Noname manuscript No.
(will be inserted by the editor)

A multi-arm bandit neighbourhood search for routing and scheduling problems

Yujie Chen · Peter Cowling · Fiona Polack · Philip Mourdjis

Received: date / Accepted: date

Abstract Local search based meta-heuristics such as variable neighbourhood search have achieved remarkable success in solving complex combinatorial problems. Local search techniques are becoming increasingly popular and are used in a wide variety of meta-heuristics, such as genetic algorithms. Typically, local search iteratively improves a solution by making a series of small moves. Traditionally these methods do not employ any learning mechanism.

We treat the selection of a local search neighbourhood as a dynamic multi-armed bandit (D-MAB) problem where learning techniques for solving the D-MAB can be used to guide the local search process. We present a D-MAB neighbourhood search (D-MABNS) which can be embedded within any meta-heuristic or hyperheuristic framework. Given a set of neighbourhoods, the aim of D-MABNS is to adapt the search sequence, testing promising solutions first. We demonstrate the effectiveness of D-MABNS on two vehicle routing and scheduling problems, the real-world geographically distributed maintenance problem (GDMP) and the periodic vehicle routing problem (PVRP). We present comparisons to benchmark instances and give a detailed analysis of parameters, performance and behaviour.

Keywords Meta-heuristic · Local search · Vehicle routing

Yujie Chen
University of York, Heslington, York, YO10 5GE, UK
E-mail: yujiechen369@163.com

Peter Cowling
E-mail: peter.cowling@york.ac.uk

Fiona Polack
E-mail: fiona.polack@york.ac.uk

Philip Mourdjis
E-mail: pjm515@york.ac.uk

1 Introduction

Complex combinatorial problems are often too time consuming to solve with exact methods; heuristic approaches that find acceptable solutions in reasonable time are often preferred. Meta-heuristics are popular, as they are capable of producing good solutions to a wide range of problems. Meta-heuristics employ many different algorithmic schemes for *exploration* of the search space. Variable neighbourhood search (VNS) [20], tabu search [10], and hybrid genetic algorithm [35] all have different exploration behaviours.

However, there is little work targeting the *exploitation* technique of search. Most approaches apply simple local search (LS) methods: starting from a feasible solution, iteratively move to a better solution by selecting it from a neighbourhood of the current solution.

The *neighbourhood* of a solution is the set of other solutions that can be obtained from it using small modifications. The process of modification is called a *move* and the size of a neighbourhood is defined by the number of moves that can operate on the current solution.

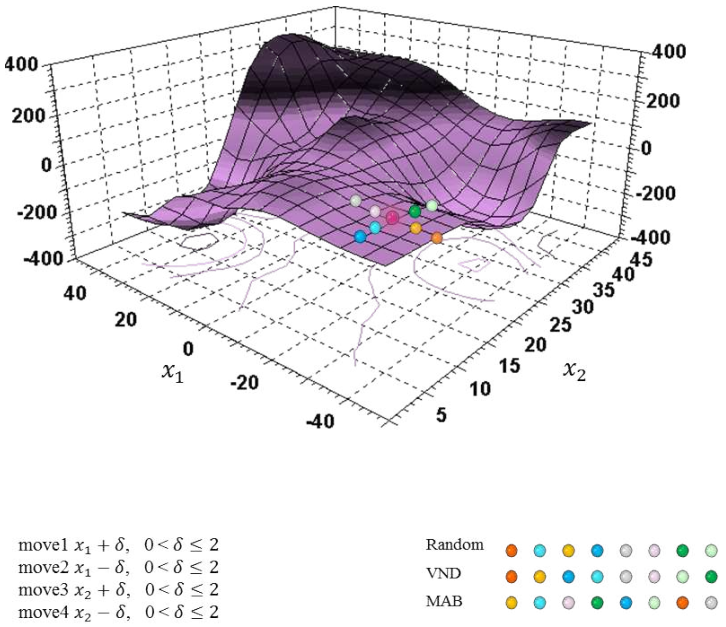


Fig. 1: Search strategies on an optimisation problem with two variables, $x_1, x_2 \in \mathbb{Z}$

In this paper, we introduce a machine learning-based local search, referred to as the dynamic multi-arm bandit neighbourhood search (D-MABNS). D-MABNS is inspired by well-known decision making models for the multi-

armed-bandit (MAB) problem. The major difference to existing approaches is that, instead of pre-specifying the examining order of neighbour solutions (e.g. lexicographic search [15]) at each iteration of local search, D-MABNS dynamically adapts the search sequence to test promising solution first. D-MABNS can be embedded in any meta-heuristic or hyperheuristic framework.

Figure 1 illustrates three approaches to searching through the neighbourhood of the current solution, on the search space of a simple discrete optimisation problem that has two variables. The purple ball shows the current solution, and we define four types of move that generate eight neighbours of the current solution. The aim is to find an improved solution (light green ball) as efficiently as possible. In Figure 1, we see that checking potential moves in a random order finds the improved move after 8 checks; checking in a prespecified order, such as variable neighbourhood decent (VND) [20], finds the improved move after 7 checks, whilst following a statistically-based sequence as in MAB, the improved solution is found after 6 checks.

The remainder of this paper is organized as follows. Section 2 reviews efficient search strategies for combinatorial problems and introduces MAB problems. We introduce D-MABNS in Section 3, and demonstrate its use on risk-driven geographical distributed maintenance scheduling problems (GDMPs) from real-world scenarios in Section 4. Section 5 analyses the performance of D-MABNS on periodic vehicle routing problems (PVRPs) using benchmark instances. The findings are summarised in Section 6.

2 Related Work

For simple combinatorial problems, a number of approaches have been proposed to reduce the computational time required to search a neighbourhood.

Sequential search (SS) decomposes the moves of a neighbourhood into partial moves that are cost-independent. A decomposition is cost-independent if the fitness change for the complete move is equal to the sum of fitness changes of all the partial moves. To avoid checking moves that are unable to produce improvement, SS constructs a complete move by first sequentially determining good partial moves. SS has been applied to the travelling salesman problem [23] and the capacitated vehicle routing problem (CVRP) [15]. Experimental results show that SS uses significantly less computation time than conventional neighbourhood search [15].

Neighbourhood restriction approaches reduce the CPU time spent on each iteration of LS. For example, Toth and Vigo [34] derive "granular neighbourhoods" from a neighbourhood by discarding moves that have none of the "promising" elements that would be likely to belong to high quality solutions. In [34], the elements are the arcs of a routing problem; an element is labelled as promising based on characteristics such as arc length, incidence of the arc to the depot, and whether the arc is used in one of the best solutions encountered so far. The "granular neighbourhoods" approach is embedded in a tabu search, and the algorithm is tested on VRP instances with up to around 500

customers [34]. The experimental results show that the method is efficient in computational time.

Fast guided local search [36] also uses a neighbourhood restriction, in which moves are only evaluated if they belong to an activated sub-neighbourhood. Voudouris et al. [36] present examples from various problem domains, using different measures to create and evaluate sub-neighbourhoods; effective solutions depend on derivation of suitably small sub-neighbourhoods.

2.1 Solutions for complex combinatorial problems

Approaches that are efficient for simple combinatorial problems are rarely appropriate for complex combinatorial problems, where there are additional properties or constraints to consider. The design of efficient local search for complex combinatorial problems is time-consuming and requires expert domain knowledge. Typically, solvers with multiple neighbourhood structure are designed (e.g. [26]). The search framework then includes decision-making to select a neighbourhood for each iteration. We consider three classes of search algorithm, based on the nature of candidate selection.

The first group of algorithms use *first improvement* or *best improvement* heuristics, referred to as *low-level heuristics (LLHs)*. The LLHs need to further define the search sequence within a selected neighbourhood (e.g. random, lexicographic, etc.). At a higher level, a decision-making process selects between neighbourhoods in either a pre-defined or a random order. VNS, a typical example of this class of algorithms, has been applied to various problem domains [20]. The hyperheuristic literature presents similar structures with some learning-based adaptive selection, for example, the tabu-search hyperheuristic [3], and the reinforcement learning hyperheuristic [7]. These algorithms employ a relatively inflexible search, in which neighbourhoods are tested in turn throughout.

The second group of algorithms typically use a random move from a selected neighbourhood structure. The best-studied example uses adaptive operator selection (AOS) [14], typically as part of an evolutionary algorithm. In recent years, AOS has also been applied in hyperheuristic approaches [32, 31]. Comparing these algorithms to the first group, the uncontrolled use of LLHs (random moves rather than improvement heuristics) may result in unproductive revisiting of the same move, and thus to inefficient search.

The third approach, proposed by Ropke [30], is adaptive large neighbourhood search (ALNS). Here the LLHs are not specified in advance and are instead created programmatically from a known set of destroy and construct methods; the system learns which combinations work effectively and focuses the search on these.

2.2 Using machine learning to guide search

Self-adaptive approaches to guiding neighbourhood search combine machine learning and classical heuristic search techniques. Many approaches use the historical performance of operators to adjust future operator utilisation. In the AOS literature, the key components are credit assignment and decision-making [11].

Credit assignment describes a way to evaluate the quality of an operator. Normally, the credit assigned to an operator is based on how often that operator makes a contribution or an improvement [24], or on how much total contribution the operator has made so far [32]. The latter measurement is more sensitive to the fitness landscape of the problem instance, and is usually combined with a reward rescaling method. To evaluate each option, most approaches consider either the instantaneous reward value after an operator has been applied or the average reward over a sliding time window. An alternative is to consider an extreme value [12], on the basis that the generation of rare but highly beneficial improvements matter more than frequent small improvements.

Decision-making determines how to select the next operator, based on past credits. Probability matching and adaptive pursuit methods are probably the most widely applied mechanisms [33]. Both methods update the option selection probability according to its evaluation value; these probabilities are then used for selection in a "weighted roulette wheel"-like process. An alternative mechanism adds an exploration term to the quality evaluation function, and the decision-making process is deterministic, based on the evaluation values (e.g. using the UCB1 algorithm [1]).

2.3 Multi-arm bandit problems

The above decision making process can be considered as a multi-arm bandit problem in which the goal is to maximise the total rewards collected over time. Auer et al. [1] define a typical static MAB problem (S-MABP) in which the K arms each have an independent reward probability p_i, where $p_i \in [0, 1]$. At each time step t, the player should select an arm j; with probability p_j the arm receives a reward $r_t = 1$, otherwise $r_t = 0$.

To solve a similar problem in a changing environment, Da Costa et al. [11] describe a dynamic MAB problem (D-MABP) in which each arm has a uniform reward distribution, from the interval $[p_{i,t} - 1, p_{i,t} + 1]$ at time step t. Thus, for every T time steps, the mean value of reward distribution of each arm $p_{i,t}$ varies. Further, the reward distribution of all arms change simultaneously.

In the next section, we use the fundamental concepts of D-MABP to address the dilemma of exploration and exploitation during a neighbourhood search process. We introduce techniques such as mapping solution fitnesses to rewards and dynamic neighbourhood management, to fit the characteristics of neighbourhood search.

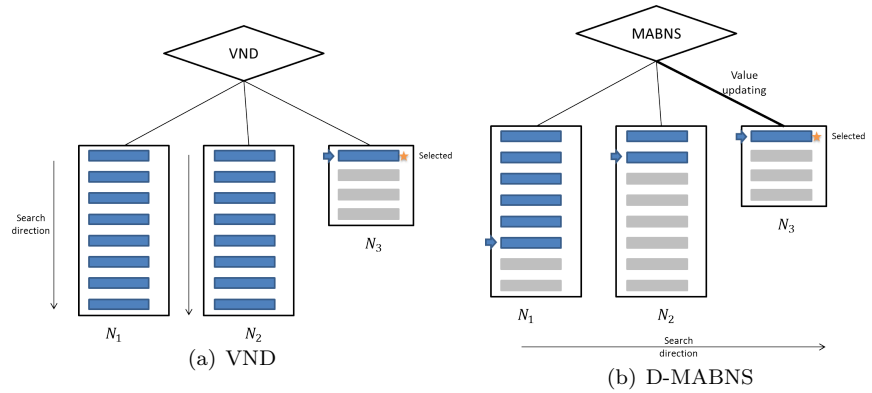
3 Dynamic Multi-arm bandit neighbourhood search

Our dynamic multi-arm-bandit neighbourhood search (D-MABNS) is inspired by many of the techniques reviewed in Section 2. D-MABNS maintains a search trace within a neighbourhood, as in improvement heuristics, but also introduces selection strategies between the neighbourhoods, like the methods that statistically select moves. Furthermore, D-MABNS uses some of the techniques from simple combinatorial search to discard unpromising moves during the search.

3.1 D-MABNS overview and framework

The goal of search is to efficiently find an improvement direction in each iteration of local search process, so as to reach a local optimum quickly. First, consider a typical VND approach to solving a CVRP. Neighbourhoods defined by moves, such as "2Opt" and "Cross-exchange" [18], are searched sequentially until an improved solution is found, as shown in Figure 2(a). The search can be made more efficient, for instance, by ordering elements or by discarding unpromising moves (Section 2), but the search is essentially over a set sequence of neighbourhoods.

The D-MABNS design is based on the observation that it can be inefficient to check the whole of one move's neighbourhood before considering the neighbourhood of the next move. D-MABNS uses a search pointer within each neighbourhood, and dynamically decides *when* to examine the next neighbour, from *which* neighbourhood structure. At each decision point, D-MABNS looks at the neighbourhood that has the best current expectation (Figure 2(b)).



(a) VND (b) D-MABNS

Fig. 2: The search strategies of VNS and of D-MABNS

A neighbourhood $N_k(x)$ represents the set of neighbours that can be obtained by one defined move from the current solution x. For a given problem,

there is a finite set of neighbourhood structures $N_k, (k = 1, \ldots, k_{max})$. In D-MABNS (Figure 2(b)) each neighbourhood structure N_k is associated with a value v_k that is used to evaluate the quality of N_k, based on the neighbours seen so far from this neighbourhood. After a selected neighbour $x' \in N_k(x)$ has been tested, a reward (or punishment) r_k is given to $N_k(x)$ and the value v_k is updated. Then, $v_k, (k = 1, ..., k_{max})$ is used to decide which neighbourhood to look at next.

In the following sections, we look in more detail at the D-MABNS decision making process, based on the MAB model. We also propose dynamic neighbourhood updating, to further improve the search efficiency of LS.

3.2 Decision making

In traditional static MAB problems, the MAB arms are rewarded either 0 or 1 according to a Bernoulli distribution [1]. Applying the MAB model to neighbourhood selection, each neighbourhood is treated as an arm and is characterized by a fitness distribution.

The fitness distribution of each neighbourhood can be approximated by empirical investigation, but identifying the neighbourhood to check next is an exploration and exploitation dilemma in the design of D-MABNS. A greedy selection strategy would select the neighbourhood with the current maximum estimated v_k. However we can introduce exploration into the deterministic decision process by using approaches such as the Upper Confidence Bound algorithm (UCB1) [1].

Formally, we denote n_k as the number of neighbours from the k_{th} neighbourhood seen so far, and v_k is the forecast value of the corresponding neighbourhood (measured from rewards collected). The UCB1 algorithm selects the neighbourhood which maximizes the value below [1]:

$$v_k + \sqrt{\frac{2log\sum_i^{k_{max}} n_i}{n_k}} \tag{1}$$

In Equation 1, the square-root component represents exploration, encouraging the search into less-explored neighbourhoods. The neighbourhood quality estimation, v_k, represents exploitation by preferring the neighbourhoods that have the best expectation value. The UCB1 algorithm ensures that each neighbourhood can be chosen, but that the elapsed time between selections of a sub-optimal neighbourhood increases exponentially.

3.2.1 Quality value estimation v_k

D-MABNS requires a neighbourhood quality value estimation mechanism to forecast the value of a neighbourhood, v_k. Some existing approaches to credit assignment are outlined in Section 2.2. In our approach, we employ *exponential smoothing* [16], often referred to in the operator-selection literature as additive

relaxation [11,13]. Unlike simple average values, exponential smoothing provides a decay mechanism; this is necessary later (Section 3.3), when D-MABNS is applied to a dynamic MAB problem in which new solutions are accepted during search. The value v_k of the selected neighbourhood is updated whenever a reward r_k is received, using Equation 2 [16], such that v_k is a weighted average of the previous smoothed value and the latest reward r_k. A smoothing factor α ($0 < \alpha \leq 1$) controls the decay rate of historical reward observations.

$$v_k = (1 - \alpha)v_k + \alpha r_k \tag{2}$$

3.2.2 Reward functions and scaling

In combinatorial search problems, we usually define a fitness function to measure the quality of solutions. Rewards are generally related to the fitness of solutions found in a neighbourhood. However, directly using the fitness as a reward can unbalance the parts of Equation 1, biasing the search either towards exploitation or towards exploration.

To reduce the bias, a scaling factor can be added to either the exploration or exploitation parts of Equation 1. However, fitness measures and the range of fitness values are domain-dependent, so the scaling factor needs to be determined experimentally on a problem-specific basis. Here, we propose an *adaptive reward function* that scales fitness into the range of $[0, 1]$, and does does not need any prior domain knowledge to balance Equation 1.

For a neighbourhood N_k, we select and test a solution x' from the set of neighbours of the current solution, x. The fitness of solution x' is returned by the function $f(x')$, and $\delta_f = f(x') - f(x)$ is the difference in fitness between the current solution and the tested solution. We record the maximum and minimum changes of fitness over the last W tests of neighbours of x, denoted as δ_f^{max} and δ_f^{min}, respectively. We refer to W as a time window parameter, which can be set manually to any suitable value.

For a minimization problem, in which $\delta_f < 0$ indicates an improvement from the current solution, we design an exponential reward function RF that aims to reward a bigger improvement with a score closer to 1 and a worsening solution with a score closer to 0. In selecting an RF, we note that, during a neighbourhood search, it is common to have already tested many worse neighbours from one or multiple neighbourhoods before finding any improvement. The fitness of these solutions gives an indication of the quality of a neighbourhood, so simply assigning a zero reward to a worse solution is not appropriate. Figure 3 shows how fitness change (x axis) maps to rewards (y axis) using the exponential reward function; a more detailed experimental analysis appears in Section 4.3.1.

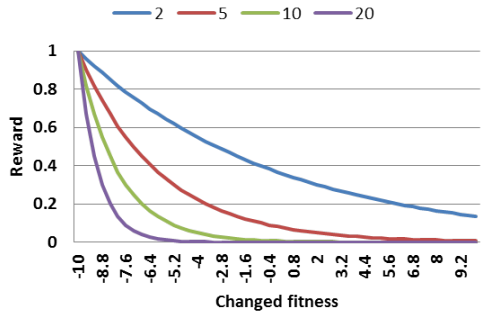


Fig. 3: Examples of exponential reward function, calculated from example fitness values in the range of -10 to 10, using different scaling factors, a. $RF = exp(-a(\delta_f - \delta_f^{min})/|\delta_f^{max} - \delta_f^{min}|)$, where $a = 2, 5, 10, 20$.

3.3 Dynamic neighbourhood management

So far, we have only talked about a search process moving from the current solution x towards x', where $x' \in N_k(x)$. Solution x' is one *move* from x and thus typically has a similar solution structure; if the same move were to operate on solutions x and x', it would also generate solutions with similar structures. Furthermore, depending on the problem and neighbourhood structure design, $N_k(x)$ and $N_k(x')$ may share many neighbours. If we denote the fitness distribution of $N_k(x)$ as $F(N_k)$ and of $N_k(x')$ as $F(N'_k)$, then $F(N_k) \simeq F(N'_k)$. There is an obvious benefit of the dynamic model, in that we do not need to build the understanding of each neighbourhood structure from scratch at each LS iteration. However, a mechanism is needed to record fitnesses changes.

Where the fitness distribution of each neighbourhood structure gradually changes as the search progresses, we have a dynamic MAB problem (D-MABP). Da Costa et al. [11] apply a hybrid of the UCB1 algorithm (Section 3.2) to D-MABP, which uses a Page-Hinkley test (PH, see Section 3.3.1) to detect abrupt changes in the environment. In our D-MABNS, "environment" refers to the fitness distributions. Algorithm 3.1 presents D-MABNS; a proposal to adopt the concept of D-MABP solvers for neighbourhood search problems.

3.3.1 Environment Change detection

We use the Page-Hinkley (PH) statistics [27,22], as proposed by Da Costa et al. [11], to detect significant changes in fitness distributions. For a given set of reward observations over time $\{r_{k,1}, ..., r_{k,t}, r_{k,t+1}...\}$, PH detects a reward $r_{k,t+1}$ that does not come from the same statistical distribution as the previous observations.

Algorithm 3.1 D-MABNS (x)

Define a set of neighbourhood structures $N_k, (k = 1, ..., k_{max})$
Assign initial UCB value UCB_k=1 to each N_k
I_k indicates the next solution $x' \in N_k(x)$ will be checked. Initially $I_k = 0, (k = 1, ..., k_{max})$
v_{best} is the best forecasting value among all neighbourhoods. $v_{best} = 0$
while $\exists I_k$ **do**
    i.e. not reached the end of $N_k(x)$
    $k^* = argmax_{k \in \{k=1,...,k_{max}\}} \text{UCB}_k$
    $x' \leftarrow$ the solution indicated by I_{k^*}.
    I_{k^*} move to next
    if $f(x') < f(x)$ **then**
        $x \leftarrow x'$
        Update neighbourhoods (Section 3.3.2)
    end if
    Calculate reward r_{k^*} based on changed fitness (Section 3.2.2)
    if Environment change (Section 3.3.1) **then**
        Reset all values used to calculating UCBs
        $v_{best} = 0$
        $\forall I_k = 0, (k = 1, ..., k_{max})$
    end if
    Update all values used to calculate UCB
    Update UCB values for all neighbourhoods
    If $v_{k^*} > v_{best}$, then $v_{best} = v_{k^*}$
    Pruning(I_{k^*},v_{k^*},v_{best}) (Section 3.3.3)
end while
return x

Formally, we use $\overline{r_k}$ to represent the average value of rewards observed so far for neighbourhood N_k; $\overline{r_k}$ is updated every time a new reward is received. We define $\epsilon_{k,t} = r_k - \overline{r_k} + \theta$ to represent the difference between the reward r_k from the current iteration and the average reward value, where θ is a tolerance parameter that is used to enhance the robustness of the PH test in a slowly changing environment. For simplicity, we set $\theta = 0$. The PH statistic calculates a variable $m_{k,t}$ as the sum of $\epsilon_{k,t_1}, ..., \epsilon_{k,t_{max}}$, and a variable $M_{k,t}$ which is the maximum value of $m_{k,1}, ..., m_{k,t}$.

Algorithm 3.2 is used to update information about the tested neighbourhood and to detect environment change. The parameter λ is a user-defined value that controls the trade-off between false positive and false negative detection errors; like θ, λ controls the sensitivity of the change detection. In Section 4.3.4, we present experimental tests on a set of λ values to analyse the impact of change detection on algorithm performance.

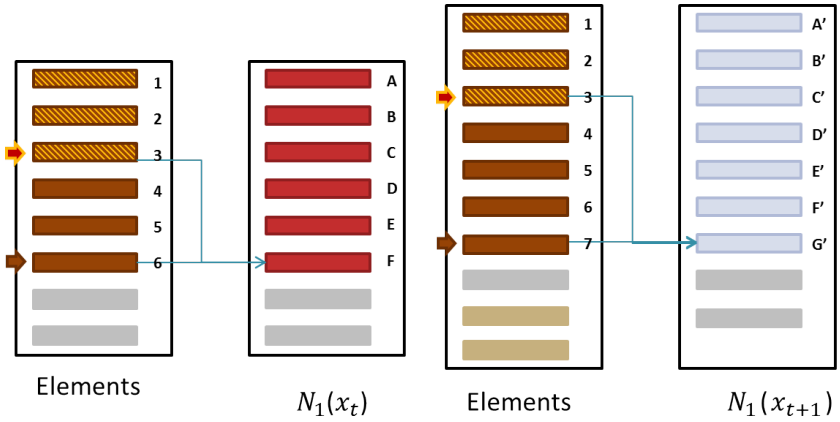
3.3.2 Feature Sequential Search and Neighbourhood Updating

Within the neighbourhood search, D-MABNS applies Feature Sequential Search (FSS). FSS identifies a set of changing elements of a move according to the cost of changing elements. The elements represent some basic units of our solution structure such that each next element of a neighbourhood should be generated in a constant time. For example, for a "2Opt" move, elements might be edges.

Algorithm 3.2 Environment change [11]

1: $\overline{r_k} \Leftarrow \frac{1}{n_k+1}(n_k\overline{r_k} + r_{k+1})$
2: $n_k \Leftarrow n_k + 1$
3: $m_k \Leftarrow m_k + (\bar{r_k} - r_k + \theta)$
4: $M_k = max(M_k, m_k)$
5: **if** $M_k - m_k > \lambda$ **then**
6: return environment is changed.
7: **else**
8: return false
9: **end if**

FSS usually considers a candidate element list sorted intuitively by features (e.g. in "2Opt" moves, length of edges), but sorting is not essential. Figure 4(a) shows construction of a complete move, involving two changing elements labelled 3 and 6, that produces a neighbour solution from N_1, using a nested loop to search through the elements list.



(a) Neighbourhood search in $N_1(x_t)$ at LS iteration t

(b) Neighbourhood search in $N_1(x_{t+1})$ at LS iteration $t+1$

Fig. 4: Neighbourhood Updating

In Figure 4(b), solution x_{t+1} is derived from solution x_t in Figure 4(a). The two solutions share mostly the same structure and many joint neighbours, so we do not need to check the neighbours that have already been checked in the previous LS iteration. In practice, it is also unlikely that a move which was checked in iteration t and did not make an improvement would produce a better solution in iteration $t+1$, even when the move leads to different solutions in iteration $t+1$.

After making a move from x_t to x_{t+1}, where $x_{t+1} \in N_1(x_t)$, we continue to check other elements that still exist in the new solution x_{t+1}, checking the new elements last. The previously checked elements are reconsidered only when

the algorithm detects an change (Algorithm 3.1, line 15). When the pointer I_k (Algorithm 3.1) reaches the end of a neighbourhood $N_k(x_t)$, a large negative value is assigned to the neighbourhood to ensure that it cannot be chosen again until PH signals a change.

3.3.3 Neighbourhood structure pruning

For each neighbourhood structure N_k, the forecasting value v_k is used to keep track of quality. v_k can also be used as an prompt to prune a bad neighbourhood. As shown in Algorithm 3.3, when v_k is set to a large negative value, the neighbourhood structure N_k will not be selected and updated until a change in fitness distribution. This strategy is especially useful in conjunction with FSS, where a promising area of a neighbourhood is explored early in the search. The user-defined parameter γ in Algorithm 3.3 adjusts the tolerance of accepting a neighbourhood that is worse than the best evaluated one. Setting $\gamma = 0$ turns off the pruning function.

Algorithm 3.3 Pruning(I_{k^*},v_{k^*},v_{best})

1: **if** $v_{k^*} < \gamma\ v_{best}$ **then**
2: set I_{k^*} to the end of N_k
3: set v_k to a big negative value
4: **end if**

Section 3 has introduced the general concept of D-MABNS, as well as the techniques that it employs. The following sections apply D-MABNS on two combinatorial optimisation problems, using a real-world scenario and a set of benchmark problems.

4 An application of D-MABNS to maintenance scheduling

The first application of D-MABNS is to a real-world geographical-distributed asset maintenance problem (GDMP) on which we have previously worked [5]; we briefly introduce the problem, and explain how we derive five test instances from our original data set [5].

In our GDMP, a geographically-distributed system has n assets that need sufficient maintenance over period D, given limited resource. Rather than imposing a hard constraint for visit pattern and frequency, the solution decides whether to visit an asset i in the D period. Each asset i is associated with a risk impact r_i that measures the impact of asset failure on the surrounding environment (e.g. as the economic loss incurred). The failure rate of each asset changes over time; a function $P_i(d)$ using time since last maintenance estimates whether an asset i is in a failure state on day d.

The objective is to select a judicious subset of assets from n assets and schedule them to daily maintenance routes within the planning period D, in

order to minimize the risk in this period, expressed as:

$$\sum_{d \in D} \sum_{i=1}^{n} r_i P_i(d) \tag{3}$$

There are three constraints that must be respected by a feasible scheduling plan: (1) a maximum of K routes can be generated each day (one per vehicle); (2) each route of maintenance actions must start and end at the depot; (3) for any route, the maximum duration constraint should be respected.

The original data [6] is for gully pot maintenance in Blackpool, UK. The gully pot maintenance system records 28,294 gullies distributed over approximately 36.1km^2. To test performance of our D-MABNS algorithm, we generate five problem instances of various sizes, by randomly selecting 10%, 25%, 50%, 75% and 100% of the gullies from the system. Each gully pot is associated with location, risk impact r_i, the number of days since its last service, and two parameters for the failure probability estimation function $P_i(d)$ [1]. The planning horizon in each instance is set to 7 days and only one vehicle is available to deliver the service.

4.1 GDMP solution approach

In [5], we use a hyperheuristic to schedule maintenance; we use the same solution framework for the D-MABNS experiments. Here, we briefly outline the process.

1. **Preparation**: generate a candidate route set S_{all} that ignores all risk impact and failure rate information; optimise for distance using a standard CVRP heuristic approach. (Extending [5], we have stored our 309 distance-optimised routes for the 28,294 gullies in a database.)
2. **Initialisation**: generate a schedule for D days by selecting the routes $s \in S_{all}$ with the highest risk, measured by $\sum_{i \in s} r_i P_i(d_1)$, where d_1 represents the first day of D period.
3. **Optimisation** The optimisation steps repeat for a fixed amount of CPU time:

   Improvement: apply a heuristic approach to improve the solution, evaluated by the objective function in Equation 3.

   Re-initialisation: randomly destroy w days' schedule from the output of the improvement stage; rebuild the destroyed routes from scratch by considering assets with the highest risk estimation and few randomly selected asset points; assign these routes randomly to the days.

In [5], we use a tabu-based hyperheuristic (binary exponential back off (BEBO) [28]) to manage the search in the improvement stage. Here, we replace BEBO by D-MABNS.

[1] A Weibull distribution is used to estimate the lifetime of a gully pot. The data set, and further details, can be found online, at `https://www-users.cs.york.ac.uk/~yujiec/`

4.2 Local search moves

We use the same six moves for both D-MABNS and BEBO [5], as follows.

move1 Delete two edges each from two routes and reconstruct to generate two feasible new routes. This move is the same as chain cross exchange; each chain contains a maximum k points, $1 \leq k \leq 5$.
move2 Insert k points that do not appear in the schedule plan, using a cheapest insertion heuristic with a relaxed route duration constraint. If, as a result of the planned insertion, any target route exceeds the duration constraint, repeatedly remove the best-condition point from the route until it becomes feasible; $5 \leq k \leq 20$.
move3 Replace the last n days' schedule with n other routes from the candidate set S_{all}, $1 \leq n \leq D$.
move4 Same as move3, except that we choose the n days' schedule to replace uniformly at random, instead of the last n days' schedule, $1 \leq n \leq D-1$.
move5 Switch two days' schedules.
move6 Move one day's schedule to an earlier day.

Our earlier implementation of BEBO [5] applies a first-improvement heuristic (using lexicographic search) for each move as a low-level heuristic. The search structure of D-MABNS is shown in Figure 2(b), Section 3.1.

4.2.1 Element sorting by features

It is not essential to sort elements, but we do so whenever we have suitable domain information. GDMP is a risk minimisation problem, so many elements can be sorted by their current risk estimation. In our implementation, *move1* uses an edge list that is sorted by the length of edges; *move2* uses an asset point list that is sorted by the current risk estimation of each point. Due to the large number of asset points we have in an instance, a single search loop that selects the next k points is used instead of a nested loop; *move3* and *move4* use a route information list that is sorted by the sum of the current risk of all points in each route. *move5* and *move6* do not sort their elements by any features.

4.3 Computational results for GDMP

This section firstly reports a series of sensitivity analyses on the parameter settings of D-MABNS. We then compare D-MABNS with two algorithms based on the multi-arm bandit. The algorithms tested apply different decision making strategies (probability matching [17] and random selection) allowing comparison to the UCB1 algorithm (Section 3.2). Section 4.5 compares D-MABNS performance to that of two BEBO hyperheuristics introduced in [5] and a modified VNS [7] (denoted as VNSr(R)). All algorithms are implemented in C$^\sharp$ and executed on a cluster composed of 8 Windows computers with 8 core

Intel Xeon E3-1230 CPUs and 16GB RAM. All tested parameter settings and algorithms start with the same initial solution obtained by the *Preparation* and *Initialisation* steps in section 4.1. For each of the problem instances, each algorithm is run 30 times. The stopping condition for all algorithms is the same CPU time, as shown in Table 1.

Table 1: Data set information and CPU time allowed for experiments

ID	number of asset	CPU allowed
mi01	2815	120s
mi02	7037	240s
mi03	14074	900s
mi04	21111	1800s
mi05	28149	3600s

4.3.1 Sensitivity analysis

Finding the best setting of parameter values is a non-trivial and time consuming task, that often requires considerable expertise and experience. Table 2 summaries the parameters used by the D-MABNS algorithm, including the time window parameter W and the scaling parameter a in the exponential reward function (Section 3.2.2), the smoothing factor α (Section 3.2.1), the fitness distribution change λ (Section 3.3.1) and the pruning rate γ (Section 3.3.3). For each parameter in Table 2, we sample values from a given range and run each parameter-set 30 times for the predefined CPU times shown in Table 1. The parameter-set that has the best average performance over all instances is used for comparison with other heuristic methods.

Table 2: Parameter settings

Parameter	Tested range	Value applied
Time window W	$[100, 200, 400, 600, 1000, 3000, 5000]$	400
Scaling parameter a	$[2, 5, 10, 20]$	5
smoothing factor α	$[0.2, 0.4, 0.6, 0.8, 1.0]$	0.8
Environment change λ	$[0.01, 0.05, 0.15, .., 0.3, 0.5, .., 1.0, 2.0, .., 4.0]$	0.05
Pruning rate γ	$[0.01, 0.03, 0.05, 0.1, 0.3, 0.5, 0.7, 0.9]$	0.7

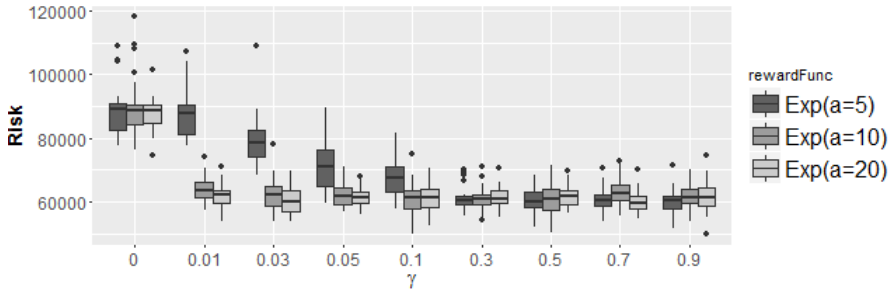
Small values of W, the time window parameter, mean that the algorithm only considers recent fitnesses in neighbourhood evaluation, resulting in a more dynamic reward process over time. Consequently, if recent fitness observations are not good, even a slightly better fitness may result in a big reward. For the smoothing factor α, larger values mean that historic rewards are forgotten more quickly.

Our preliminary tests show a strong impact on performance from the environment change detection parameter λ and the neighbourhood pruning pa-

rameter γ. Some interesting behaviours have also been observed in reward function usage and search within neighbourhoods. Therefore, we design further experiments and explore these parameters in more detail.

4.3.2 Reward function and Neighbourhood pruning

As discussed in Section 3.2.2, the reward function maps the fitness changes to rewards that are used to in the neighbourhood quality evaluations. The reward function, along with the pruning parameter (Section 3.3.3), are essential parts of D-MABNS algorithm design. We test the exponential reward functions with different settings of the neighbourhood pruning parameter γ. The other parameters (Table 2) are fixed as $\{W = 400, \alpha = 0.8, \lambda = 0.05\}$. Each setting runs 30 times.



$$RF = exp(-a(\delta_f - \delta_f^{min})/|\delta_f^{max} - \delta_f^{min}|)$$

Fig. 5: Effect of reward function with different neighbourhood pruning parameter settings γ, illustrated for mi02. We measure the algorithm performance using the objective function of risk minimisation.

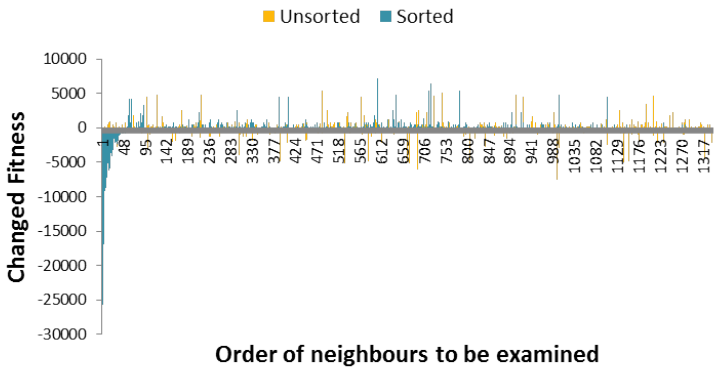
We have run the experiment on all five GDMP instances, and observe similar results. In all plots, we can see that the pruning parameter γ is important to algorithm performance. Figure 5 presents the results for the mi02 instance, as box plots of risk (y axis) against pruning parameter settings (x axis). When $\gamma = 0$, there is no significant difference across all setting of a. However, when γ is increased to values in the range 0.01 to 0.05, bigger a values achieve better results. This is because a bigger a value emphasises the difference between results of improving and worsening moves (Fig. 3, Section 3.2.2). Combining the effect of the pruning mechanism and the scaling factor a for the exponential reward function, bigger a values result in an earlier or more harsh neighbourhood pruning policy, which seems to especially benefit our problem instances. However, this advantage fades as γ increases. The best performing combination here, measured by the mean fitness value, is $\{\gamma = 0.5, a = 5\}$.

Recall that the reward function uses a positive constant a to map the raw fitness change δ_f to the range $[-1, a-1]$, such that when $\delta_f > 0$, $\lim_{a\to\infty} r_k = 0$.

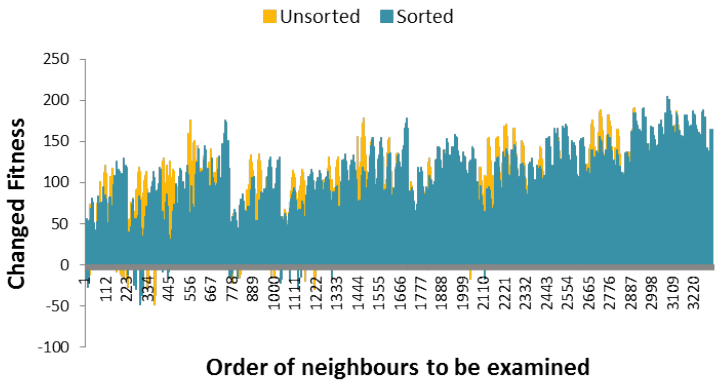
The exponential reward function amplifies small differences in δ_f and needs an extra parameter a to adjust the shape of reward function; this may affect the overall performance of algorithms. Increasing a leads to a smaller reward range of worsening fitnesses ($\delta_f > 0$), which dilutes information from most tested neighbours. We suggest assigning a to a value smaller than 10 to map the worst found fitnesses to values about 10^{-5}.

4.3.3 Neighbourhood sorting and pruning

Our proposed D-MABNS uses FSS (Section 3.3.2) to determine the order in which neighbours are checked within one neighbourhood. To verify the



(a) Neighbourhood structure using *move2*



(b) Neighbourhood structure using *move1*

Fig. 6: A snapshot of two neighbourhood structures using or not using FSS, illustrated for the mi02 instance.

importance of the FSS strategy, we capture a few neighbourhood structures for a current solution x, and plot the δ_f.

Figure 6(a) shows one example structure of the neighbourhood $N_{move2}(x)$. Because the search using *move2* applies a single search loop that picks every next k elements (Section 4.2), only a sub-area of $N_{move2}(x)$ is checked. In this case, the sorted features seem especially helpful in forming improved solutions early in the search stage and pruning the neighbourhood after about the 50th examination looks like it would produce savings in CPU time without missing good moves.

Comparing the search with unsorted and sorted elements for *move1*, the feature sorting strategy shows no significant contribution to the search in $N_{move1}(x)$ (Figure 6(b)). In this case, the pruning strategy may not be beneficial as improvements could be found throughout the neighbourhood. Recall that the elements in *move1* are sorted by the length of edges, whereas our objective function is risk minimisation. This result reveals the importance of problem domain knowledge in FSS design. If the fitness distribution of a neighbourhood shows strong orderliness on one or a few features, FSS significantly improves the search efficiency. In other situations, the sorting process may not be worth the effort.

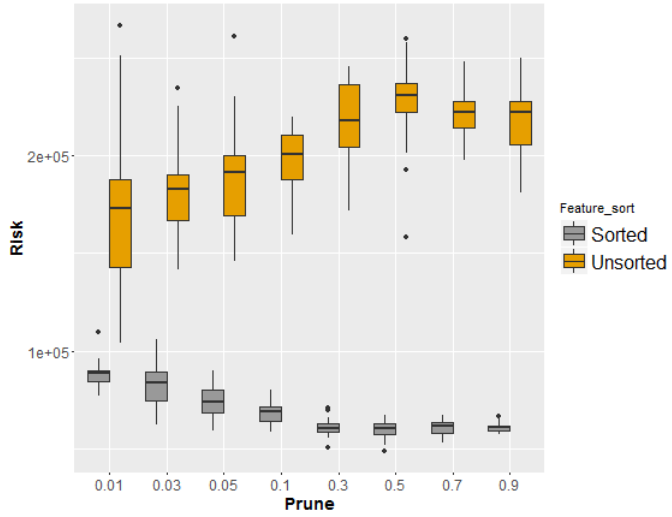


Fig. 7: Impact of sorted features with different pruning parameter γ, on the mi02 instance. Parameter settings: $\{W = 400, RF = exp(a = 5), \alpha = 0.8, \lambda = 0.05\}$

An example of another, more direct, way to illustrate the impact of FSS and pruning is shown in Figure 7. We repeat each parameter setting 30 times with the predefined CPU time and record the solution quality. The same experiment has been tested for our five instances (see Appendix, Figure 17)

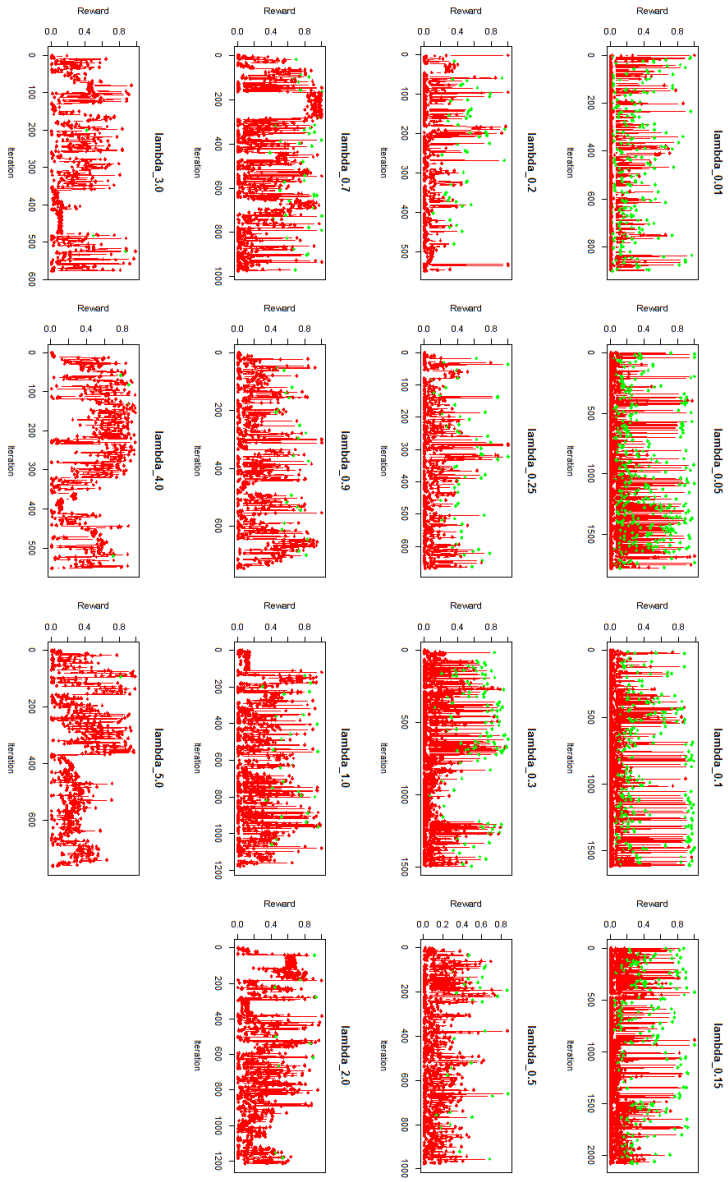


Fig. 8: The effect of parameter λ on environment change detection. The green points indicate a change leading to resetting of arms; the red points represent the normal situation. Other parameter settings are presented in Table 2

and similar effects have been observed across all of them. Overall, the results clearly show the positive impact of FSS on algorithm performance. Feature sorting guides the search to promising moves in the early stages. Combined

with the pruning strategy, FSS significantly improves the search efficiency. On the other hand, when not using feature sorting, pruning too early reduces the chance of finding good neighbours. As we can see from Figure 6, when a neighbourhood is unsorted and its fitness landscape is chaotic, the pruning decision (see Algorithm 3.3) becomes less meaningful.

4.3.4 Environment change detection

D-MABNS resets the evaluation of all of the neighbourhood arms when the PH statistic signals an environment change (Section 3.3.1). The PH statistic is widely used in D-MABP, but no analysis is presented [12,31]. In order to understand the contribution of the PH statistic and the impact of parameter λ, we test different λ values, given other parameters $\{W = 400, RF = exp(a = 5), \alpha = 0.8, \gamma = 0.05\}$. A small γ value is used for this experiment as preliminary tests showed that a large γ diminishes the impact of λ.

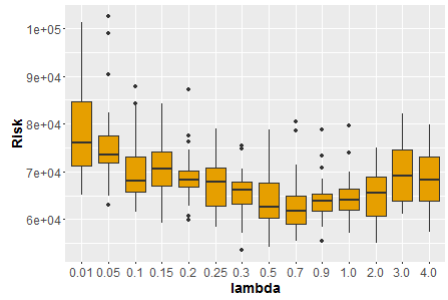
We evaluate the PH statistic via algorithm performance in terms of final solution quality. As before, each parameter setting is repeated 30 times with the predefined CPU time (Table 1). Figure 8 illustrates an example of rewards collected by *move2(*$k = 10$*)* given different λ setting. As λ gets bigger, the PH statistic becomes more tolerant of reward variations. Figure 9(a) shows that D-MABNS achieves worse results when λ is too small, because there is too much noise in the PH alarm signals.

To further analyse the relation between solution quality and environment change detection, we measure the distance between PH signals for environment change and PH for the normal situation. We repeat each parameter setting 5 times for 180 CPU seconds, and each run records the rewards collected over time by each neighbourhood arm. Each data point is labelled *True* or *False* depending on whether it is a changing point or not. We then use a Dunn index [29] to evaluate the decision quality of PH by measuring the distance within and between environment changing and non-changing points. The intuition is that the lower the noise in the detection, the better the solution quality that an algorithm can achieve. Figure 9(b) includes the average value of the Dunn index of each neighbourhood detection result. We can see that as the Dunn index gradually gets bigger, the solution quality in Figure 9(a) gets better. Note that the Dunn index could thus be used to efficiently tune the environment threshold parameter.

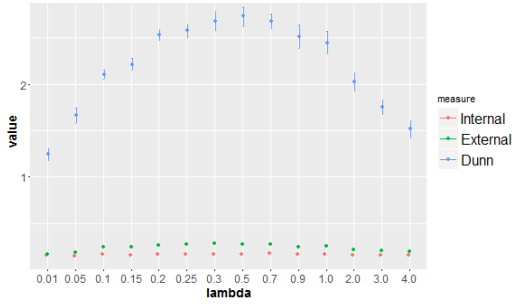
4.3.5 Environment change detection and Neighbourhood pruning

In the previous experiments, we observe the strong positive impact of the pruning parameter γ on algorithm performance for our GDMP instances. As γ increases, the algorithm performance also improves. When γ is bigger than 0.3, sensitivity to other parameters is reduced.

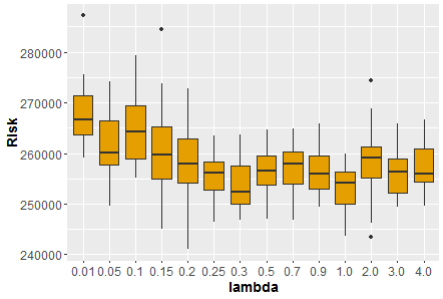
Our experiments on the combined impact of environment detection λ and pruning parameter γ give a rather different view. For example, Figure 10 shows



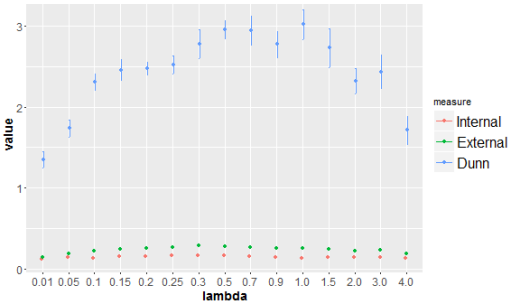
(a) mi02: Performance with different lambda setting



(b) mi02: Cluster validation index



(c) mi04: Performance with different lambda setting



(d) mi04: Cluster validation index

Fig. 9: The effect of parameter λ on algorithm performance. The Dunn index is calculated as follows: $Dunn = \frac{1}{K}\sum_{i=1}^{i=K} Dunn_i$, where K is the number of neighbourhoods; $Dunn_i = \frac{\delta(C_{true}, C_{false})}{max\ \Delta_C}$, where $\delta(C_{true}, C_{false})$ measures the Euclidean distance between the centres of two clusters (denoted as external distance), and $\Delta_C = \frac{1}{|C|}\sum d(s, v(C))$ measures the average distance between all samples $s \in C$ and the cluster's centre $v(C)$ (denoted as internal distance). For more information about cluster validation index see [29].

that, when the environment detection parameter λ is bigger than 0.25, increasing the pruning parameter γ starts to loose its effect. To compare the search strategies, three box plots, A, B, and C, are picked out in Figure 10. Parameter settings at A ($\lambda = 0.01, \gamma = 0.5$) emphasise wide exploration, whilst those at C ($\lambda = 4.0, \gamma = 0.05$) concentrate on exploitation. During exploitation, historical reward information becomes useful to guide the search; smaller λ values give poorer results because the MAB arms are reset too frequently, resulting in loss of historical information. Parameter setting B is the best performing in this experiment, as it achieves a balance between exploration and exploitation. Results for the other GDMP instances are presented in the Appendix, Figure 18.

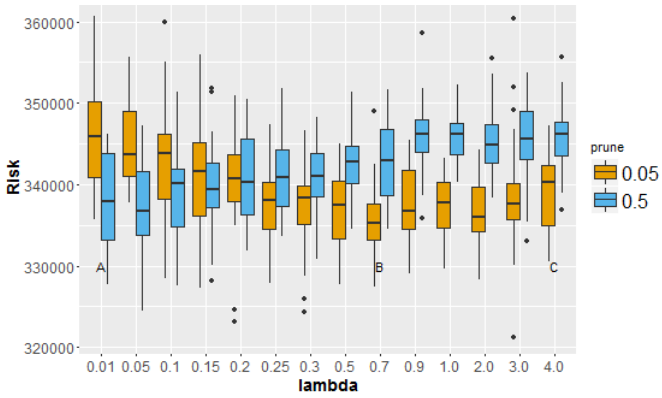


Fig. 10: The effect of parameter λ on the algorithm performance with two different pruning parameter setting γ. The other parameter settings are $\{W = 400, RF = exp(a = 5), \alpha = 0.8\}$. Tested on mi05.

4.4 Other search strategies between neighbourhoods

Apart from the UCB1 algorithm, many other decision making strategies are proposed in literature (Section 2.2). These methods introduce strategies from different perspectives to tackle the exploration and exploitation dilemma of MABPs. In this section, we compare the UCB1 method to probability matching (PM) [17,33] and a simple random (Ran) selection strategy.

4.4.1 Probability matching

Compared to UCB1, PM implements exploration by adding uncertainty to the selection process instead of using a statistical equation. At each decision point, the probability p of examining a neighbourhood is proportional to its forecast quality. The benefit is obvious: there is no need to rescale fitness to tune the statistic.

Our implementation of PM follows the pseudo code provided by [33]. We use an exponential reward function, shown in Function 4, for which no scaling factor a is needed. Other parameters for PM are: $\{p_{min} = 0.01, \alpha = 0.8, \gamma = 0.94\}$, where p_{min} is the minimal probability value to ensure that none of the arms is ignored; α and γ are the same parameters as for the UCB1 algorithm, with values selected through preliminary experiments.

$$RF = exp(-\delta_f / f(x)) \tag{4}$$

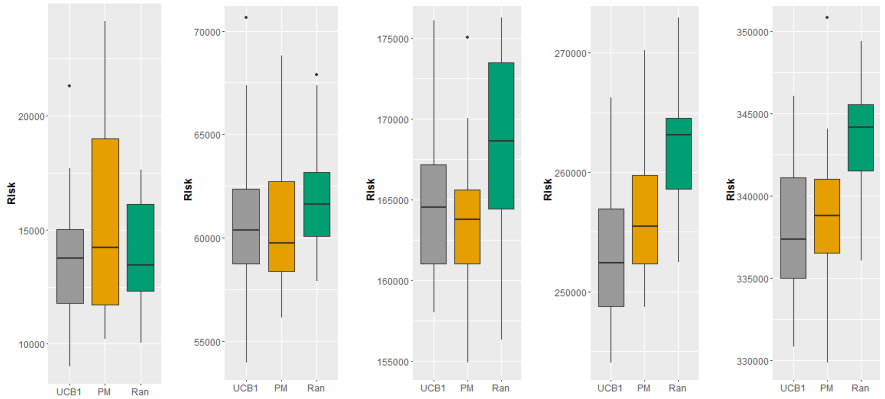
4.4.2 Random selection

Ran randomly selects a neighbourhood N_k from the available candidate set following a uniform distribution. Compared to UCB1 and PM, Ran does not

need any statistical technique to evaluate arms. To apply a neighbourhood pruning strategy, we simply cut off a neighbourhood if no improvement has been found through the last m tries of this neighbourhood. We use $m = 50$, as the value that produces the best average results over our five GDMP instances.

4.4.3 Comparing the UCB1, PM and Ran strategies

We compare UCB1 (parameter settings in Table 2), PM and Ran. Each algorithm repeats 30 times and the results are shown in Figure 11. For small problem instances (instance mi01, mi02), there is no obvious difference between the three tested methods. However, for larger problem instances, the performance of Ran is much worse than the other two methods. This suggests that there is some useful information in the fitness distribution and landscape and that we can statistically capture this information, via the reward function, to guide the search in the correct direction.



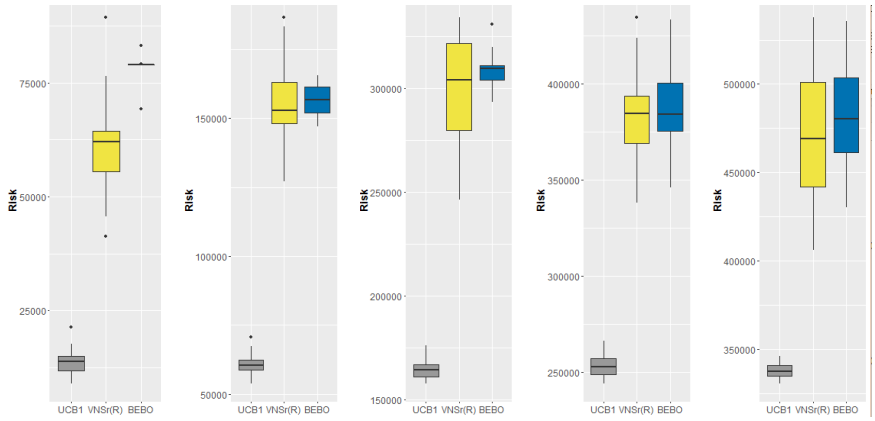
(a) mi01 (b) mi02 (c) mi03 (d) mi04 (e) mi05

Fig. 11: Performance comparison of different decision making strategy: UCB1, Probability matching (PM) and random selection (Ran)

4.5 Comparison to traditional hyperheuristic

In this section, we compare D-MABNS with two hyperheuristic algorithms, BEBO [28] and the VNSr(R) variant of VNS [20]. The implementation details have been published in [5,7], respectively. As Figure 12 shows, D-MABNS improves the solution quality for all tested instances.

Comparing the architecture of the three algorithms, D-MABNS (Figure 2) is more like a breadth first search. Whereas BEBO and VNSr(R) use first improvement low-level heuristics to repeatedly examine one type of move until an improvement is found, D-MABNS biases the neighbourhood search in



(a) mi01 (b) mi02 (c) mi03 (d) mi04 (e) mi05

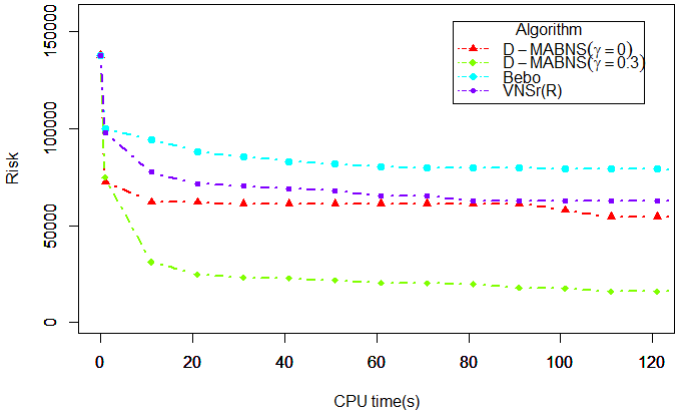
Fig. 12: Performance of D-MABNS (labelled as UCB1), VNSr(R) and BEBO

a promising direction, which tends to improve the algorithm efficiency. At a higher level, compared to VNSr(R), D-MABNS builds a descent search path with a more flexible combination of moves. Furthermore, D-MABNS introduces many tricks, such as dynamic neighbourhood updating to avoid repeatedly checking the same elements, and pruning to discard unpromising areas of the neighbourhood.

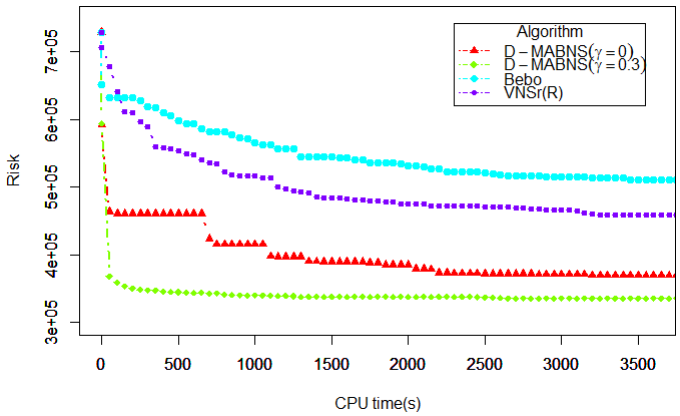
Figure 13 plots an example of the risk value (minimisation objective) changing over time using BEBO, VNSr(R), D-MABNS without pruning ($\gamma = 0$) and D-MABNS with pruning ($\gamma = 0.3$). Pruning reduces unnecessary search space, allowing the algorithm to converge earlier. Even without pruning, the D-MABNS algorithm has the advantage over the two hyperheuristics, especially for larger problem instances. We attribute the achievement to the D-MABNS algorithm's breadth first style of search and dynamic neighbourhood updating.

5 Applying D-MABNS to PVRP

So far, we have tested the D-MABNS on five instances of GDMP and shown that it significantly out-performs other hyperheuristic approaches. To test our algorithms on other combinatorial optimisation problems, we choose the Periodic vehicle routing problem (PVRP) ([8,4,9,19]), a widely-used standard model for periodic maintenance and on-site service problems. Compared to GDMP, PVRP has tighter constraints on service pattern requirements. Consequently, the search space of feasible solutions for visiting pattern assignment is smaller. In this section, we test the D-MABNS on 42 benchmark PVRP



(a) mi01



(b) mi05

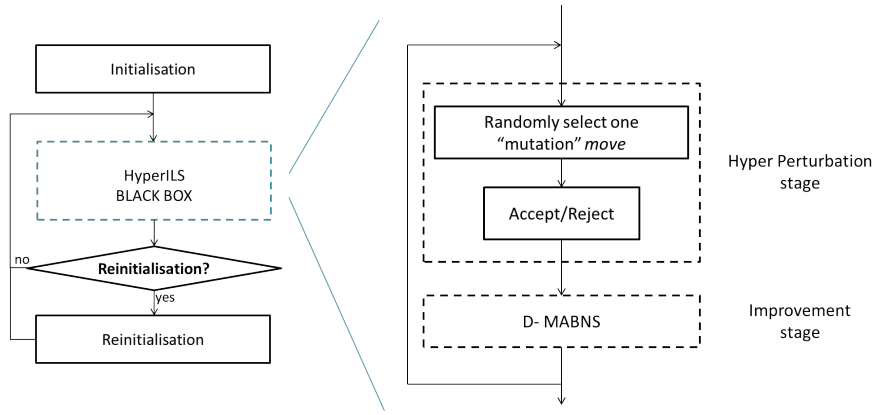
Fig. 13: Comparing single runs of D-MABNS, VNSr(R) and BEBO.

instances, allowing comparison to a wide range of heuristic approaches. Background information on the benchmark instances can be found in [21][2].

5.1 Using D-MABNS as an improvement heuristic in a hyperheuristic framework

To evaluate D-MABNS performance on PVRP benchmark problems, we embed the D-MABNS algorithm in the improvement stage of the HyperILS framework [25,2], as shown in Figure 14. The work presented here extends [7], and uses the same (re)initialisation and mutation moves in the hyper-perturbation stage.

[2] The data for the benchmark instances is available at http://neo.lcc.uma.es/vrp/vrp-instances/periodic-vrp-instances/



(a) PVRP solution (b) HyperILS

Fig. 14: Algorithm framework

Table 3: Local search moves for PVRP. Details are given in [7].

Type	**Moves**	**Element sorting (FSS)**
Route related	2Opt, 3Opt, 2PS, Relocate, Cross	Edges sorted by length
Pattern related	Customer pattern reassign, Two customer pattern swap	Customers sorted by adjacent edge lengths
Mixed	Relocate with pattern, Cross with pattern	Edges sorted by length

To build the neighbourhoods for D-MABNS, we consider three types of moves: route modification, customer service pattern modification and mixed operators [7]. Unlike the first improvement low-level heuristics employed by VNSr(R), all local moves are managed by D-MABNS. FSS is used, as described in Table 3.

5.2 Computational results for PVRP

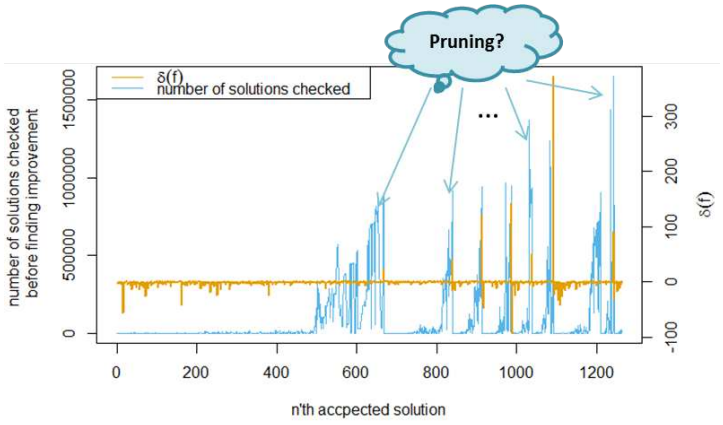
In this section, we discuss the behaviour of D-MABNS on PVRP, and compare results to D-MABNS on GDMP. In addition, we compare D-MABNS with state-of-the-art meta-heuristics that are specifically designed for PVRP.

5.2.1 Sensitivity analysis

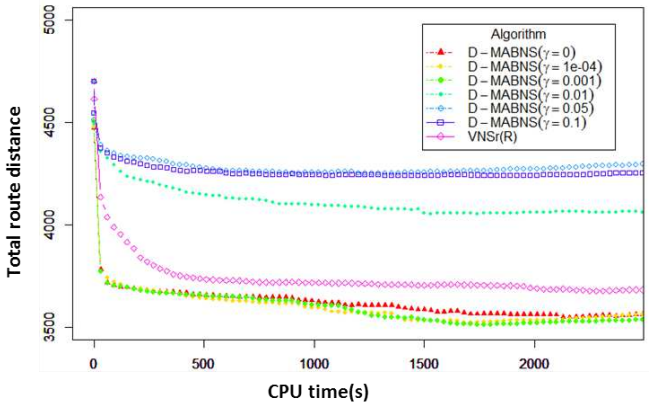
We conduct parameter sensitivity experiments, similar to those in Section 4.3.1; the best performing parameter settings for D-MABNS are used for performance comparison with other methods.

Compared to solving GDMP, neighbourhood pruning and sorting exhibit some different behaviours. Figure 15(a) shows an example tested on PVRP instance *p13*, the largest benchmark containing 417 customers, of recording

the number of solutions that D-MABNS ($\gamma = 0$) examines before it moves to the next solution. Every interval between two worsening solution (shown as positive $\delta(f)$) is a local search process. We can see that, at a late stage of each local search, the cost to find an improvement increases significantly. When is the optimal time to prune the search within a neighbourhood? Tested on *p13* (Fig. 15(b)), we can see the algorithm performance decreases significantly when the pruning parameter γ is bigger than 0.01. Because of early pruning, the algorithm cannot reach local optima and the final solution is of poor quality.



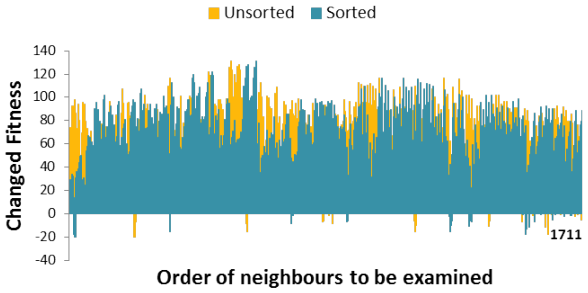
(a)



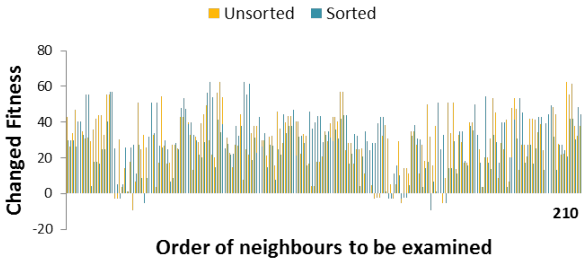
(b) D-MABNS with different γ

Fig. 15: Impact of neighbourhood pruning on PVRP benchmark *p13*. Graph (a) records the changed fitness $\delta(f)$, and the number of solution examined before accepting the next solution, for D-MABNS ($\gamma = 0$).

Similarly, sorting strategies do not give any obvious advantage in the PVRP solver. In contrast to GDMP, the PVRP fitness landscape does not show ob-



(a) Relocated neighbourhood



(b) 3Opt neighbourhood

Fig. 16: PVRP sorting effects: examples of current-solution neighbourhoods

vious orderliness on the features we tried. Figure 16 illustrates two examples, from "relocate" and "3Opt" neighbourhoods.

Our experiments raise an important question: when is the appropriate moment to stop the local search and restart the journey somewhere else in the solution space? From our experience, we summarise guideline rules as follows.

1. For small problem instances, such as the PVRP benchmarks (between about 50 and 200 customers), the total time needed for each local search procedure is short. In this situation, no pruning strategy is needed.
2. In any problem where a sorting strategy does not help to guide the local search within neighbourhood structures, a pruning strategy is also unhelpful.
3. When the solution space is very big but a sorting strategy fails, memory techniques plus adaptive pruning rate can be applied to control exploration and exploitation in early and later stages of the search process. The memory technique needs to record potentially-promising areas of the solution space that have been cut off in the early search.
4. When the solution space is big and the fitness distribution of many neighbourhood structures shows strong orderliness on some features, neighbourhood sorting and pruning bring big benefits to the search process.

5.3 Comparing MAB methods to other meta-heuristics

We compare our results with state-of-the-art hybrid-meta-heuristics [35,10] and the best results using a hyperheuristic (VNSr(R)) achieved in our previous work [7]. We use MAB with the simple random selection (Ran) and D-MABNS with parameter settings $\{W = 400, RF = exp(C = 5), \alpha = 0.8, \lambda = 0.7, \gamma = 0.001\}$. The experiments are designed to replicate benchmark conditions from [35]. In particular, the search is always terminated after a fixed amount of CPU time [35]. The results are presented in the Table 4; the best-known value for each problem instance is shown in bold.

Table 4: Performance on PVRP benchmarks, hybrid-GA (VCGLR)[35], parallel tabu search (CM) [10]. The first column shows the number of customers in each instance.

		VCGLR	CM	VNSr(R)		Ran		D-MABNS		
				mean	best	mean	best	mean	best	CPU(s)
	ID	10 run	10 run	10 run		10 run		10 run		
51	p01	**524.6**	**524.6**	**524.6**	**524.6**	**524.6**	**524.6**	524.7	524.6	13.2
50	p02	**1322.9**	1328.7	1333.2	1325.1	1336.1	1324.2	1336.2	**1322.9**	26.4
50	p03	**524.6**	**524.6**	**524.6**	**524.6**	524.7	**524.6**	524.7	**524.6**	10.8
75	p04	836.6	836	841.2	835.8	843.5	836.8	842	**835.3**	63
75	p05	**2033.7**	2034.7	2077	2062.6	2068	2048.5	2069	2047	136.2
75	p06	842.5	**836.4**	851.7	841.8	861.7	845.1	856.3	843.4	53.4
100	p07	827	826.8	829.2	**826.1**	831.2	827.8	831.9	827.5	52.8
100	p08	**2022.9**	2044.3	2069.2	2053.7	2072.3	2054.1	2075.1	2050.2	152.4
100	p09	826.9	826.6	831	827.4	833.3	829.5	833.8	**826.1**	60.6
100	p10	1605.2	**1600.9**	1637.4	1617.5	1651.1	1634.2	1649.1	1618.6	108
139	p11	**775.8**	780.6	792.2	785.2	792.8	786.9	795.4	785.7	276
163	p12	**1195.3**	1196.8	1249.8	1220.3	1243.6	1217.9	1246.4	1218	320.4
417	p13	3599.9	**3518.7**	3662.8	3585.6	3581.2	3548.7	3607.6	3541.8	2400
20	p14	**954.8**	**954.8**	**954.8**	**954.8**	**954.8**	**954.8**	**954.8**	**954.8**	4.8
38	p15	**1862.6**	**1862.6**	**1862.6**	**1862.6**	**1862.6**	**1862.6**	**1862.6**	**1862.6**	10.2
56	p16	**2875.2**	**2875.2**	**2875.2**	**2875.2**	**2875.2**	**2875.2**	**2875.2**	**2875.2**	19.2
40	p17	1597.8	1597.8	1621.8	**1597.7**	1627	**1597.7**	1620.8	**1597.7**	16.2
76	p18	**3131.1**	3155.2	3159	3152.4	3157.1	3150.2	3158.4	3151.6	53.4
112	p19	**4834.5**	4843	4846.5	4846.5	4846.5	4846.5	4846.5	4846.5	135.6
184	p20	**8367.4**	**8367.4**	**8367.4**	**8367.4**	**8367.4**	**8367.4**	**8367.4**	**8367.4**	240.6
60	p21	**2170.6**	2184.1	2187.6	2183.5	2184.4	2182.6	2189.4	2182.6	54
114	p22	**4194.2**	4213.7	4282.3	4229	4288.6	4235.6	4279.2	4232.5	256.2
168	p23	**6434.1**	6575.5	6674.8	6609.2	6633.5	6589	6704.5	6572.7	257.4
51	p24	**3687.5**	3695.2	3731	3693.5	3725.6	3693.5	3734	3693.5	19.2
51	p25	**3777.2**	3780.2	3785.6	3781.4	3781.4	3781.4	3783.6	3781.4	35.4
51	p26	**3795.3**	**3795.3**	3831.8	**3795.3**	3834	3834	3833.5	3815.3	19.8
102	p27	21885.7	**21877.5**	22293	22196.3	22136.7	22071.4	22158.7	22057.7	211.2
102	p28	22272.6	**22271.4**	22559	22444.7	22455.8	22360.7	22471.4	22391.4	280.2
102	p29	**22564.1**	22586.6	23103	22775.2	22749.8	22698.8	22825.4	22663.6	231.6
153	p30	**74534.4**	74547.5	77525.9	76776.7	75903.5	75215.7	76187.1	75421	599.4
153	p31	**76686.7**	76883.2	78650.4	78128.5	77663	77346.3	77916.9	77373.2	600
153	p32	**78168.8**	78366.2	80832	79647.3	79404.6	78907.4	79652.9	78927.3	600
48	pr01	**2209**	2209.9	**2209**	**2209**	2210.6	**2209**	2211.7	**2209**	17.4
72	pr02	**3768.9**	3779.2	3842.2	3822	3849.8	3825.6	3851.7	3817.6	149.4
96	pr03	**5174.8**	5206.6	5303.3	5254.7	5324.3	5255.6	5342.3	5259.1	439.2
144	pr04	**5936.2**	5947.9	6109.3	6056.6	6132.9	6065.7	6150	6070.2	600
144	pr05	**6651.8**	6664.8	6873.1	6810.2	6873.4	6826.3	6885	6792.8	1200
192	pr06	**8284.**9	8316.4	8619.2	8569.9	8667.6	8560.9	8735.3	8597.8	1200
216	pr07	**4996.1**	**4996.1**	5013.8	5000.6	5022.3	5012.3	5024.2	5000.3	89.4
240	pr08	**7035.5**	7041.1	7240.6	7193.6	7246.8	7190.7	7276.6	7174.7	600
288	pr09	**10162.2**	10174	10527.4	10397	10541.4	10484.7	10578.7	10445.4	1200
	pr10	**13091**	13105.9							
summary										**average CPU(s)**
<100	gap	0.00062	0.001999	0.009529	0.004327	0.011071	0.005371	0.011078	0.004115	68.86
>100	gap	0.00118	0.002645	0.026084	0.016554	0.020872	0.013821	0.023947	0.013163	594.63
All	gap	0.00089	0.002307	0.017201	0.009993	0.015613	0.009287	0.017041	0.008308	312.51

The two MAB-based algorithms achieve competitive results. On average, D-MABNS produces routes that are 1.7% longer than the best-known solu-

tions. D-MABNS has found 10 best solutions out of 42 tested instances and a new best solution for instance *p04*; the routes are in the Appendix, Table 5.

Comparing the average results of D-MABNS with *Ran*, in 29 out of 41 tested instances, *Ran* outperforms D-MABNS. Further analysis suggests that *Ran* reaches more different local optima than D-MABNS within given CPU times. Consequently, it increases the chance of finding better solutions. For GDMP, we found that the advantage of the pure random strategy was lost for larger solution spaces (Section 4.4), and we might predict a similar result for larger PVRP.

6 Conclusion

In this paper, we introduce a D-MABNS algorithm for a scheduling and routing problem with geographically-distributed assets maintenance (GDMP). The original problem concerns the scheduling of real-world drainage system maintenance and contains a large number of assets [6]. In order to solve this problem more effectively, the D-MABNS utilises the orderliness property of the neighbourhood structures and applies techniques that focus the search in promising areas of the solution space, such as dynamic neighbourhood updating, feature sequential search and neighbourhood pruning. We perform a comprehensive sensitivity analysis and gain insights into the relationship between FSS and neighbourhood pruning, showing pruning is reliant on good sorting to gain big advantages. Compared to the two heuristic approaches that we developed in previous work (BEBO and VNSr(R) [7]), D-MABNS achieves significant better results in all tested instances.

We then test our algorithm on 42 PVRP benchmark instances to compare its performance with other heuristic approaches found in recent literature. Unlike GDMP, the PVRP neighbourhood structures show no orderliness on the features that have been tested, which reduces the impact of D-MABNS's tricks during the search. However, D-MABNS still achieves very competitive results. On average, D-MABNS achieves solutions within 1.7% of the best-known solutions. This reinforces its success at solving the larger and more challenging instances of GDMP.

References

1. Peter Auer, Nicolo Cesa-Bianchi, and Paul Fischer. Finite-time analysis of the multi-armed bandit problem. *Machine Learning*, 47(2-3):235–256, 2002.
2. E Burke, T Curtois, M Hyde, G Kendall, G Ochoa, S Petrovic, J A Vazquez-Rodriguez, M Gendreau, and Ieee. Iterated Local Search vs. Hyper-heuristics: Towards General-Purpose Search Algorithms. *Ieee Congress on Evolutionary Computation*, 2010.
3. E. K. Burke, G. Kendall, and E. Soubeiga. A Tabu-Search Hyperheuristic for Timetabling and Rostering. *Journal of Heuristics*, 9(6):451–470, 2003.
4. I-Ming Chao, Bruce L. Golden, and Edward Wasil. An improved heuristic for the period vehicle routing problem. *Networks*, 26(6):25–44, 1995.

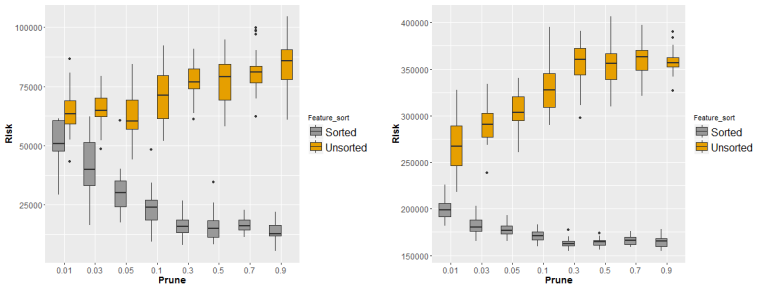
5. Yujie Chen, Peter Cowling, Fiona Polack, Stephen Remde, and Philip Mourdjis. Dynamic optimization of preventative and corrective maintenance schedules for a large scale urban drainage system. *European journal of operational research*, 2016.
6. Yujie Chen, Peter Cowling, and Stephen Remde. Dynamic Period Routing for a Complex Real-World System : A Case Study in Storm Drain Maintenance. In *Evolutionary Computation in Combinatorial Optimisation*, pages 109–120, 2014.
7. Yujie Chen, Philip Mourdjis, Fiona Polack, Peter Cowling, and Stephen Remde. Evaluating Hyperheuristics and Local Search Operators for Periodic Routing Problems. In *Evolutionary Computation in Combinatorial Optimisation*, pages 104–120, 2016.
8. N. Christofides and J. E. Beasley. The period routing problem. *Networks*, 14(2):237–256, 1984.
9. Jean-Francois Cordeau, Michel Gendreau, and Gilbert Laporte. A tabu search heuristic for periodic and multi-depot vehicle routing problems. *Networks*, 30(2):105–119, sep 1997.
10. Jean-François Cordeau and Mirko Maischberger. A parallel iterated tabu search heuristic for vehicle routing problems. *Computers & Operations Research*, 39(9):2033–2050, sep 2012.
11. Luis Da Costa, Alvaro Fialho, Marc Schoenauer, and Michele Sebag. Adaptive Operator Selection with Dynamic Multi-Armed Bandits. *Proceedings of the 10th annual conference on Genetic and evolutionary computation GECCO 08*, page 913, 2008.
12. Álvaro Fialho, Luis Da Costa, Marc Schoenauer, and Michèle Sebag. Dynamic multi-armed bandits and extreme value-based rewards for adaptive operator selection in evolutionary algorithms. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 5851 LNCS:176–190, 2009.
13. Álvaro Fialho, Luis da Costa, Marc Schoenauer, and Michèle Sebag. Analyzing bandit-based adaptive operator selection mechanisms. *Annals of Mathematics and Artificial Intelligence*, 60(1):25–64, 2010.
14. Álvaro Fialho, Marc Schoenauer, and Michèle Sebag. Toward comparison-based adaptive operator selection. *Proceedings of the 12th annual conference on Genetic and evolutionary computation - GECCO '10*, page 767, 2010.
15. Birger Funke, Tore Grunert, and Stefan Irnich. Local search for vehicle routing and scheduling problems: Review and conceptual integration. *Journal of Heuristics*, 11:267–306, 2005.
16. Everette S Gardner. Exponential smoothing: The state of the art, 1985.
17. David E. Goldberg. Probability Matching, the Magnitude of Reinforcement, and Classifier System Bidding. *Machine Learning*, 5(4):407–425, 1990.
18. Chris Groër, Bruce Golden, and Edward Wasil. A library of local search heuristics for the vehicle routing problem. *Mathematical Programming Computation*, 2(2):79–101, 2010.
19. Damon Gulczynski, Bruce Golden, and Edward Wasil. The period vehicle routing problem: New heuristics and real-world variants. *Transportation Research Part E: Logistics and Transportation Review*, 47(5):648–668, sep 2011.
20. Pierre Hansen, Nenad Mladenović, and José A. Moreno Pérez. Variable neighbourhood search: Methods and applications. *Annals of Operations Research*, 175(1):367–407, 2010.
21. Vera C. Hemmelmayr, Karl F. Doerner, and Richard F. Hartl. A variable neighborhood search heuristic for periodic routing problems. *European Journal of Operational Research*, 195(3):791–802, jun 2009.
22. D. v. Hinkley. Inference about the change-point from cumulative sum tests. *Biometrika*, 58(3):509–523, 1971.
23. S Lin and BW Kernighan. An effective heuristic algorithm for the traveling-salesman problem. *Operations research*, 1973.
24. Alexander Nareyek. Choosing search heuristics by non-stationary reinforcement learning. *Metaheuristics: Computer decision-making*, 2004.
25. Gabriela Ochoa and Edmund K Burke. HyperILS : An Effective Iterated Local Search Hyper-heuristic for Combinatorial Optimisation. *International Conference of the Practice and Theory of Automated Timetabling*, (August):26–29, 2014.

26. Gabriela Ochoa, Matthew Hyde, Tim Curtois, Jose A. Vazquez-Rodriguez, James Walker, Michel Gendreau, Graham Kendall, Barry McCollum, Andrew J. Parkes, Sanja Petrovic, and Edmund K. Burke. HyFlex: A benchmark framework for cross-domain heuristic search. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 7245 LNCS:136–147, 2012.
27. E S. Page. Continuous inspection schemes. *Biometrika*, 41(1):100–115, 1954.
28. Stephen Remde, Keshav Dahal, Peter Cowling, and Nic Colledge. Binary exponential back off for tabu tenure in hyperheuristics. *Evolutionary Computation in Combinatorial Optimization*, pages 109–120, 2009.
29. Eréndira Rendón, Itzel Abundez, Alejandra Arizmendi, and Elvia M. Quiroz. Internal versus External cluster validation indexes. *International Journal of Computers and Communications*, 5(1):27–34, 2011.
30. Stefan Ropke and David Pisinger. An Adaptive Large Neighborhood Search Heuristic for the Pickup and Delivery Problem with Time Windows. *Transportation Science*, 40(4):455–472, 2005.
31. N R Sabar, M Ayob, G Kendall, and R Qu. A Dynamic Multiarmed Bandit-Gene Expression Programming Hyper-Heuristic for Combinatorial Optimization Problems. *Cybernetics, IEEE Transactions on*, PP(99):1, 2014.
32. Jorge A. Soria-Alcaraz, Gabriela Ochoa, Jerry Swan, Martin Carpio, Hector Puga, and Edmund K. Burke. Effective learning hyper-heuristics for the course timetabling problem. *European Journal of Operational Research*, 238(1):77–86, 2014.
33. Dirk Thierens. An adaptive pursuit strategy for allocating operator probabilities. *Belgian/Netherlands Artificial Intelligence Conference*, pages 385–386, 2005.
34. Paolo Toth and Daniele Vigo. The Granular Tabu Search and Its Application to the Vehicle-Routing Problem. *INFORMS Journal on Computing*, 15(4):333–346, 2003.
35. Thibaut Vidal, Teodor Gabriel Crainic, Michel Gendreau, Nadia Lahrichi, and Walter Rei. A hybrid genetic algorithm for multidepot and periodic vehicle routing problems. *Operations Research*, 60(3):611–624, 2012.
36. Christos Voudouris, Edward P K Tsang, and Abdullah Alsheddy. Guided Local Search. *Handbook of Metaheuristics*, 146:321–361, 2010.

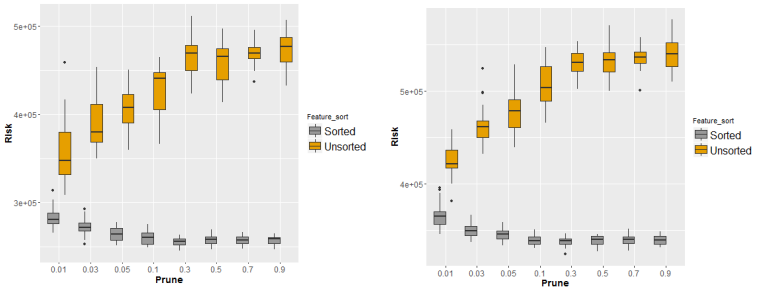
Appendix

Table 5: Best found solutions

p04 (total length of routes: 835.3)
day1:
Route0: 0,67,46,34,4,75,0,
Route1: 0,6,33,63,23,56,24,49,16,0,
Route2: 0,27,37,20,70,60,71,69,36,47,48,0,
Route3: 0,62,22,64,42,41,43,1,73,51,0,
Route4: 0,30,74,21,61,28,2,68,0,
day2:
Route5: 0,17,40,9,39,12,26,0,
Route6: 0,38,65,66,59,14,7,0,
Route7: 0,52,19,54,13,57,15,5,29,45,0,
Route8: 0,8,35,53,11,10,58,0,
Route9: 0,3,44,32,50,18,55,25,31,72,0,

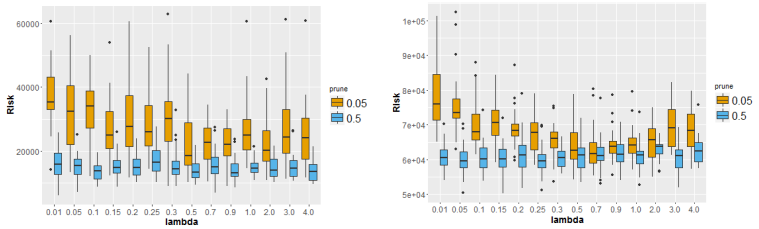


(a) mi01: 2815 vertices (b) mi03: 14074 vertices

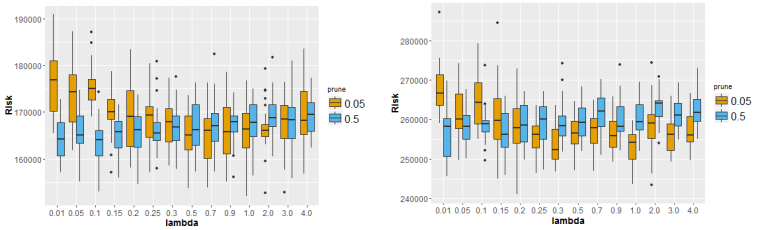


(c) mi04: 21111 vertices (d) mi05: 28149 vertices

Fig. 17: Effect of feature sorting strategy with different neighbourhoods prune rate γ to different sizes of GDMP instances



(a) mi01: 2815 vertices (b) mi02: 7037 vertices



(c) mi03: 14074 vertices (d) mi04: 21111 vertices

Fig. 18: The effect of parameter λ on the algorithm performance with two different pruning parameter setting. The other parameter settings are $\{W = 400, RF = exp(a = 5), \alpha = 0.8\}$. Tested on different sizes of GDMP instances