalling system may lead to the failure of other components or even propagate to the whole network. This differentiation makes the partitioning of each sub-system particularly influential, affecting the levels of operability and maintainability of the entire railway network [19].

Given the huge investment required to implement ERTMS - Denmark has invested approximately 3 billion Euros in the system [1] - effective maintenance is critical, and as the new system uses completely different hardware to the previous system. In addition, the maintenance tasks required differ significantly, with very strict time limits and constraints on recovery operations.

As defined by the industrial partner of the ERTMS project in Denmark, a maintenance plan should define the sub-regions in which different maintenance crew members work. In addition to the workload being fairly balanced across sub-regions, the geography of these regions should ensure that crew members can travel between two points quickly, when needed, in order to handle unexpected failures and breakdowns. Once the sub-regions are defined, the planner can estimate the maximum distance each crew member must travel within their own region, in case of failure in the future. Following this notion, the best route for each crew member can be determined and the overall driving distance cost calculated for the entire maintenance plan. We must emphasise here that this routing phase is considered as a separate optimisation problem and will not be studied in this paper.

The focus of this paper is the allocation of maintenance tasks to crew members for the Jutland peninsula, the largest region in Denmark. The current maintenance planning system in the country is decentralised, with crew members starting their duties from different locations rather than from a single depot. This structure requires an effective assignment of tasks to avoid high total driving distance costs or, in some cases, to ensure a feasible plan is made. Based on the allocations found, each crew member is responsible for undertaking tasks within their own sub-region.

Considering the characteristics of the maintenance planning problem introduced above, the problem can be seen as Multi-Depot Vehicle Routing Problem (MDVRP) [18], where each vehicle operates on its own routes, starting and finishing at a specific depot. According to the industrial partner of the project, each crew member is equipped with a technical vehicle and all the necessary equipment to undertake any task. Each crew member in our problem can be seen as a vehicle within the MDVRP, with their home location corresponding to a depot. Starting and ending their route at the depot location, each crew member must complete all of the tasks that they have been assigned. As the MDVRP is an NP-hard problem, heuristic methods have been used widely within the literature. Among the existing heuristic approaches, Tabu Search [6] and adaptive large neighbourhood search [26] have been shown to be particularly successful. Montoya-Torres et al. [22] provide a comprehensive survey on approaches to solving the MDVRP.

Due to the structure of the MDVRP, the process of determining which customers are served by which depots has been fundamental to many proposed solution approaches. Such approaches fall under the research spectrum of cluster-first, route-second approaches [12, 25], in which the clustering phase is usually solved by an assignment algorithm [32]. Giosa et al. [15] proposed a number of assignment algorithms for the MDVRP, three of which, namely Parallel Assignment, Simplified Assignment and Sweep Assignment [29], were referred to as methods which perform *assignment through urgencies*. These methods define a precedence relationship between customers, to determine the order in which they are serviced by the depot, with high-priority or "urgent" customers served first.

Hyper-heuristics represent a class of high-level search techniques employed for solving combinatorial optimisation problems [5]. Unlike traditional search methods, which operate on a space of solutions, hyper-heuristics operate on a search space of low-level heuristics or heuristic components. A recent definition of hyper-heuristics is given by Burke et al. [5]:

> 'A hyper-heuristic is a search method or learning mechanism for selecting or generating heuristics to solve computational search problems'.

This definition covers the two main categories of hyper-heuristics: *selection* hyper-heuristics, which choose a heuristic to apply at each step of a search, and *generation* hyper-heuristics, which generate new heuristics from existing sets of low-level heuristics or components. A traditional selection hyper-heuristic iteratively selects and applies low-level heuristics to a single solution, using a move acceptance criterion to make a decision regarding whether to keep the new solution for each step. While there has been sustained research interest in hyper-heuristics in the last decade or so in particular, methods exhibiting hyper-heuristic behaviour can be traced back to as early as 1961 [11]. Selection hyper-heuristics have been previously applied successfully to a wide array of problem domains, including bin packing [20], dynamic environments [17], examination timetabling [23], the multidimensional knapsack problem [9], nurse rostering [4], sports scheduling [14] and the vehicle routing problem [13]. Here we will use a selection hyper-heuristic to define working sub-regions for maintenance crew members across the Danish rail network.

This paper is organised into five sections. In Section 2, we present the problem definition, including a mathematical model of the railway maintenance crew scheduling problem and a description of the instances used. Section 3 describes the proposed framework used to solve the problem, and Section 4 presents experimental results and a discussion on the proposed framework. Finally, this paper closes with a conclusion in Section 5.

2. Problem definition

2.1. Mathematical model

The mathematical model of the problem that we deal with in this paper is as follows. Given a set of crew members C and a set of maintenance tasks M, with crew indices $k, v \in C$ and maintenance task indices $l, h \in M$, decision variable $x_{k,l}$ is set to 1 if task l is assigned to crew member k; otherwise, it is 0. $Q_{k,l}$ denotes the distance between crew k and task l, while $S_{l,h}$ is the distance between task l and task h and d_l is the duration of task l. The objective function (1) is multi-criteria, whereby the first term in the objective function minimises the total travel time from a crew member's location to the assigned

tasks for each crew member. The second term ψ, together with constraint (2), aims at minimising the maximum distance among task pairs within each sub-region. This reflects the definition of the diameter of a sub-region as the maximum distance between any two tasks assigned to a maintenance crew member.

In addition, fair distribution of the tasks among the crew is considered as a third criterion (w). Workload distribution is modelled according to the balancing constraints defined by Bredstrom and Ronnqvist [3]. Using this formulation, constraint (3) balances mismatches across different sub-regions, where w represents the biggest difference in the total duration of assigned tasks between any two sub-regions. Constraint (4) ensures that each task is assigned only to one crew member.

$$Minimise \quad \sum_{k \in C} \sum_{l \in M} x_{k,l} * Q_{k,l} \; + \; \psi \; + \; w \tag{1}$$

subject to:

$$x_{k,l} * x_{k,h} * S_{l,h} \leq \psi \quad \forall \, k \in C \quad \forall \, l, h \in M \tag{2}$$

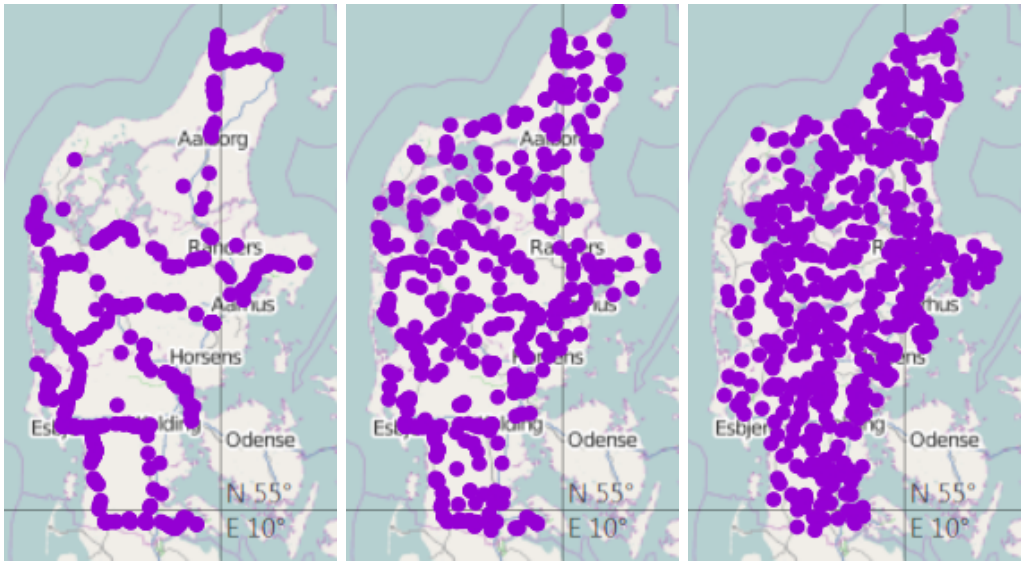
$$\sum_{l \in M} x_{k,l} * d_l \; - \sum_{l \in M} x_{v,l} * d_l \; \leq \; w \quad \forall \, k \in C. \, \forall \, v \in C \setminus \{k\} \tag{3}$$

$$\sum_{l \in M} x_{k,l} = 1 \quad \forall \, k \in C \tag{4}$$

2.2. Dataset

As ETRMS has not yet been implemented, this is exploratory work commissioned by Banedanmark, the state-owned Danish company in charge of maintenance and traffic control of most of the Danish railway network. As such, there is currently no solution implemented in practice yet. This work has been done prior to the implementation of ERTMS, to give some indication of the problem that they are likely to face, and ensure that they are prepared when it comes to solving the problem in the future. In this section we define the instances used for experimentation. The geographical points are all located in the Danish peninsula of Jutland. Tasks should be assigned to a number of crew members. Coordinates representing the geographical location of the tasks were generated by utilising the Google Map API. This was done based on three different task location generation strategies:

1. Exact (E). Tasks are all located on the rail tracks of the Jutland region.
2. Mixed (M). Tasks are located at a mix of on- or off-track positions within the Jutland region.
3. Random (R). Tasks are scattered randomly across the Jutland region.



(a) E500 (b) M500 (c) R500

Figure 1: Geographical Visualisation of the three types of instance

For each of these three cases, four instances were generated with a different total number of tasks: 100, 500, 1000 and 5000, resulting in 12 problem instances overall. These should be serviced by a team of eight crew members. These numbers were chosen respectively according to the numbers of maintenance tasks which need to be done on a daily, weekly, monthly and annual basis. To standardise our test cases, we follow the file format of the classical benchmark test sets for the Vehicle Routing Problem with Time Windows (VRPTW), introduced by Solomon[1]. The dataset and documentation about how the instances were created are accessible at `http://github.com/ShahrzadMP/Dataset`. Each instance is referred to by its *locationType-taskTotal* pair herein, e.g. E100, R5000 etc. Figure 1

[1]http://w.cba.neu.edu/~msolomon/problems.htm

presents a geographical visualisation of the on-track, on- and off-track and random instances with 500 tasks.

3. Proposed framework

Given an existing solution generated by an initial constructive phase, we use a selection hyper-heuristic to improve the assignment of maintenance tasks to crew members. As with many existing selection hyper-heuristics, the search is performed on a single candidate solution, in an attempt to improve a given solution at each iteration, using two phases: heuristic selection and move acceptance [23]. By applying a selected heuristic at each iteration, a candidate solution (Sol_t) at a given time (t) is modified into a new solution. A move acceptance criterion makes the decision whether to accept or reject the new solution.

In the proposed framework, task assignments are modified by reassigning tasks that are far away from a maintenance crew member's starting position to another maintenance crew member's sub-region. Such tasks are representative of the concept of outliers, explained in more detail in Section 3.2. The algorithm starts with a constructive phase to generate an initial feasible solution. Next, at each iteration, the algorithm tries to detect an outlier in a particular sub-region. If no outlier is found for any of the sub-regions of the current solution, the algorithm terminates and the best solution is returned as the final solution. If an outlier is detected, the hyper-heuristic selects and applies a low-level heuristic to reassign the outlying task, before the move acceptance criteria decides whether to accept this new allocation. This process continues until either no outliers remain or one of the given termination criterion is met. The overall framework is illustrated in Figure 2.

3.1. Initial solutions

To generate initial solutions, we present a constructive deterministic heuristic based on two different ordering strategies, in order to assign tasks to maintenance crew members. The set of tasks allocated to each crew member represents the sub-region in which the crew member operates. The constructive heuristic starts with a list of maintenance tasks, sorted according to the distance of each task from the crew member's starting location, and in each step a task is allocated to a crew member, depending on the ordering strategy being used. We define two strategies to decide the order in which tasks are allocated: Furthest Task First (FTF) and Closest Task First (CTF). In

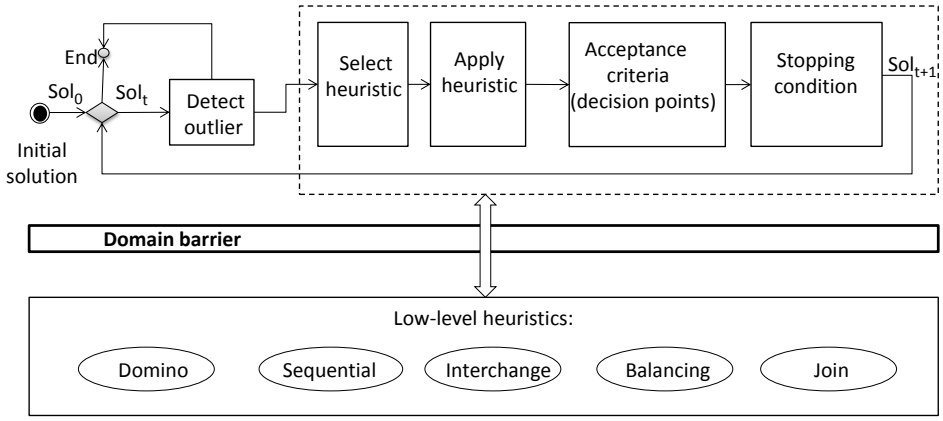


Figure 2: Proposed perturbative selection hyper-heuristic framework

FTF, tasks are ordered in descending order of distance from the closest crew member, with the task furthest from its closest crew member allocated first. This strategy intends to allocate "difficult to assign" tasks which are a long distance from any crew member early on in the construction process. Conversely, CTF allocates tasks in a greedy manner, assigning them in ascending order of distance away from the closest crew member.

In order to ensure that tasks are distributed fairly among all crew members, a Tabu list is used to manage those who are able to be allocated a task at a given point. Once a task is allocated to a crew member, the heuristic is prohibited from allocating this person another task until the Tabu list becomes empty. In this way, the number of tasks assigned to each crew member is balanced while constructing the solution. Algorithm 1 presents the pseudocode for the constructive heuristic. For comparison, we have also implemented the Simplified Assignment (SA) algorithm [15] from the literature, which orders tasks by the difference in distance from a task to the closest and second closest crew member.

3.2. Identifying outliers

In the task allocation problem described above, in order to ensure a quick response across the network in the event of failure, the maximum distance between the tasks should be minimised within each sub-region (cluster). This reflects the definition of the diameter of a cluster, that is, the maximum distance between any two points of the sub-region [27]. Explicitly calculating

Algorithm 1: Ordering heuristic, employed to generate initial solutions

```
1: Order task list M according to ordering strategy (FTF or CTF)
2: Initialise tabuList as empty
3: Set tabuList size to number of crew member - 1
4: for each task l in M do
5:    if Size of tabuList equals to maximum size of tabuList then
6:       empty the tabuList
7:    Allocate l to closest non-Tabu crew member c
8:    Add c to tabuList
9: end for
```

the diameter of a sub-region can be costly, and requires checking all pairs of tasks within that sub-region. In terms of time complexity this is $O(n^2)$, where n is the number of tasks within the sub-region. To reduce the time complexity of our approach and allow for better scalability, we use the radius of the sub-region instead of the diameter. The radius of a sub-region is defined as the maximum distance between all the points and the sub-region centre and can be calculated in $O(n)$ time. Whilst the radius and diameter of a cluster are not associated directly, they do have a propensity for being proportional [27].

Figure 3 shows the outlier detection module in the proposed framework. A sub-region is selected randomly from the current solution at hand. In order to detect an outlier, the module finds the task furthest away from the sub-region centre, defined as the starting location of a crew member. If the radius is greater than half of the maximum allowed distance during the failures, it is recognised as an outlier. In the Banedanmark problem, the maximum allowed distance is 100 km which corresponds to roughly an hour and a half travel time. For example, if the furthest task away from the sub-region centre (radius) is 80 km, the task will be detected as the outlier, as the radius is greater than half of the maximum allowed distance, which is 50 km in this example.

If an outlier is detected within the current sub-region, the algorithm will enter the improvement phase, carried out by the selection hyper-heuristic. If not the algorithm will add the selected sub-region to a Tabu list, to avoid re-selecting sub-regions that do not contain any outliers. After a sub-region is added to the Tabu list, the algorithm continues to keep selecting a non-Tabu sub-region until it finds either a sub-region with an outlier, or there

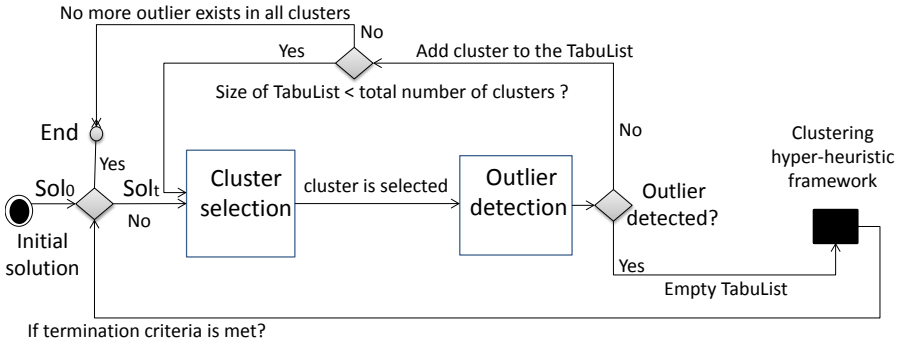


Figure 3: Outlier handling module

are no more non-Tabu sub-regions from which to choose. Each time an outlier is detected successfully, the Tabu list is emptied. Outlier detection is possible until the radius (furthest task away from the centre of the sub-region) of all sub-regions is no further than half of the maximum distance a crew member is allowed to travel in the case of a breakdown. In the worst case the maximum distance from a crew members current location to the location of a failure within the sub-region should be twice the radius of the sub-region, and therefore within the maximum distance allowed.

3.3. Choice function heuristic selection

Once an outlying task has been identified, a low-level heuristic is applied to reassign the task to another sub-region. The impact of different low-level heuristics on a certain solution is dependent on two factors: the nature of the low-level heuristic and the point in the search at which they are applied. Hence, if the state of the search can be acknowledged through some mechanism, a hyper-heuristic can apply an appropriate heuristic at each step, in order to guide the solution towards better areas of the solution space. The choice function is an intelligent heuristic selection strategy, introduced by Cowling et al. [7] to evaluate and rank the performance of multiple low-level heuristics. Choice function-based hyper-heuristics and variants have since been used to solve a variety of different problems [16, 9, 21].

The choice function comprises three terms and utilises information about the impact of each low-level heuristic individually (f_1), the combined impact of applying two heuristics successively (f_2) and the amount of time elapsed since the heuristic was last called (f_3) [7]. At each decision point, the low-level heuristic with the highest score, calculated using the choice function, is

selected and applied to the current solution. Exploitation of the search space is taken into account by gathering performance information on the heuristics through f_1 and f_2. Exploration of other parts of the search space is achieved by selecting low-level heuristics that have not been applied recently (f_3). The parameters α, β and γ are used to weight each of the three components (f_1, f_2 and f_3), giving greater weight to recent performance. The complete formulation of these components is as follows:

$$f_1(h_j) = \sum_n \alpha^{n-1} \frac{I_n(h_j)}{T_n(h_j)} \tag{5}$$

$$f_2(h_k, h_j) = \sum_n \beta^{n-1} \frac{I_n(h_k, h_j)}{T_n(h_k, h_j)} \tag{6}$$

$$f_3(h_j) = \tau(h_j) \tag{7}$$

where $I_n(h_j)$ and $T_n(h_j)$ are changes in the objective function and CPU time taken the n^{th} last time the heuristic h_j was called. $I_n(h_k, h_j)$ and $T_n(h_k, h_j)$ indicate the change in the evaluation function and the amount of CPU time taken, the n^{th} last time the heuristic h_j was called directly after heuristic h_k. Finally, $\tau(h_j)$ is the time elapsed since the heuristic h_j was last called. The choice function, F, for a given heuristic is calculated as:

$$F(h_k, h_j) = \alpha f_1(h_j) + \beta f_2(h_k, h_j) + \gamma f_3(h_j) \tag{8}$$

To enhance the generality and robustness of our hyper-heuristic, a self-adaptive version is preferable. Accordingly, we use the parameter-free choice function introduced by Cowling et al. [8] which tunes the parameters of the choice function at each decision point based on the state of the search space, rather than using constant values for α, β and γ during the search. The parameters α, β and γ are rewarded or punished if the resulting solution following the application of a low-level heuristic is better or worse than the previous solution, respectively. This adaptivity allows for regular interplay between the parameters of the choice function, modifying the weighting assigned to each parameter according to the performance of each low-level heuristic application. Various approaches can be implemented as

a reward/punishment strategy to control α, β and γ. Examples include a linear scheme (e.g. $\alpha = \alpha(1 + \epsilon)$) or non-linear (e.g. $\alpha = \alpha^{(1+\epsilon)}$) scheme, where ϵ can be either a negative or positive constant, or a function of the relative improvement obtained from the change in the evaluation function after employment of the last selected heuristic [30]. Here we employ the adaptive choice function hyper-heuristic taken from the schematic view given by Soubeiga [30], using a linear scheme with a constant value of 0.1 with the positive or negative sign for the reward and punishment scheme, respectively. Initially, α, β, and γ are set to 1.

This adaptive variant of the choice function will be referred to as CFHH in the remaining sections of the paper. In addition, our experiments will also use a simple random hyper-heuristic (SRHH) for comparison, which makes a uniform random selection of low-level heuristic to apply at each step.

3.4. Low-level heuristics

We introduce five low-level heuristics the hyper-heuristics to select from. A low-level heuristic defines a strategy to reallocate a task identified as an outlier in one sub-region to another maintenance crew member. The five low-level heuristics are illustrated in Figure 4, in which a circle represents a single maintenance crew member's sub-region, with each point denoting a task allocated within that particular sub-region. Red points are tasks identified as outliers, while black points could be either an outlier or a non-outlying task. All of the proposed low-level heuristics, except for Balancing, have been defined as hill-climbing methods. This means that when they are applied to a solution, if the solution is not improved, the new solution is discarded and the original solution retained. The balancing low-level heuristic does not consider the change in objective function value, and only attempts to balance the number of tasks allocated to each crew member in the current solution.

Domino: the Domino heuristic first moves the identified outlying task to the sub-region of the closest other maintenance crew member. Subsequently, the sub-region which has received the outlier does the same and reassigns its furthest task to the sub-region of the closest crew member's starting location, thereby having a "domino effect" on the overall solution.

Pair: this heuristic removes two outliers sequentially from the selected sub-region and assigns them to the best possible sub-region in terms of the distance of the outlier to the other sub-regions' centres. The destination

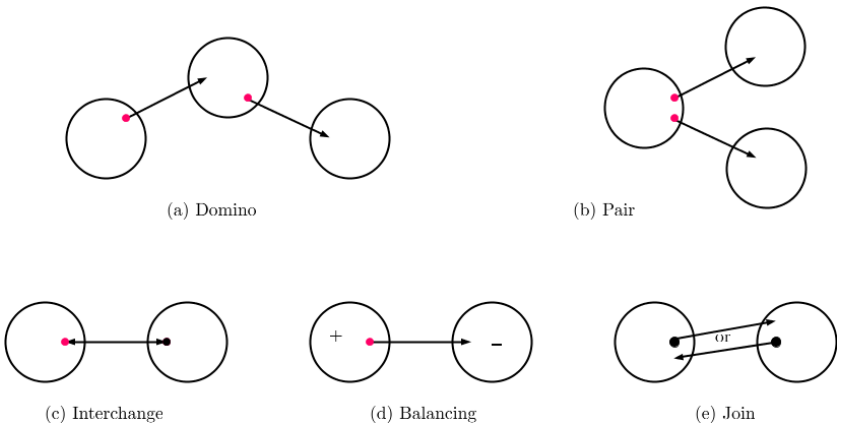


Figure 4: Proposed low-level heuristics

sub-region for the two outliers could be the same or different. This heuristic changes the balance of the sub-regions.

Interchange: this heuristic tries to allocate an outlying task to the closest other crew member in exchange for another task, which is closer to the first crew member than the original outlier. The task received from the second crew member could either be an outlier or another task which is closer to the first crew member's starting position.

Balancing: in order to try to balance the number of tasks between crew members, the Balancing heuristic moves an outlying task to another crew member, who is currently allocated fewer tasks in total.

Join: this low-level heuristic looks for two tasks which are close to each other in terms of distance, but belong to different sub-regions. It then tries to place the two tasks in the same sub-region. Out of the two possible moves, the assignment which yields the lowest average distance of the two tasks away from the centre of the sub-regions is kept.

3.5. Pseudocode for the proposed framework

The framework that we present in this paper is composed of three phases: generating an initial solution, detecting the outlier and improving the solution using a selection hyper-heuristic. In each run of the algorithm, one initial solution is generated and then the solution is improved through collaboration between the outlier detection and improvement hyper-heuristic phases.

Algorithm 2 presents the pseudocode for the proposed choice function hyper-heuristic approach to the problem (CFHH). The search space of the

high-level heuristic consists of all possible permutations of the low-level heuristics defined in Section 3.4. The algorithm starts by generating an initial solution using one of the constructive heuristics introduced in Section 3.1. Once a solution is constructed, the algorithm enters the main loop to find an outlier of one of the sub-regions and improve the solution iteratively, until the stopping condition is met. Outlier detection (line 5) has been explained in detail in Section 3.2. If an outlier is found, the algorithm will attempt to improve the solution using the choice function hyper-heuristic introduced in 3.3 operating over the low-level heuristics described in Section 3.4.

As discussed earlier, in order to enhance the robustness of the presented framework in this paper, we employ the adaptive choice function [30], which automatically changes its parameters according to the search space in which it is operating. The rest of the algorithm from line 7 refers to the schematic flow chart of the adaptive choice function introduced by Soubeiga [30]. At the beginning of the search, the variable $nonImprovement$ is declared, to keep track of the number of consecutive iterations no changes to the objective function are made. The choice function value is then computed for each heuristic, and the heuristic h_j with the highest F value is selected (lines 7 and 8). $H2$ is another heuristic, with the highest value for f_3, used to provide an appropriate level of exploration of the heuristic search space (line 9). In order to determine whether the hyper-heuristic needs to exploit or explore the solution space at each iteration, G, the biggest contributor to the F value of the selected heuristic, is identified. This prescribes the way in which the chosen heuristic is applied (line 13). In the case of N consecutive non-improving iterations, $H2$ is applied to the solution (line 12).

In general, when the algorithm is in an exploitation phase ($G = f_1$ or $G = f_2$), the chosen heuristic is applied in steepest descent fashion (line 14). If the solution requires exploration ($G = f_3$), the heuristic with the smallest f_3 value is applied in steepest descent fashion (line 18). If this yields an improvement γ is punished (line 20), otherwise h_j is applied using steepest descent (line 22). If this still doesn't lead to an improvement, the solution is returned to the previous solution and h_j applied once (line 24). If no component of the choice function dominates the others in terms of contribution to F, h_j is applied in steepest descent fashion (line 26). Following the application of a low-level heuristic to the solution, $nonImprovement$ is incremented if no improvement has been found and set to 0 in the case of improvement (line 27). After more than N consecutive non-improving iterations, the algorithm rewards γ and $H2$ is applied to the solution (line 29 to 33).

Algorithm 2: Pseudocode of the choice function selection hyper-heuristic framework (CFHH)

```
1: Generate initial Solution
2: Initialise heuristic list h = h1, h2, h3, h4, h5
3: N = Num of low-level heuristics, iteration = 0, nonImprovement = 0
4: while termination criteria not met do
5:    Outlier detection
6:    if any outlier is found then
7:       Compute choice function F for each heuristic
8:       Select heuristic hj for which F is max
9:       Select heuristic H2 where f3 is max, and H2 ≠ hj
10:      if nonImprovement is ≤ N then
11:         if nonImprovement = N then
12:            Apply heuristic H2 to Solution
13:         G = biggest contributor to F, either f1, f2 or f3
14:         if G = f1 or f2 then
15:            Apply hj in steepest decent
16:            Reward or punish α or β, based on solution improvement/deterioration
17:         else if G = f3 then
18:            Select hi for which F − f3 is max and apply in steepest descent
19:            if there is any relative improvement and hi ≠ hj then
20:               Punish γ
21:            else
22:               Apply hj in steepest decent
23:               if there is no relative improvement then
24:                  Undo steepest descent and apply hj once
25:         else
26:            Apply hj in steepest decent
27:         Calculate absolute improvement and update nonImprovement
28:      else
29:         Reward γ
30:         Apply H2 in steepest decent
31:         nonImprovement = 0
32:         if there is no relative improvement then
33:            Undo steepest decent and apply H2 once
34:   iteration = iteration + 1
35: end while
36: return Solution
```

The algorithm terminates under three different criteria. The first occurs when no outlier is found in any of the sub-regions within the solution. If no outliers are detected, the low-level heuristics have no task to reassign to another sub-region. The second criterion is met when an outlier is detected, but the hyper-heuristic cannot improve the solution after a certain number of iterations. This threshold is set to 0.1 * the number of tasks in the problem instance. Finally, if the algorithm does not fail under the previous conditions, the framework will stop after a set number of iterations (2 * number of tasks in the instance).

4. Results and discussion

This section presents a number of experiments to analyse various aspects of the proposed framework. Firstly, the results of the initial solutions obtained using the CTF, FTF and SA assignment algorithms introduced in Section 3.1 are compared. Following this, the results of the proposed choice function selection hyper-heuristic (CFHH) applied to the three different initial solutions generated for each instance are presented. Next, we compare CFHH to a baseline simple random hyper-heuristic (SRHH) using the solutions generated by FTF. Detailed analysis of the performance of low-level heuristics is then performed, using the three largest instances. Finally, detailed performance of the choice function hyper-heuristic (CFHH) during a single run is presented, using one of the largest instances as an example. All experiments were run using an Intel Core (TM) i7-4600U CPU 2.10 GHz processor, with 8.00 GB RAM.

4.1. Quality of the initial solutions generated using different constructive heuristics

Table 1 summarises the results of using three different constructive heuristics to generate solutions for the 12 instances introduced in Section 2.2. This table shows five different measurements related to each solution. Total_D is the total distance cost, calculated as the sum of the distances between each task and the crew member to which it is assigned. MDD gives the maximum distance between two tasks allocated to a single crew member within the whole solution. This gives an indication of the worst case scenario in terms of travel time in the case of unexpected failures or breakdowns. Similarly, AVG_MDD calculates the average maximum distance travelled by each crew member, to give an "average worst case" across the entire solution. w is

the imbalance in workload distribution across different sub-regions on the railway network. The CPU time taken to generate the solution in seconds is also given (CPU_T). The best value for each metric between the three constructive heuristics is highlighted in bold.

From Table 1, we can see that SA generates many of the best results in terms of Total_D and MDD. In other measurements, FTF generates marginally better results in the majority of cases for AVG_MDD and CPU_T(s), and CTF generates slightly better results in terms of Total_D for the 'R' instances. The only exceptional cases are as follows: FTF generates results much more quickly (256.48, 201.76, 360.94) for large instances compared to SA (575.89, 416.75, 412.36) on E5000, M5000 and R5000, respectively. CTF also generates significantly better results in terms of Total_D (7283.62) for instance R100 compared to SA (7413.68). Regarding workload imbalance (w), SA results in a better distribution of tasks overall, however there is not a big difference compared to CTF and FTF.

It is evident that the results achieved by FTF are close to the results of SA, while CTF generates the worst results. Using FTF ordering, tasks are assigned to the crew members, starting with the most difficult tasks through to the easiest. Using FTF the algorithm penalises the solution in the early steps of solution construction, however this protects the solution from receiving high penalties for assigning the remaining faraway tasks to the crew in the final steps of solution construction. Distant tasks which are difficult to place are assigned to a better possible choice in the early stages of constructing a solution, unlike CTF which effectively assigns tasks in a greedy manner. Similarly, the difference measure used by SA prevents bigger penalties later on in the construction of a solution by assigning tasks which are close to a single crew member early on. In the remaining sections of the paper, we will use the solutions obtained by the CTF, FTF and SA construction heuristics as input for hyper-heuristics attempting to improve the initial task allocations.

4.2. Results of CFHH using different initial solutions

Here we will analyse the impact of different initial solutions with different qualities on the performance of CFHH. For this purpose, we performed 10 CFHH runs, starting from the same initial solution for the solutions generated by CTF, FTF and SA for each instance. Table 2 shows the average performance obtained by CFHH, using different initial solutions based on the five measurements introduced in Section 4.1. Each of these measurements

Table 1: Results of initial solutions obtained by CTF, FTF and SA on all instances

Closest Task First (CTF)					
Instance	Total_D	MDD	AVG_MDD	w	CPU_T(s)
E100	5935.83	297.65	170.01	4	0.29
E500	28334.50	303.68	236.99	**3**	1.92
E1000	57073.51	323.33	241.69	**0**	7.02
E5000	287313.42	328.52	253.34	**0**	282.13
M100	5419.80	300.79	134.78	4	0.11
M500	31825.80	327.17	233.27	**3**	1.89
M1000	58566.95	322.90	237.94	**0**	9.31
M5000	292217.82	331.62	247.80	**0**	528.53
R100	**7283.62**	301.75	170.06	4	0.09
R500	33667.66	318.43	224.48	4	1.21
R1000	**64439.48**	317.52	231.03	**0**	5.64
R5000	**333296.85**	330.25	248.54	**0**	326.87
Farthest Task First (FTF)					
Instance	Total_D	MDD	AVG_MDD	w	CPU_T(s)
E100	5546.58	255.10	**143.05**	**3**	0.23
E500	25568.83	189.41	138.48	4	1.44
E1000	51971.83	**189.95**	142.48	**0**	**4.77**
E5000	260716.26	265.46	**208.25**	**0**	**256.48**
M100	5401.86	**248.63**	**131.49**	4	**0.11**
M500	31378.27	**254.40**	**192.15**	4	1.88
M1000	55425.78	258.17	**197.51**	**0**	**6.31**
M5000	280743.54	259.59	198.23	**0**	**201.76**
R100	7526.52	255.07	**164.50**	**3**	0.12
R500	33290.34	**259.71**	**184.85**	**3**	**1.57**
R1000	64619.51	**264.88**	197.39	**0**	**5.50**
R5000	333592.20	266.09	**195.79**	**0**	**360.94**
Simplified Assignment (SA)					
Instance	Total_D	MDD	AVG_MDD	w	CPU_T(s)
E100	**5233.94**	**255.07**	143.78	**3**	**0.22**
E500	**25460.18**	**189.40**	**138.19**	**3**	**0.75**
E1000	**51901.78**	190.00	**142.47**	**0**	5.90
E5000	**260694.49**	**265.03**	208.30	**0**	575.89
M100	**5154.75**	**248.63**	143.31	**3**	**0.11**
M500	**31302.68**	**254.40**	193.42	4	**1.16**
M1000	**55317.69**	**258.02**	197.56	**0**	6.42
M5000	**280666.80**	**257.41**	**197.96**	**0**	416.75
R100	7413.68	**245.83**	166.05	**3**	**0.09**
R500	**33214.83**	**259.71**	184.87	**3**	1.58
R1000	64545.90	**264.88**	**197.26**	**0**	6.90
R5000	333471.23	**265.10**	195.81	**0**	412.36

is followed by a column indicating the relative ranking of that measurement compared to the other two methods for generating initial solutions.

At a glance, the results indicate that CFHH using solutions constructed on an FTF basis, performs better in the majority of measurements for all instances, ranked mainly first and second, with SA also performing well. This is despite the fact that the quality of the initial solutions generated by FTF were often of poorer quality than those generated by SA in the previous subsection, especially in terms of Total_D. Notably, CTF generates the worst results in all instances under the Mixed (M) and Random (R) categories in terms of Total_D, MDD and AVG_MDD. This demonstrates that starting with a solution which makes decisions on a greedy basis makes any improvement to the solution more difficult when applying CFHH. In other words, a good balance between the greediness of the initial solution and the adaptiveness of the hyper-heuristic is not found. It is notable that the results obtained using these distance-based measurements seem to be correlated, with the best solutions in terms of Total_D often also performing best in MDD and AVG_MDD.

4.3. Comparison between CFHH and simple random hyper-heuristic (SRHH)

Here we will make a direct comparison between a simple random hyper-heuristic (SRHH), which makes a uniform random choice of low-level heuristic at each step, and the adaptive choice-function-based hyper-heuristic (CFHH). Both SRHH and CFHH start with a solution produced with FTF following the results presented in the previous subsection. Results (best and average over 10 runs) are given in Table 3 for all 12 instances. This table shows the three distance-based measures as before (Total_D, MDD and AVG_MDD). Each of these measurements is followed by a column showing the percentage of the improvement to the corresponding measurement compared to the initial solution constructed by FTF, shown earlier in Table 1. In the case that this percentage value is negative, the solution quality by this metric is worse than the initial solution. The last row of each set of results represents the average percentage of the improvement achieved by SRHH and CFHH for each measurement over all instances.

From Table 3 we can see that both SRHH and CFHH improved the initial starting solution in terms of Total_D for all instances. CFHH improves in all three measures on average over the 12 instances. This is likely to be due to the rationale behind the proposed low-level heuristics, Domino, Pair and Join, which minimise the maximum distance between two tasks

Table 2: Average performance over 10 runs of the choice function hyper-heuristic (CFHH) on all instances, starting from initial solutions obtained by FTF, CTF and SA

CFHH starting from solutions generated by CTF										
Instance	Total_D		MDD		AVG_MDD		W		CPU_T(s)	
E100	**4809.50**	**1**	212.50	3	117.47	2	1.84	3	**0.35**	**1**
E500	23887.20	2	192.47	2	143.94	3	10.47	2	5.07	2
E1000	**47549.37**	**1**	258.91	3	156.04	2	21.11	2	**11.49**	**1**
E5000	240049.59	2	220.66	2	160.39	2	106.37	3	**394.79**	**1**
M100	4957.29	3	297.46	3	114.59	2	**2.16**	**1**	0.44	2
M500	29059.77	3	314.63	3	213.63	3	**9.32**	**1**	**6.26**	**1**
M1000	49094.02	3	254.92	3	173.27	3	21.16	2	25.45	3
M5000	252617.38	3	272.07	3	183.79	3	**106.79**	**1**	314.46	2
R100	6853.28	3	267.31	3	162.55	2	**1.47**	**1**	0.37	3
R500	31191.02	3	295.42	3	216.17	3	9.21	2	**2.60**	**1**
R1000	59758.42	3	302.45	3	219.07	3	**19.58**	**1**	9.26	2
R5000	313062.13	3	330.25	3	237.76	3	105.26	2	269.51	2
CFHH starting from solutions generated by FTF										
Instance	Total_D		MDD		AVG_MDD		W		CPU_T(s)	
E100	5044.48	3	187.67	2	121.19	3	**1.42**	**1**	0.44	2
E500	**23013.49**	**1**	**182.23**	**1**	122.92	2	**10.00**	**1**	**3.28**	**1**
E1000	49083.62	2	**208.07**	**1**	**152.60**	**1**	**21.05**	**1**	19.39	2
E5000	**237120.77**	**1**	**217.40**	**1**	**159.01**	**1**	105.84	2	442.64	2
M100	4822.05	2	242.25	2	**106.56**	**1**	2.21	2	**0.38**	**1**
M500	27242.56	2	286.88	2	160.33	2	10.68	2	7.15	2
M1000	**48578.86**	**1**	**238.22**	**1**	160.90	2	**21.05**	**1**	23.63	2
M5000	**243729.68**	**1**	**235.46**	**1**	**159.69**	**1**	108.68	2	325.99	3
R100	6757.66	2	262.07	2	170.29	3	2.00	2	**0.27**	**1**
R500	**29919.34**	**1**	**291.16**	**1**	**189.61**	**1**	10.11	3	4.69	3
R1000	**55105.87**	**1**	269.81	2	173.19	2	21.05	2	**7.64**	**1**
R5000	**294064.31**	**1**	**290.10**	**1**	**191.08**	**1**	123.32	3	276.32	3
CFHH starting from solutions generated by SA										
Instance	Total_D		MDD		AVG_MDD		W		CPU_T(s)	
E100	5009.45	2	**167.15**	**1**	**108.24**	**1**	1.58	2	0.48	3
E500	23919.98	3	204.53	3	**116.29**	**1**	10.89	3	5.76	3
E1000	49564.08	3	209.48	2	157.58	3	**21.05**	**1**	19.52	3
E5000	240576.10	3	265.03	3	160.55	3	**105.42**	**1**	561.33	3
M100	**4365.29**	**1**	**211.81**	**1**	115.35	3	3.32	3	0.55	3
M500	**26985.56**	**1**	**254.40**	**1**	**139.92**	**1**	12.32	3	7.63	3
M1000	48660.98	2	240.00	2	**155.91**	**1**	**21.05**	**1**	**16.17**	**1**
M5000	247793.86	2	255.36	2	163.76	2	**106.79**	**1**	**286.68**	**1**
R100	**6611.53**	**1**	**254.33**	**1**	**158.43**	**1**	2.11	3	0.34	2
R500	30058.69	2	295.34	2	192.67	2	**9.11**	**1**	4.56	2
R1000	55526.61	2	**279.44**	**1**	**172.54**	**1**	24.58	3	9.80	3
R5000	297663.38	2	323.19	2	206.62	2	**104.00**	**1**	**239.04**	**1**

Table 3: Best and average results over 10 runs of SRHH and CFHH on FTF initial solutions

Total_D								
Instance	$SRHH_{best}$	%	$SRHH_{avg}$	%	$CFHH_{best}$	%	$CFHH_{avg}$	%
E100	5283.08	4.75	5377.82	3.04	4869.63	12.20	5044.48	9.05
E500	24366.58	4.70	25032.68	2.10	23025.33	9.95	23013.49	9.99
E1000	50741.68	2.37	51466.77	0.97	48433.95	6.81	49083.62	5.56
E5000	256778.70	1.51	257254.64	1.33	235211.66	9.78	237120.77	9.05
M100	4867.30	9.90	4954.91	8.27	4431.50	17.96	4822.05	10.73
M500	29441.70	6.17	30054.39	4.22	26648.68	15.07	27242.56	13.18
M1000	52683.04	4.95	53544.37	3.39	48118.26	13.18	48578.86	12.35
M5000	274068.83	2.38	274306.10	2.29	242640.78	13.57	243729.68	13.18
R100	6587.40	12.48	7312.07	2.85	6497.85	13.67	6757.66	10.22
R500	30903.55	7.17	32140.56	3.45	29476.04	11.46	29919.34	10.13
R1000	61396.75	4.99	62152.78	3.82	53910.16	16.57	55105.87	14.72
R5000	325434.58	2.45	325263.00	2.50	290807.64	12.83	294064.31	11.85
Avg		**5.32**		**3.19**		**12.75**		**10.83**
MDD								
Instance	$SRHH_{best}$	%	$SRHH_{avg}$	%	$CFHH_{best}$	%	$CFHH_{avg}$	%
E100	186.98	26.70	195.82	23.24	182.47	28.47	187.67	26.43
E500	205.37	-8.43	205.26	-8.37	195.18	-3.05	182.23	3.79
E1000	207.94	-9.47	207.73	-9.36	203.23	-6.99	208.07	-9.54
E5000	219.87	17.17	220.12	17.08	216.93	18.28	217.40	18.10
M100	224.37	9.76	253.31	-1.88	232.69	6.41	242.25	2.56
M500	324.19	-27.43	297.57	-16.97	327.17	-28.60	286.88	-12.77
M1000	232.33	10.01	246.65	4.46	238.48	7.63	238.22	7.73
M5000	250.78	3.39	251.18	3.24	251.59	3.08	235.46	9.30
R100	248.38	2.62	279.96	-9.76	255.07	0.00	262.07	-2.74
R500	270.99	-4.34	301.08	-15.93	304.24	-17.15	291.16	-12.11
R1000	311.81	-17.72	311.84	-17.73	203.54	23.16	269.81	-1.86
R5000	327.75	-23.17	297.30	-11.73	284.09	-6.76	290.10	-9.02
Avg		**-1.74**		**-3.64**		**2.04**		**1.66**
AVG_MDD								
Instance	$SRHH_{best}$	%	$SRHH_{avg}$	%	$CFHH_{best}$	%	$CFHH_{avg}$	%
E100	137.36	3.98	139.41	2.55	101.22	29.24	121.19	15.28
E500	135.12	2.43	141.14	-1.92	124.85	9.84	122.92	11.23
E1000	148.83	-4.46	151.53	-6.35	145.82	-2.34	152.60	-7.10
E5000	159.15	23.58	158.06	24.10	153.76	26.17	159.01	23.65
M100	115.03	12.52	132.80	-1.00	94.67	28.00	106.56	18.96
M500	171.66	10.66	181.87	5.35	158.05	17.75	160.33	16.56
M1000	171.88	12.98	171.26	13.29	158.42	19.79	160.90	18.54
M5000	178.46	9.97	179.61	9.40	164.98	16.77	159.69	19.44
R100	150.96	8.23	178.06	-8.24	156.91	4.61	170.29	-3.52
R500	184.48	0.20	194.63	-5.29	190.44	-3.02	189.61	-2.57
R1000	179.68	8.97	193.69	1.87	151.06	23.47	173.19	12.26
R5000	193.20	1.32	188.43	3.76	192.11	1.88	191.08	2.40
Avg		**7.53**		**3.13**		**14.35**		**10.43**

in a sub-region, subsequently minimising the overall distance of a solution by reassigning outlying tasks to a better sub-region. These heuristics help intensify the search space by focusing only on minimising total distance, in order to provide a better solution. The Interchange heuristic, which tends to both minimise the total distance and maintain the balance of the allocation of tasks, attempts to intensify the search space in the same way as the previous three, despite the fact that it does not affect the balancing state of the solution. The Balancing heuristic only takes the balancing of sub-regions into account. The effect of this heuristic is to diversify the search space, in order to avoid getting trapped in a local optimum; however, there is also the possibility of exploiting the search space if it leads to a solution with less total cost compared to the previous solution. The obtained results indicate that although the effects of these methods are very dependent on when and how long they are applied to a solution in the framework, they have still been designed to be able to explore different areas of the search space effectively.

The only exception is that SRHH could not improve the MDD measurement across the average of all instances (-1.74 and -3.64 for the best and average results). This is likely due to the lack of learning mechanism to guide this hyper-heuristic, leading to an imbalance between intensification and diversification when traversing the search space. Despite this, the overall improvement yielded on all instances on Total_D and the AVG_MDD measurement of the corresponding instances is an indicator of an improvement in the solution compared to the quality of the initial solution.

Comparing the best values obtained over all 12 instances, CFHH yielded approximately 12.75%, 14.35% and 2.04% improvement for Total_D, MDD, and AVG_MDD respectively, while SRHH improved by 5.32% and 7.53% but only on Total_D and AVG_MDD, a deterioration in quality is observed on average in terms of MDD. In the case of the average values obtained, CFHH achieved roughly 10.50 on both Total_D and AVG_MDD and 2% in MDD, while SRHH improved the initial solutions by approximately 3.1% on Total_D and AVG_MDD out of the three measurements.

Since we use the same low-level heuristics in both frameworks, the difference in performance of CFHH compared to SRHH is likely due to the self-adaptive nature of the hyper-heuristic, appropriatly controlling the amount of exploitation/exploration by adjusting parameters α, β and γ in every iteration. Meanwhile, in SRHH, choosing the low-level heuristic randomly may lead the solution to the area of the search space where it is difficult to move quickly to another area. For instance, applying the low-level heuristics which

only pay attention to minimising distance and not workload balancing, such as Domino, Pair or even Join, might lead the space to an area with very high quality in terms of overall total distance and maximum distance but very low quality in relation to balancing. In this situation, moving the solution space back to a space resulting in a balanced solution might cause a penalty in terms of the objective function value.

4.3.1. Compactness validation

As mentioned earlier, the framework presented in this paper is used to partition the maintenance tasks within the Danish railway system, allocating a set of maintenance tasks to a set of maintenance crew members. This phase takes place before maintenance planning in the ERTMS signalling system. In this way, the system attempts to ensure that no distant tasks are assigned to any crew member in the scheduling phase. In any scheduling problem, the main objective is to minimise total cost (i.e. a weighted function of the number of routes and their length) and to ensure that all tasks are completed. Therefore, the density of the tasks in each sub-region can affect the length of routes and subsequently the total cost in the scheduling phase.

To calculate the cohesion of the sub-regions, in addition to results found in other problem-specific measurements, we calculate the validity factor of compactness, which is a well-known measurement in the literature [31]. Compactness is a validation factor employed to measure the cohesion of objects in a cluster by mean normalised variance and indicates how well data points are clustered in terms of object homogeneity. In other words, this index is formulated to decide whether or not a given subset is internally dense. Essentially, the higher this value, the lower average cohesion of the cluster:

$$\mathcal{C} = \sum_{k=1}^{K} \sum_{i=1}^{N} P_{k,i} \left\| X_i - \mu_k \right\|^2 \tag{9}$$

where $\mathcal{C}$ is the compactness value for the clusters that need to be minimised, K is the number of the clusters, N is the number of tasks, P is the partition matrix and $P_{i,k}$ specifies if task X_i is in cluster k. μ_k is the centre of cluster k.

Figure 5 presents the comparative results of the compactness measurement of the initial solution obtained using FTF, and after applying CFHH and SRHH as above. The compactness of the solutions obtained by SRHH and CFHH is shown as a ratio of their compactness measurement to the

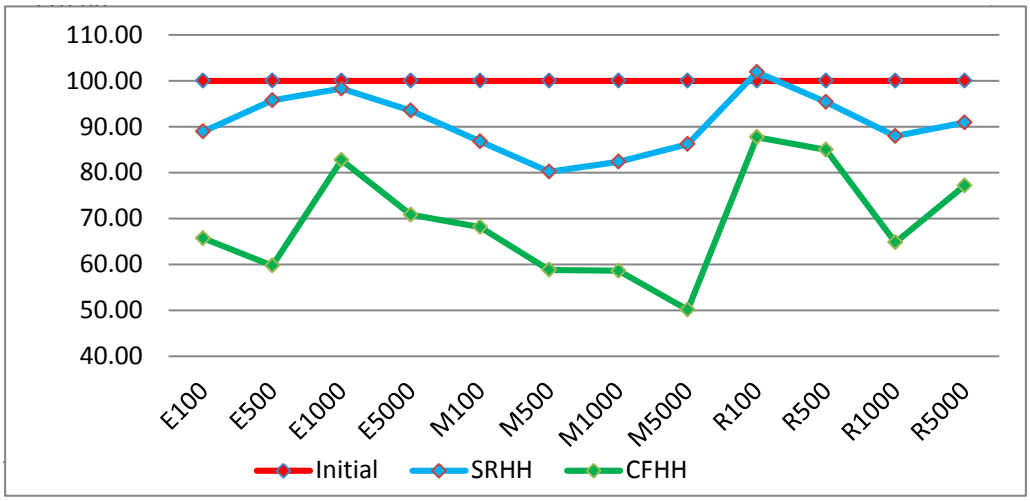


Figure 5: Compactness of solutions generated by FTF, and following improvement by CFHH and SRHH

compactness measurement of the initial clustering result (FTF). As a lower compactness measurement indicates more dense clusters, it is evident that CFHH generates sub-regions that are much more compact than SRHH and the initial solution generated using FTF. It is also notable that CFHH improves approximately 31% on the compactness of the initial solution, while SRHH improves 9.30% of the measurement, respectively, on average across all instances.

One anomaly is the performance of SRHH on the R100 instance, where it cannot improve the compactness of the initial solution, obtaining a compactness factor roughly 2% worse. However, this outcome is not unanticipated, as SRHH generated the worst result for R100 in terms of the average maximum distance (-8.24%) in Table 3, as exemplified earlier.

4.4. Detailed low-level heuristic performance

To assess the impact of different low-level heuristics during a run, Table 4 gives the number of calls of each low-level heuristic by CFHH, during the first 100 and last 100 iterations, for the run where the best solution for each of the largest instances was found (E5000, M5000 and R5000).

From the number of calls during the first 100 iterations, it is clear that in the early stages of the search, different low-level heuristics are selected more frequently than in the last 100 iterations of the search. It is interesting that

Table 4: Number of heuristic calls during the first 100 and last 100 iterations of CFHH on large instances

	First 100 calls			Last 100 calls		
Heuristic	E5000	M5000	R5000	E5000	M5000	R5000
Balancing	2	3	1	19	15	**21**
Domino	**83**	**60**	**50**	**23**	16	20
Join	9	18	1	**23**	**37**	20
Interchange	6	5	1	16	16	19
Pair	0	14	47	19	16	20

during the first 100 iterations, Domino is selected most often (83, 60 and 50) and Balancing (2, 3 and 1) is selected least often for all three instances. This indicates that the hyper-heuristic recognises the low-level heuristics which intensify and diversify in terms of minimising distance - even in the early stages of the search. Applying the Domino heuristic, which only causes an improvement to total distance, is still an indicator of greedy behaviour in the framework at this point in time. Interestingly the Pair heuristic is selected far more often for the Random instance than the Exact instance, indicating that different low-level heuristics are more or less effective depending on the type of instance being solved. This provides some justification for using a hyper-heuristic approach, mixing multiple low-level heuristics as appropriate during a particular search.

From the last 100 calls it is noticeable that the spread of calls over the low-level heuristics reduces as the search progresses. This suggests that there is less improvement towards the end of the search. If no improvement is found for a large number of iterations, the only component that will contribute towards the choice function score is f_3 (time since last called). As such, the choice function will behave more like a simple random hyper-heuristic when fewer improvments are made.

In Table 5 we show the proportion of calls to each heuristic over the full run of the same examples as above, with the relative rank of each low-level heuristic given in brackets. Note that these percentages have been rounded to 1 decimal place, and as a result may not all add up to exactly 100%.

From the overall ratio of calls we see that in general, across the three instances, the Join and Interchange heuristics appear among the top two heuristics, whereas the Balancing heuristic is always selected the least often. Join and Interchange explore the solution space in slightly different ways

Table 5: Percentage of calls (rounded to 1 d.p.) and relative rank of low-level heuristics selected by CFHH on large instances

Heuristic	E5000 Call % (rank)	M5000 Call % (rank)	R5000 Call % (rank)
Balancing	7.9 (5)	12.1 (5)	14.5 (5)
Domino	12.5 (2)	15.4 (4)	16.6 (4)
Join	**58.8 (1)**	**39.9 (1)**	**34.1 (1)**
Interchange	9.9 (4)	16.8 (2)	17.9 (2)
Pair	10.9 (3)	15.9 (3)	16.8 (3)

compared to the other low-level heuristics. Join is the only low-level heuristic that tries to minimise total distance, not by dealing with outliers but by joining close tasks from different sub-regions. There may be many close tasks which belong to different sub-regions, which can be joined to the same sub-region to improve the total distance in different ways. This is particularly important when the hyper-heuristic cannot improve the solution by only dealing with outliers, whether the best assignment is the current sub-region or the solution space gets stuck in a local optima. Interchange is designed in a way that not only improves the solution without being limited to dealing with the outliers, but also takes care of balancing between sub-regions. The rank of the Balancing heuristic is perhaps not a surprise, as it doesn't attempt to minimise the total distance directly. However, the number of calls of this heuristic shows that the parameter γ has been appropriately controlled to explore the search space by calling the Balancing heuristic during the search despite potential poor performance in objective function terms.

4.5. Trend of solution improvement during a run using CFHH

Figure 6 and Figure 7 show the trend of improvement for three different measures, using the run in which the best solution for instance E5000 was found by CFHH. The *y-axis* in Figure 6 is the total cost of driving distance (Total_D). In Figure 7(b), it is the maximum distance of a crew to a task (MDD - red plot) and the average of the maximum distance obtained by all of the crew over the iterations (AVG_MDD - green plot). Because the heuristics selected by CFHH shown almost the same trend in all large instances in the previous subsection, only the trend of one instance is investigated.

It is evident that CFHH shows an overall trend of improvement, in terms of minimising total distance throughout the run. In early iterations, it seems

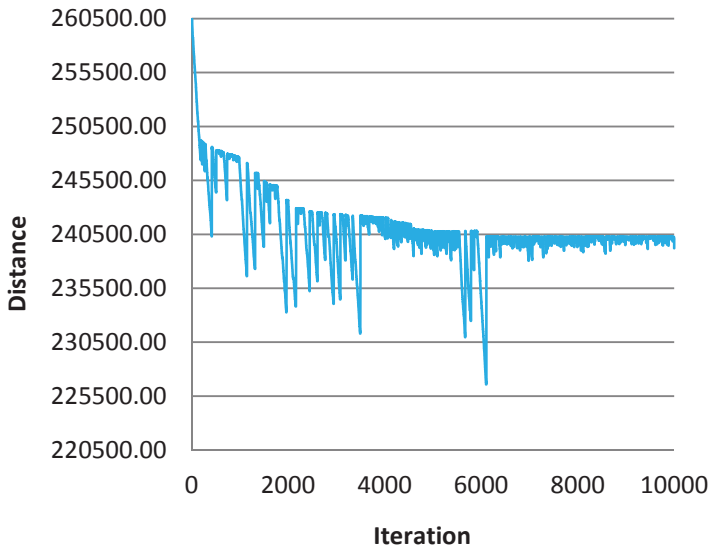


Figure 6: Trend of improvement of Total_D over a sample run of CFHH on instance E5000

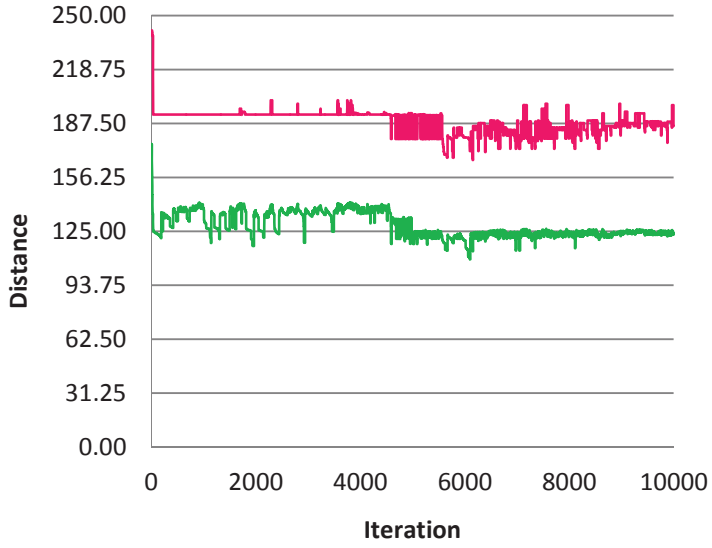


Figure 7: Trend of improvement of MDD (red) and AVG_MDD (green) over a sample run of CFHH on instance E5000

that CFHH improves the initial solution quickly, however the best solution fluctuated between 1000 and 4000 iterations. One possible explanation might be due to punishment of the Balancing heuristic after each call, since whenever it is applied, it incurs a bad penalty in terms of total distance. This could be mitigated by somehow considering the balancing of the solution as an objective, instead of calculating only the penalty of an increase in total distance. In this way, Balancing could be called more often and consequently lead to less fluctuation in solution quality compared to the current trend. It is notable that the performance stabilises after approximately half of the iterations pass. Similarly, the average of the maximum distance (AVG_MDD) in Figure 7 (green plot) shows the same trend with a significant drop in early iterations, followed by a fluctuation and finally remaining steady with marginal changes in the latter stages.

In contrast to Total_D and AVG_MDD, the maximum distance (MDD) plot (red plot in in Figure 7) fluctuates more in the second half of the search than in the early stages, indicating that the low-level heuristics can be combined in order to improve all of the embedded factors (minimising total distance, minimising maximum distance and balancing the sub-regions) over time, with the hyper-heuristic adapting appropriately through the parameters α, β and γ.

5. Conclusions

In this study, we have proposed a perturbative hyper-heuristic framework using choice function heuristic selection, which improves the allocation of maintenance tasks to a set of crew members in the Danish Railway system. Our framework generates a set of *sub-regions* of maintenance tasks, with each sub-region representing the working area of a single crew member. It is desirable to minimise the distance between any two tasks in each sub-region, in order to ensure a fast response in the case of recovery failure. Using the concept of outliers, tasks which are a long distance from the starting location of each crew member, tasks are reassigned to different sub-regions using one of five low-level heuristics, with the intention of reducing the maximum distance between two tasks within the same sub-region.

An adaptive choice function hyper-heuristic has been used to search the space of low-level heuristics. Once an appropriate allocation of maintenance tasks have been decided, the sub-regions can be passed on to a routing algorithm to decide the individual routes each crew member should take. Our

results show that, higher quality initial solutions do not always lead to higher quality solutions following improvement by the hyper-heuristic. Using initial solutions which are slightly lower quality does not restrict the search to particular regions of the search space, allowing hyper-heuristics to traverse the search space with more flexibility. An adaptive choice function (CFHH) was shown to be able to adaptively learn which heuristics to apply at a given stage of the search, balancing intensification and diversification within the search, outperforming simple random search (SRHH). The results obtained using CFHH were demonstrated to have a a high degree of cohesion, in terms of compactness ratio, a desirable property in preparation for the subsequent routing phase. Future work will seek to link the clustering phase addressed in this paper to the scheduling phase, where the sub-regions defined are used to schedule and route individual crew members.

Acknowledgements

This work has been partially funded by the DAASE project, EPSRC programme grant EP/J017515/1.

References

[1] Banedanmark. The signalling programme - a total renewal of the danish signalling infrastructure. Technical report, Trafikministeriet, 2009.

[2] Pavol Barger, Walter Schon, and Mohamed Bouali. A study of railway ertms safety with colored petri nets. In *The European Safety and Reliability Conference (ESREL'09)*, volume 2, pages 1303–1309. Taylor & Francis Group, 2009.

[3] David Bredstrom and Mikael Ronnqvist. Combined vehicle routing and scheduling with temporal precedence and synchronization constraints. *European journal of operational research*, 191(1):19–31, 2008.

[4] E. K. Burke, G. Kendall, and E. Soubeiga. A tabu-search hyperheuristic for timetabling and rostering. *Journal of Heuristics*, 9(6):451–470, 2003.

[5] Edmund K Burke, Matthew Hyde, Graham Kendall, Gabriela Ochoa, Ender Özcan, and John R Woodward. A classification of hyper-heuristic approaches. In *Handbook of metaheuristics*, pages 449–468. Springer, 2010.

[6] Jean-Francois Cordeau, Gilbert Laporte, and Anne Mercier. A unified tabu search heuristic for vehicle routing problems with time windows. *Journal of the Operational research society*, 52(8):928–936, 2001.

[7] Peter Cowling, Graham Kendall, and Eric Soubeiga. A hyperheuristic approach to scheduling a sales summit. In *International Conference on the Practice and Theory of Automated Timetabling*, volume 7245 of *Lecture Notes in Computer Science*, pages 176–190. 2001.

[8] Peter Cowling, Graham Kendall, and Eric Soubeiga. A parameter-free hyperheuristic for scheduling a sales summit. In *Proceedings of the 4th Metaheuristic International Conference (MIC 2001)*, pages 127–131. Citeseer, 2001.

[9] John H. Drake, Ender Özcan, and Edmund K. Burke. Modified choice function heuristic selection for the multidimensional knapsack problem. In *Proceedings of the International Conference on Genetic and Evolutionary Computing (ICGEC2014)*, Advances in Intelligent Systems and Computing, pages 225–234. Springer, 2014.

[10] Adnen El Amraoui and Khaled Mesghouni. Colored petri net model for discrete system communication management on the european rail traffic management system (ertms) level 2. In *Computer Modelling and Simulation (UKSim), 2014 UKSim-AMSS 16th International Conference on*, pages 248–253. IEEE, 2014.

[11] H. Fisher and G.L. Thompson. Probabilistic learning combinations of local job-shop scheduling rules. In *Factory Scheduling Conference*, Carnegie Institute of Technology, 1961.

[12] Marshall L Fisher and Ramchandran Jaikumar. A generalized assignment heuristic for vehicle routing. *Networks*, 11(2):109–124, 1981.

[13] Pablo Garrido and Carlos Castro. Stable solving of cvrps using hyperheuristics. In *Proceedings of the 11th Annual conference on Genetic and evolutionary computation*, pages 255–262. ACM, 2009.

[14] Jonathon Gibbs, Graham Kendall, and Ender Özcan. Scheduling english football fixtures over the holiday period using hyper-heuristics. In *Proceedings of Parallel Problem Solving from Nature (PPSN 2011)*, volume

6238 of *Lecture Notes in Computer Science*, pages 496–505. Springer, 2011.

[15] ID Giosa, IL Tansini, and IO Viera. New assignment algorithms for the multi-depot vehicle routing problem. *Journal of the operational research society*, 53(9):977–984, 2002.

[16] Giovani Guizzo, Gian Mauricio Fritsche, Silvia Regina Vergilio, and Aurora Trinidad Ramirez Pozo. A hyper-heuristic for the multi-objective integration and test order problem. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2015)*, pages 1343–1350. ACM, 2015.

[17] Berna Kiraz, A. Sima Uyar, and Ender Özcan. Selection hyper-heuristics in dynamic environments. *Journal of the Operational Research Society*, 64(12):1753–1769, 2013.

[18] Jan Karel Lenstra and AHG Kan. Complexity of vehicle routing and scheduling problems. *Networks*, 11(2):221–227, 1981.

[19] Tomas Lidén. Survey of railway maintenance activities from a planning perspective and literature review concerning the use of mathematical algorithms for solving such planning and scheduling problems. 2014.

[20] Eunice Lopez-Camacho, Hugo Terashima-Marin, and Peter Ross. A hyper-heuristic for solving one and two-dimensional bin packing problems. In *Proceedings of the 13th annual conference companion on Genetic and evolutionary computation*, pages 257–258. ACM, 2011.

[21] Mashael Maashi, Graham Kendall, and Ender Ozcan. Choice function based hyper-heuristics for multi-objective optimization. *Applied Soft Computing*, 28:312–326, 2015.

[22] Jairo R Montoya-Torres, Julian Lopez Franco, Santiago Nieto Isaza, Heriberto Felizzola Jimenez, and Nilson Herazo-Padilla. A literature review on the vehicle routing problem with multiple depots. *Computers & Industrial Engineering*, 79:115–129, 2015.

[23] Ender Ozcan, Mustafa Misir, Gabriela Ochoa, and Edmund K. Burke. A reinforcement learning - great-deluge hyper-heuristic for examination

timetabling. *International Journal of Applied Metaheuristic Computing*, 1(1):39–59, 2010.

[24] Ambika Prasad Patra, Pierre Dersin, and Uday Kumar. Cost effective maintenance policy: A case study. *International Journal of Performability Engineering*, 6(6), 2010.

[25] Fan Peng. *Scheduling of track inspection and maintenance activities in railroad networks*. PhD thesis, University of Illinois at Urbana-Champaign, 2011.

[26] David Pisinger and Stefan Ropke. A general heuristic for vehicle routing problems. *Computers & operations research*, 34(8):2403–2435, 2007.

[27] Anand Rajaraman, Jeffrey D Ullman, Jeffrey David Ullman, and Jeffrey David Ullman. *Mining of massive datasets*, volume 1. Cambridge University Press Cambridge, 2012.

[28] Rob Redekker. Working towards an ertms maintenance regime. In *Issue 4 2008*. European Railway Review, 2008.

[29] David M Ryan, Curt Hjorring, and Fred Glover. Extensions of the petal method for vehicle routing. *Journal of the Operational Research Society*, 44(3):289–296, 1993.

[30] Eric Soubeiga. *Development and application of hyperheuristics to personnel scheduling*. PhD thesis, University of Nottingham, 2003.

[31] Pang-Ning Tan, Michael Steinbach, and Vipin Kumar. Data mining cluster analysis: basic concepts and algorithms. *Introduction to data mining*, 2013.

[32] Libertad Tansini, María E Urquhart, and Omar Viera. Comparing assignment algorithms for the multi-depot vrp. *Reportes Técnicos 01-08*, 2001.

[33] R Tapsall. Application of ertms to the diverse australian network. In *AusRAIL PLUS 2003, 17-19 November 2003, Sydney, NSW, Australia*, 2003.