A Reinforcement Learning Hyper-Heuristic for the Capacitated Team Orienteering Problem

Michael Dontas[a]

[a]*Department of Management Science and Technology, School of Business, Athens University of Economics and Business, Athens, Greece*

Abstract

This work proposes a novel Deep Reinforcement Learning Hyper Heuristic algorithm for the Capacitated Team Orienteering Problem (CTOP). The CTOP generalizes the Orienteering Problem (OP) by considering a fleet of vehicles where every one has a maximum loading capacity. The profit associated with a certain customer is collected if it is visited and its demand can be covered from the current load capacity. The objective is to collect the greatest amount of profit while respecting an upper route duration constraint. We propose a hyper heuristic algorithm based on Deep Q Learning. An RL agent is trained by exploring the solution space to determine the optimal selection of an action on a given solution with a view of maximizing the objective increase after a large number of iterations. The available actions for every solution-state constitute classical low-level heuristics such as customer insertion or removal from the solution. The suggested framework is also equipped with two powerful heuristic methods capable of more intensely exploring the solution space. Their purpose is to guide the agent towards different and more promising regions when it gets stuck in local maximums. The algorithm is compared against previously published CTOP algorithms and analytical results are presented. The novel approach manages to offer a marginal improvement on the test instances that involve a larger number of vehicles in comparison with the rest of the available ones.

Keywords: Transportation, Routing, Team Orienteering, Hyper-Heuristics, Deep Q learning

Contents

1. Introduction

1.1. CTOP and variants

This project tackles the Capacitated Team Orienteering Problem, a multi-vehicle routing problem with the following traits: it aims at maximizing the profit collected from customers that are associated with a specific demand, service time, and profit value. In order for a vehicle to be able to serve a customer and collect the available profit, two conditions are required: a) the additional load discharged to the customer must not surpass the vehicle's capacity and b) the additional visit and service time must not exceed a specified time limit.

Routing problems with profits are examined in the literature as orienteering problems where the goal is to maximize a profit-related objective function under a set of constraints. The name was inspired by the orienteering sport, a game played normally in mountainous or forested areas, where players are called to collect a total score visiting marked places linked with a score under a certain time limit. The term was firstly coined by Tsiligirides (1984) who presented the Orienteering Problem (OP), a problem of creating a single vehicle route choosing among the available customers associated with profits and respecting a predefined time constraint. The Orienteering Problem has received a lot of attention and various approaches and methodologies have been proposed. Sevkli, Sevilgen (2010) implement a Strengthened Particle Swarm Optimization (StPSO) algorithm with the use of Local Search to facilitate solution space exploration and avoid premature convergence. Particle Swarm Optimization (Eberhart, Kennedy 1995) is a population based method where each solution-particle is comprised of a position vector, representing its current point in the solution space and a velocity vector, expressing the direction towards which the particle is about to move. The goal is to perform a series of operations on the population in order to alter each particle's position and velocity vector and guide the swarm towards the most promising solutions. Liang et al. (2013) suggest a Multi-Level Variable Neighborhood Search algorithm, where, in contrast with the conventional Variable Neighborhood Search (VNS) approach, multiple solutions are explored simultaneously and are updated according to the outcome of each solution's neighborhood search. This allows for a faster and more effective exploration, since the solutions of highest quality in each iteration are used to update the instances and thus quicker convergence is achieved.

An interesting variant of the OP is the Orienteering Problem with Stochastic Travel and Service Times (OPSTS), initially proposed by Campbell et al. (2011). This problem simulates the notion that in practice the time needed to visit and serve a customer cannot be accurately predicted and thus should not be considered deterministic. If customers are not reached before the specified deadline, a penalty is imposed. In that sense, the objective of the OPSTS is to maximize the expected collected profit, counting for the expected amount of penalty imposed. Campbell et al. (2011) propose a VNS framework that makes use of a Local Search with a shaking step, along with a dynamic programming approach that manages to produce competitive results, though at a larger computational time. Papapanagiotou, Montemanni (2014) apply Monte Carlo sampling in order to improve the accuracy of the objective function evaluation and use an analytical method to solve the problem.

The Team Orienteering Problem (TOP), introduced by Chao et al. (1996), can be considered as an extension of the OP with multiple vehicles. Among the most prominent implementations for this problem due to their dominating performance stand the works of Dang et al. (2013) and Keshtkaran et al. (2015). The former proposes a Branch-and-Cut algorithm based on a linear formulation with a polynomial number of binary variables and the latter finds proven optimal solutions for the TOP using a Branch-and-Price approach. One of the most thoroughly studied variants of that problem is the Team Orienteering Problem with Time Windows, where each location to visit is accompanied by an additional constraint requiring that it is visited at a certain time window.Vansteenwegen et al. (2009) use an Iterated Local Search combining an insertion step and a shaking step to tackle this problem. Labadie et al. (2012) propose a Granular VNS aimed at reducing the size of the explored neighborhood and thus achieving a shorter computational time.

This work tackles the Capacitated Team Orienteering Problem, a TOP variant where each customer is associated with a certain demand and each vehicle has a maximum capacity. Hence, the additional constraint that must be respected is that the total load of each vehicle cannot exceed its capacity. The variant was introduced by Archetti et al. (2009) with a VNS algorithm encapsulating two Tabu Search algorithms, one for exploring the feasible space and the other for allowing solutions to violate the capacity constraint and then repairing them to produce feasible solutions. Next, Tarantilis et al. (2013) tackled the problem by constructing a heuristic decomposition framework that divides the initial problem in two individual sub-problems: the Knapsack problem for the selection of customers to serve and the Vehicle Routing problem

for the optimal assignment and ordering of customers in vehicles. The solution of the former sub-problem involves a Filter and Fan method and for the latter a Variable Neighborhood Descent procedure is applied. Finally, Ben-Said et al. (2019) propose a Greedy Randomized Adaptive Search Procedure algorithm to solve CTOP. The framework implemented alternates between two different search spaces, the route space and the solution space. The route space refers to the different customer sequencing combinations for a certain subset of selected customers and for this scope the authors make use of an Evolutionary Local Search (ELS) framework which manages to extensively explore the current region of a given solution space before advancing to another one. On top of that method, a Tabu Search as well as a Simulated Annealing scheme is applied for the generation of a solution that will determine the region at which ELS will focus and explore the solution in a deeper level.

Applications of the CTOP can be found in carrier operations. Carriers and transportation companies manage a fleet of vehicles with a specific capacity that can carry a limited amount of products to deliver to customers. In addition, they can outsource the delivery of certain packets, choosing whether or not to undertake the service of a customer, according to the cost of serving that customer relatively to the fleet's designed trajectory as well as the earned profit. Based on this intuition, the problem becomes to model the vehicles' routes in such a way that maximizes the total collected profit.

1.2. Hyper-Heuristics

Solving computationally complex problems using mathematical programming approximation methods can be an arduous task, even when an effective and powerful set of heuristics has been elaborated. This challenge inspired the emergence of approaches that tackle the problem of controlling and coordinating the use of individual heuristics. These methods are called hyper-heuristics and the first definition, coined by Cowling et al. (2001), refers to processes designed to adequately interchange between different low-level heuristics at each decision point, in one word described as 'heuristics to choose heuristics'. However, further works in that subject have extended the research beyond the limitations of that simple definition, presenting methodologies capable of not only selecting but also generating novel operators using given heuristics as building blocks which are combined to form a new one. Therefore, an updated definition of hyper-heuristics that more adequately illustrates the respective current literature would be the one proposed by Burke et al. (2010), which states that they constitute 'a search method or learning mechanism for selecting or generating heuristics to solve computational search problems'. In that sense, the goal of such methodologies is to improve an algorithm's performance by exploring the heuristic space instead of the solution space.

Burke et al. (2010) classify hyper-heuristics in two main categories: heuristic selection and heuristic generation. Evidently, the former refers to methodologies for choosing heuristics and the latter to the combination of already existing for the generation of new ones. Delving one step deeper, each category can be divided in two separate approaches, constructive and perturbative search methods. Constructive paradigms consider partial candidate solutions that are iteratively extended to form a complete solution, whereas perturbative methods modify complete solutions in order to improve their objective function. An example of a heuristic selection framework with construction based heuristics can be found in Garrido, Riff (2010), where an evolutionary algorithm is presented for the solution of the Dynamic Vehicle Routing Problem that generates sophisticated sequences of existing low-level heuristics. Mısır et al. (2012) suggest a perturbative based heuristic selection framework tested on various problems that makes use of an iteratively updated value function in order to determine the optimal heuristic to be selected at each decision point. On the other hand, examples of heuristic generation methodologies can be traced to the works of Runka (2009), who apply Genetic Programming methods for the Travelling Salesman Problem in order to evolve the functionality of a series of operators running under an Ant Colony Optimization framework, as well as in Lon van et al. (2012), who develop a GP based multi-agent system that can solve the dynamic dial-a-ride problem.

In this paper we are interested in learning hyper-heuristics, meaning algorithms that use feedback information on the performance of the low-level heuristics from the search process. Such approaches have been developed for improving performance in vehicle routing problems. Wu et al. (2019) describe a heuristic generation methodology for the TSP and the Capacitated Vehicle Routing Problem applying Reinforcement Learning to learn the solution picking strategy. Qin et al. (2021) suggest a similar approach for tackling the Heterogeneous Vehicle Routing Problem. Finally, Arnau et al. (2018) solve the Dynamic VRP by combining a multi-start metahueuristic with a regression model used to predict the cost of an edge from a set of known variables of the problem.

1.3. Reinforcement Learning

In order for a hyper-heuristic algorithm to be able to learn, it ought to be exposed to an environment for which we possess no prior information and adjust its behavior by gaining continuous feedback from exploring the solution space. Such a task fits appropriately the definition of Reinforcement Learning, which is 'learning what to do, how to map situations to actions, so as to maximize a numerical reward signal. The learner is not told which actions to take, but instead must discover which actions yield the most reward by trying them' (Sutton, Barto 2018). There are two main approaches to solving RL problems: methods based on value functions and methods based on policy search.

Value function methods aim at estimating the expected outcome of an action a taken at state s, i.e. to approximate what the reward will be at the final state if we opt for that action. In a more formal context, we are seeking for a function of the following form:

$$Q^{\pi}(s, a) = \mathbb{E}[R|s, a, \pi]$$

The above function is called Q-function and the process of adapting it to the environment in which we want to perform is called Q-learning (Watkins, Dayan 1992). Given a state and an action, we wish to estimate the expected rewards R if we execute the action on the current state and then follow an optimal policy π thereafter. Arulkumaran et al. (2017) claim that the best policy can be found by greedily choosing the action a with the highest reward at each state, assuming that the function has been formed to accurately derive the rewards at the final state. Hence, in its simplest form, the method is comprised of an estimator that learns to predict rewards applying a greedy strategy for the exploration of the environment. Further works have proposed more complex and advanced approaches. Double Q-learning utilises double estimation to counteract overestimation problems with traditional Q-learning (Hasselt 2010). One estimator serves as an optimal Q-value function approximator (essentially a Q^* function) producing the expected rewards. The other is used to learn to adjust to the discrepancy between the predicted and the expected rewards. The two estimators change their role at every learning step so that they can both learn from one another. Advantage function learning (Wang et al. 2015) offers a different take on the value function, which is formed as: $A(s, a) = Q^{\pi}(s, a) - V(s)$, where $V(s)$ is the value function of the state. This implies that we consider the difference of Q-value and the average of actions which it would have taken at that state. It gives the extra reward that could be obtained by the agent by taking that particular action.

Policy search methods have a more intuitive and straightforward approach: instead of predicting the impact of an state-action pair on the final outcome, they attempt to determine the optimal strategy, namely the appropriate course of actions that maximizes the expected final reward. Hence, a parameterized policy π_{θ} is learned, whose parameters are constantly updated to maximize the expected reward after a certain number of time steps. The parameters can be updated either by using gradient based methods (Heess et al. 2015, Lillicrap et al. 2015) or by gradient-free methods, such as Hill Climbing and Genetic Programming (Gomez, Schmidhuber 2005, Koutník et al. 2013). The most common application of policy search methods is the Monte Carlo REINFORCE rule (Williams 1992), where we use a certain function both for updating the parameters of the policy using stochastic gradient ascent and for exploring the environment by sampling from the constantly reevaluated policy.

Over the years some hybrid approaches that have proven to work effectively have been proposed, such as the actor-critic model. This methodology suggests that a policy search method —the actor— determines the transition to next states by following a specific action at each step and a value approximation method - the critic - trains the actor by providing feedback over the outcome, thus combining the advantages of both policy search and value based algorithms (Barto et al. (1983)).

For the scope of this paper, we are interested in Deep Reinforcement Learning, an aspect of RL in which the model to be trained, either a policy method or a value function approximator, is a neural network. The evolution of neural networks and their predominance in solving complex problems over other methods has pushed the academic interest towards their application in the field of Reinforcement Learning, with the advent of algorithms such as Deep Q Learning (Mnih et al. 2015), Deep Actor-Critic (Lillicrap et al. 2015) and Deep Policy Gradient (Schulman et al. 2017). It is worth mentioning that, although the aforementioned RL algorithms as well as neural networks have been present in the literature for many decades, their combination as a research contribution has only been made possible during the last years. This fact gives us an insight on the level of difficulty and complexity in terms of creating a robust and stable model that will accurately depict the target environment and that can be trained in a reasonable computational time.

Despite the evident challenges, the field of DRL has attracted a great amount of research interest over the last years and its generalizability in combination with the promising performance it often yields have extended its applications further than the obvious domains. Consequently, applications of DRL can also be found in Vehicle Routing Problems like the one we tackle in this project. Bello et al. (2016) construct a Recursive Neural Network (RNN) for the Traveling Salesman Problem and manage to achieve close to optimal results. Nazari et al. (2018) attack the classical VRP problem by implementing a RNN with Attention Mechanism which is used as a black-box heuristic that distorts a given solution to produce a new one. Khalil et al. (2017) work on the Traveling Salesman Problem and construct a Deep Q-Network that learns a policy for implementing a node selection heuristic that works within a greedy algorithm framework. Finally, Kool et al. (2019) managed to outperform previous DRL works on TSP and CVRP replacing deprecated RNN architectures with Transformer models (Vaswani et al. 2017), capable of employing more effective attention mechanisms.

In this project we develop a perturbative-based heuristic selection hyper-heuristic framework for the solution of the CTOP. We construct a set of low-level heuristics capable of altering components of a complete solution, each one exploring a limited neighborhood space. We attempt to unify and coordinate these operators by designing a Deep Q-Learning based value function approximation method for predicting the outcome of each heuristic given the state of the solution space. We also construct a couple of intensification methods aimed at repositioning the solution to neighborhoods of higher quality when stuck in local maximums and thus producing more promising solutions. The remainder of the project is organized as follows: Section 2 presents the proposed optimization framework and analyzes the individual algorithmic components. Then, in Section 3 computational experiments are provided. Finally, Section 4 summarizes and concludes the project.

2. Methodology

This section presents the proposed optimization algorithm for the CTOP. The overall scheme is initially outlined. Then, the various algorithmic ingredients are separately discussed: the initial solution construction process is firstly presented, followed by the neighborhood search moves. Moreover, powerful intensification methods capable of achieving broader exploration of the solution space are discussed. These methods include solving a suitably defined Traveling Salesman as well as an Integer Programming model able to perform multiple customer deletions and insertions. Following that, the hyper-heuristic framework is presented, a Deep Q-Learning implementation responsible for forming the algorithmic policy in terms of selecting a move on each iteration based on the expected long-term reward for each move on the current point of the solution space. The environment exploration strategy is analyzed, a concept for balancing between exploitation and exploration of the solution space. Finally, our value function approximator is presented, a neural network called Q-network, where its architecture and parameter update process are explained in detail.

2.1. Overall scheme

The proposed algorithm can be decomposed in two parts: the learning process and the execution process. During the learning process, a Deep Q Network is trained on the instance specified in order to obtain knowledge about the environment and be able to predict the long-term rewards of every available action on a given state. After the network has been trained, the execution cycle starts which repeats for a predefined number of iterations. At every iteration, we find the best action to perform by choosing the one with the highest reward predicted by the DQN. We execute the action to get the new solution and check if a new best has been found. If a new best solution has not been found for a certain number of consecutive iterations, the intensification procedure is called to transfer the solution to a new neighborhood. Specifically, the intensification procedure consists of two methods: the first refers to a Mixed Integer Programming (MIP) based move which can optimally delete and insert up to a given number of customers from and to the route for every route of the solution. The second is a powerful TSP heuristic for optimally sequencing the customers of each route of the solution. The overall scheme of the suggested framework is presented in Algorithm 1.

Initially, a feasible solution S is constructed (line 1). The DQN is trained on the instance (line 2). For every iteration we track and execute the best action by selecting the action a that maximizes the expected rewards (lines 4-5): $a = \{a_i \in A : DQN(s, a_i) = \max\{DQN(s)\}\}$. The global best solution S^* is updated in lines 6-8. The intensification methods are executed under the condition that a certain number of iterations has passed without improvement of the S^* (lines 9-14).

Algorithm 1 Overall Scheme of *DQL-HH*

1: $S \leftarrow$ *ConstructInitialSol()*, $S^* \leftarrow \emptyset$, $iter\_unimproved \leftarrow 0$
2: $DQN \leftarrow$ *TrainDQN()*
3: **for** $i \leftarrow 1$ to *max_iterations* **do**
4: $\quad action \leftarrow$ *ChooseAction(DQN, S)*
5: $\quad S \leftarrow$ *Execute(action, s)*
6: $\quad$ **if** $Z(S) > Z(S^*)$ **then**
7: $\quad\quad S^* \leftarrow S$
8: $\quad\quad iter\_unimproved \leftarrow 0$
9: $\quad$ **end if**
10: $\quad$ **if** $iter\_unimproved = max\_iter\_unimproved$ **then**
11: $\quad\quad S \leftarrow$ *Intensify(S)*
12: $\quad\quad iter\_unimproved \leftarrow 0$
13: $\quad$ **else**
14: $\quad\quad iter\_unimproved \leftarrow iter\_unimproved + 1$
15: $\quad$ **end if**
16: **end for**
17: **return** S^*

2.2. Solution construction

A greedy construction algorithm is proposed for creating an initial solution. Unserved nodes are iteratively inserted into the solution using as a selection criterion the value $s = M * \Delta profit - \Delta cost$. At every iteration, insertions of nodes that do not violate any constraint are considered. The algorithm uses a l-sized restricted candidate list for storing at each iteration only the l moves with the highest score s and then finally applying a randomly chosen one. The procedure repeats continuously and terminates when no feasible insertion is identified. This construction algorithm is used only in the first cycle of the proposed framework. A detailed outline of the construction algorithm is presented in Algorithm 2.

In line 1 we initialize an empty solution and a list of all unvisited nodes. At every iteration we initialize the restricted candidate list of feasible insertions (line 3). Lines 4-6 specify all the different combinations that will be examined at each iteration: for every unvisited node, every position of every route is checked for whether it generates a feasible insertion (line 7). If it does so, the move's score ($s = M * \Delta profit - \Delta cost$) is calculated and if it is greater than any element of the l-sized rcl, it is inserted to the list. In lines 16-20, a move is selected and executed at random and the inserted node is removed from the unvisited.

2.3. Neighborhood search

We define the following six low-level heuristics that explore exhaustively a narrow neighborhood:

- *Insertion*: it inserts any customer of the set of unserved customers into any available route position at any route.
- *Deletion*: it removes any customer served by the routes.
- *Outer Swap*: it replaces any served customer with any unserved customer.
- *Inner Swap*: it interchanges the positions of any two served customers.
- *Relocation*: it relocates any served customer to a new position at the same or at a different route.
- *2opt*: it relocates sequences of customers between any two routes in order to restore paths that cross over one another.

Algorithm 2 *Construct Initial Solution*

```
1: N_unvisited ← N, S ← ∅
2: do
3:     rcl ← []
4:     for node in N_unvisited do
5:         for route in routes do
6:             for pos ← 1 to |route| − 1 do
7:                 if InsertionIsFeasible(node, route, pos) then
8:                     s ← CalculateScore(node, route, pos)
9:                     if ScoreFitsRCL(rcl, s) then
10:                        UpdateRCL(rcl, [node, route, pos, score])
11:                    end if
12:                end if
13:            end for
14:        end for
15:    end for
16:    if |rcl| > 0 then
17:        move ← SelectRandom(rcl)
18:        UpdateSolution(S, move)
19:        N_unvisited.Remove(move[0])
20:    end if
21: while |rcl| > 0
22: return S
```

The objective of every move that belongs to the aforementioned operators is calculated as: $z(m) = M * p(m) - c(m)$, where $p(m)$ and $c(m)$ denote the profit and the cost change respectively incurred by applying m and M is a large positive value. Moves are evaluated only if they lead to feasible solutions. Each operator acts in a greedy manner by opting for the move that maximizes its objective among all the possible combinations that do not violate any constraint.

2.4. Intensification methods

As mentioned in the overall description of our framework, when the trained network gets stuck at local optimum and does not manage to improve the solution for a certain number of iterations, two powerful and advanced intensification methods are deployed to resolve that and transfer the solution to a new and potentially of higher quality neighborhood. The two methods, which are serially and independently applied are described in the following paragraphs.

2.4.1. TSP

The candidate solution is composed of a set of routes starting from and ending to the depot. With that in mind, if we hold fixed the customers served in every route, we can think of a new sub-problem worth solving: the problem of reducing the cost of the fixed route, which can implicitly improve the solution's quality by allowing more customers to be accomodated. This problem can be precisely formulated as a TSP problem, where the target is to find the optimal path that passes through all points starting and ending to a specific point. The TSP problem is optimized via the LKH TSP solver of Helsgaun (2000). The solver is based on the well-known Lin-Kernighan heuristic (Lin, Kernighan 1973). For the test cases that involve a few hundreds of nodes, LKH usually obtains the optimal solutions very fast (less than a CPU second in our testing environment). Thus, we apply a fast and effective cost reduction method by implementing the TSP solver proposed by Lin-Kernighan-Helsgaun at every route of the solution separately.

2.4.2. Customer Insertion-Deletion MIP

The second method designed for deeper exploration of the solution space constitutes a matheuristic for solving an IP model which considers multiple customer insertions and deletions on a given solution S. We call this model Customer

Insertion-Deletion MIP (*CID*). The maximum number of customer insertions and removals are determined by two model parameters which control how smooth or steep will be the transition of the solution to a new neighborhood after the execution of the MIP model. Obviously, only customers that are not served in S may be inserted, and only customers already served in S may be removed.

To facilitate the *CID* model description, let us introduce the following notation. Let N denote the set of all available customers. A profit p_i, a demand d_i and a service time s_i is associated with every customer i. Also, a cost c_{ij} is associated between any two customers i and j. Moreover, y_{iv} will be a binary variable equal to 1 iff node $i \in N$ is served by vehicle $v \in V$. In addition, routing variable x_{ijv} is equal to 1, if the arc $(i, j) \in A$ is traversed by vehicle $v \in V$ and 0 otherwise. Furthermore, let $N^+ = \{i \in N \mid \sum_{v \in V} y_{iv} = 1\}$ be the set of customers served in solution S, and $N^- = \{i \in N \mid \sum_{v \in V} y_{iv} = 0\}$ be the unserved customers. Also, let constant I_i denote the insertion cost (in terms of c_{ij}) of any customer $i \in N^-$ in any route $v \in V$ of the solution. This insertion cost is approximated by considering all possible insertions of every customer $i \in N$ into every possible insertion position of the current solution and by obtaining the lowest cost. Similarly, constants R_i are the cost savings of removing customer $i \in N^+$ from a route v in solution S. Obviously, when only one insertion or removal takes place, the actual cost modification matches the one calculated by the approximation. However, when multiple insertions occur, these can take place in consecutive solution positions, and as a result the approximation modification cost does not match the actual one.

Additionally, let us introduce binary variables α_{iv} and λ_{iv} which are equal to 1, if customer i is inserted or removed from a route traversed by vehicle v, respectively. Variables α_{iv} and λ_{iv} are only defined for customers such that $y_{iv} = 1$ and $y_{iv} = 0$, respectively. Finally, parameters T_I and T_R limit the maximum number of allowed insertions and removals, respectively.

Below, the *CID* MIP is presented:

$$\max_{\alpha,\lambda} f = \sum_{i \in N^-} \sum_{v \in V} \alpha_{iv} p_i - \sum_{i \in N^+} \sum_{v \in V} \lambda_{iv} p_i \tag{1}$$

subject to

$$\sum_{i \in N^-} \alpha_i \leq T_I \tag{2}$$

$$\sum_{i \in N^+} \lambda_i \leq T_R \tag{3}$$

$$\sum_{(i,j) \in A} (c_{ij} + s_{ij}) x_{ijv} + \sum_{i \in N^-} \alpha_{iv} I_i - \sum_{i \in N^+} \lambda_{iv} R_i \leq T_{max} \qquad , \forall v \in V \tag{4}$$

$$\sum_{i \in N} d_i y_{iv} + \sum_{i \in N^-} \alpha_{iv} d_i - \sum_{i \in N^+} \lambda_{iv} d_i \leq L_{max} \qquad , \forall v \in V \tag{5}$$

$$\alpha_i \in \{0, 1\} \qquad i \in N^- \tag{6}$$

$$\lambda_i \in \{0, 1\} \qquad i \in N^+ \tag{7}$$

Objective function (1) maximizes the difference between the additional profit collected by the inserted customers (first term) minus the profit lost due to customer removals (second term). Constraint (2) ensures that up to T_I customers are inserted into the solution. Similarly, constraint (3) limits the number of removed customers to T_R. Constraint (4) guarantees that the the total route cost for every solution vehicle does not exceed the maximum duration T_{max}. The first term represents the current route duration. The second one is the extra cost incurred for the same route serving the inserted customers and the third one is the cost savings enjoyed by removing customers. Note that this constraint does not guarantee that the modified solution cost respects the maximum duration. The use of approximate routing costs may result into infeasible solutions, especially when large values of T_I and T_R are used. Moreover, constraint (5) reassures that

the maximum vehicle capacity constraint L_{max} will not be violated by insertions and removals of customers to the route. Finally, constraints (6) and (7) are the binary variable bounds.

It is worth mentioning that in practice, the pre-calculated insertions and removal costs used to formulate and execute the model will not reflect the actual ones and may significantly differ from the real costs. This is due to the fact that when multiple removals and insertions take place simultaneously, the insertion or removal cost for each customer diverges based on the solution structure after each action. Therefore, in order to resolve this discrepancy and to ensure at the same time that *CID* can be efficiently solved for large-scale instances, all customer insertion and removal costs are approximated based on the given solution S. The insertion cost of a customer i in a route v of the solution S is optimistically set as the cost of inserting this customer between the best (lower cost increase) pair of nodes at any route of the solution. The removal cost of a customer i from a route v of the solution S is the difference in cost between the route containing that customer and the route without that customer.

As a consequence to the implemented approach, the optimal *CID* solution may violate the *CTOP* maximum duration due to the approximate costs used for customer removal and insertions. If an infeasible solution is produced, feasibility is heuristically restored by iteratively removing customer nodes. At each iteration, the customer yielding the highest cost savings is removed from the solution. Note that customers pushed into the solution by the *CID* model are not considered as candidates for being removed when the feasibility is restored.

2.5. Learning Mechanism

The learning mechanism implemented is based on the Deep Q Learning framework proposed by Mnih et al. (2015). It was the first effective combination of the Q-learning methodology and deep networks as agents and its performance was tested on seven Atari 2600 games.

As already explained, Q-Learning is based on the idea of estimating expected future rewards of the available actions using a state-action value function called Q-function, which is defined as:

$$Q^{\pi}(s,a) = \mathbb{E}[r + \gamma max_{a'} Q^{\pi}(s',a')]$$

It is obvious by this definition that Q-function estimates rewards by taking into account two components: the immediate reward that derives from executing the action a as well as the long-term reward (discounted by a factor γ) from following the optimal policy π from the state s' and thereafter. It is worth mentioning that the Q-function is a Bellman optimality equation, which approaches the task of determining the rewards at the final state by having information only on the current state from the scope of dynamic programming: it breaks up the above task in two parts, finding the best reward for the next step, and finding the best reward from the next step until the final. Using the Q-function itself as an estimator for the second part and iteratively updating it as we move on to the next steps, the method eventually converges - at least theoretically - to the optimal Q-function.

For most problems, passing the whole state as a parameter to the Q-function can be an arduous and computationally challenging task. The technique that is most frequently applied in such cases is to use instead a set of features which offer essential information that expresses a certain state and distinguishes it from others. In our project, we make use of a 10-feature vector as an input to the Q-function. The exact solution attributes used as features are explained in the section 2.5.3.

Another common technique in RL research that we employ is that of Experience Replay (Lin (1992)). This approach aims at increasing the level of stability of the trained network by using as training sample a randomly chosen subset of previous experiences. This can be achieved through storing previous steps applied by the exploration mechanism to a structure called replay buffer. Such a method offers two types of benefits: Firstly, we avoid training the agent on sequential experiences that emerge from step-by-step exploration and would result to poor learning due to high correlation of the data. Secondly, we attain better data efficiency by reusing each transition in many updates and not only once as is proposed by the traditional Q-learning algorithm.

The suggested framework is thoroughly discussed in the next sections. In particular, a description of the training algorithm is provided and explained. In continuation, we analyze the strategy for moving on new states of the solution space in order to amplify our agent's range of exploration and to discover promising regions. Finally, the Q-network agent is discussed and its architecture as well as its parameter update procedure are examined.

2.5.1. Training Algorithm

Algorithm 3 presents an outline of the learning mechanism:

Algorithm 3 Train *Deep Q Network*

Input: v, s, k, episodes, ep_duration, steps, γ, τ, b, epochs, l
Output: DQN

InitializeEnvironment(s)
$DQN \leftarrow$ *InitializeQNetwork(v, l)*
$DQN\_target \leftarrow$ *InitializeQNetwork(v, l)*
for $ep \leftarrow 1$ to *episodes* **do**
    $Buffer \leftarrow []$
    $states \leftarrow$ *ResetEnvironment()*
    for $i \leftarrow 1$ to *ep_duration* **do**
        $features \leftarrow$ *GetFeatures(states)*
        $actions \leftarrow$ *Explore(features, gamma, DQN_target, k)*
        $states, rewards \leftarrow$ *Execute(actions, gamma)*
        Buffer.Add([features, actions, rewards, states])
        if $i \mod steps = 0$ **then**
            $sample \leftarrow$ *Sample(Buffer,* τ*)*
            $inputs \leftarrow$ *sample*$[:, 0]$
            $targets \leftarrow$ *CalculateTargets(sample, DQN_target)*
            for $epoch \leftarrow 1$ to *epochs* **do**
                $batches \leftarrow$ *BatchPartition(inputs, targets, b)*
                for $batch \leftarrow 1$ to *batches* **do**
                    $loss \leftarrow ComputeLoss(batch)$
                    $grads \leftarrow$ *CalculateGradients(loss, DQN)*
                    UpdateQNetwork(grads, DQN, l)
                end for
            end for
        end if
    end for
    $w_{DQN\_target} \leftarrow w_{DQN}$
    $\gamma \leftarrow \gamma + t$
end for
return DQN

For the training process to commence, the following parameters must be defined:

- *v*: the number of vehicles forming a solution. Note that the learning mechanism is instance-based, therefore we need to explicitly determine the instance specific parameter.
- *s*: the number of parallel solutions guided by the simulation
- *k*: the number of hash functions applied to form the state's hash code
- *episodes*: the number of simulation-training phases
- *ep_duration*: the number of steps in every episode simulation
- *steps*: the step frequency by which we train the model on a sample of the replay buffer

- γ: the discount factor applied to target network's predicted Q-values in order to reduce their influence on the exploration procedure as well as the target rewards. γ is initially set at a small value between 0 and 1 and is constantly increased with a steady rate. The intuition behind this approach is that at first, the network is not capable of making good approximations of the expected rewards and hence we should not count on its predictions. However, as it is gaining experience of the environment and learning more accurately we want to increase the level of confidence that we give to its approximations.
- τ: the size of the sample drawn from the replay buffer
- *b*: the batch size for training
- *epochs*: the number of epochs for training
- *l*: the learning rate

In line 1, the environment is initialized with instance parameters that set the boundaries and the characteristics of the solution space. In addition, the Solution Construction method is called to generate the initial s states, from where the exploration will set off at every episode. In lines 2 and 3, the trained and the target network are initialized respectively and their architecture is set based on the number of vehicles for the instance, as described in 2.5.3.

Lines 5-12 depict the simulation process which works as follows: For each episode, a replay buffer storing all steps in the solution space is initialized. States are set to their starting solutions. At every step, state features are extracted which will serve as inputs to the Q-network. Then, actions for every state are chosen implementing an exploration strategy explained in 2.5.2. Actions are executed and the immediate rewards along with new states are calculated. The features of the current state, the selected action, the next step reward and the next step state are stored in the replay buffer to form an experience.

At every predefined number of steps, the DQ network is trained on samples derived from the replay buffer. Based on a random sample drawn from the buffer, the training inputs and the targets are formed (lines 13-15). Inputs are simply the state features for every example of the replay buffer and will be used to compute the predicted Q-values. Targets represent the "true" values resulting from the formula $y = r + \gamma * maxQ_{target}(s_i; \theta)$. Then, during each epoch, the above values are partitioned into batches (line 17). For every batch, network parameters are updated by gradients calculated based on the loss function, a process described in 2.5.3 (lines 17-21).

In line 26, the target network is updated based on trained network's updated weights. In line 27, γ is increased by a precalculated constant t, which is set to $\frac{1-\gamma_0}{episodes}$, where γ_0 is the initial value at which γ is set. This ensures that γ will be steadily increased at each episode until it maximizes at 1 at the end of the final episode, where the training procedure will be over and we expect to be able to fully "trust" our network's predictions.

Notice that during the exploration of the solution space only the target network is used, whereas for the learning process we make use of the target network to compute the target values and the trained network to perform the gradient steps. This approach aims at providing a more stable learning mechanism, since trained network's parameters are updated during each batch, but the target network will be updated only after the episode is terminated and thus its recent experiences will not affect its trajectory.

2.5.2. Exploration Strategy

The exploration strategy is based on the idea of intrinsic rewards (Oudeyer, Kaplan (2013)) and aims at balancing between exploitation and exploration by reinforcing not only actions that render a high reward but also actions that lead to points of the environment that we haven't visited before. That being said, the exploration strategy for a given state s is defined as follows:

$$a = \{a_i \in A : R_{a_i} = \max\{r(s, a_i) + r'(s, a_i) + u\}\}$$

For all the available actions on each state, we select to execute the one with the highest sum of exploitation and exploration reward respectively as well as a vector u of random rewards used for avoiding getting stuck in local cycles. The exploitation reward is simply the normalized immediate reward that derives from executing the action. On the other

hand, the exploration reward is calculated as the normalized euclidean distance between the next state, i.e. the state that will emerge if we choose to execute the action, and the nearest state that we have ever visited. This metric will give us an idea of how far the next state is from the directly closest state we have already visited and thus how "unexplored" the candidate state is.

$$r'(s, a_i) = \min\{d(s, s_j)\} \qquad , \forall s_j \in Buffer$$

However, the process of finding the closest state among the visited ones and calculating the distance from the said state is a computationally expensive task. For this reason, we develop an additional feature with a view of avoiding as much trivial calculations as possible. The method is based on Locality-Sensitive-Hashing (LSH) and has been inspired by the works of Charikar (2002) and Tang et al. (2016). The intuition behind this method is that we attempt to apply a series of hash functions on a state in order to retrieve a hash code. States (feature-vectors) with similar values will receive the same hash code, whereas vectors with distant values will have different code. In this way we can narrow down the search for the closest vector only on the ones with the same hash code. The procedure for generating the hash code, according to Charikar (2002), is comprised of the following steps:

1. Initialize k random vectors drawing values from the standard Gaussian distribution.
2. Calculate the angular distance between the state vector and each of the k vectors as

$$1 - \frac{\theta(s, r_i)}{\pi}, \forall i \in \{0, 1, ..., k\}$$

   This will result to a vector v of size k.
3. Turn the v vector into a binary vector b by setting

$$b_i = \begin{cases} 1, & v_i \geq 0.5 \\ 0, & v_i < 0.5 \end{cases}$$

4. Generate the hash code by transforming the binary vector to a decimal number.

The value of k can have a double and counter-acting effect. Setting k to a large value can increase the number of calculations needed to derive the hash function, though at the same time it enhances the resolution of the codes, partitioning vectors to more sparse bins and therefore reducing the number of calculations needed to find the closest vector.

2.5.3. Q-Network

In the following paragraphs we present the design and functionality of the Q-network. The purpose of the network is to predict the expected long term rewards of every action on a state, given its features.

Architecture. The neural network will be trained as a value function approximator capable of predicting the long-term rewards of applying a certain action on a specific state. It is designed to perform as a feed-forward network. The input to the network will be a matrix representing the features that characterize and distinguish a state. These features are calculated route-wise, meaning we construct a feature vector for each route separately and then append these vectors to form the feature matrix. The size of the matrix is $(|V|, |F|)$, where V stands for the vehicles available (instance-specific parameter) and $|F|$ is the number of features determined for a single route. Based on the structure and variables of the problem, we set $|F| = 10$ as we keep track of the following features:

- Profit of route
- Profit per node
- Variance of profit per node
- Load of route

- Load per node
- Variance of load per node
- Cost of route
- Cost per node
- Variance of cost per node
- Nodes served

The output of the network will be a vector of size A reflecting the Q-value of every action. Furthermore, we elaborate on the inner architecture of the network, which consists of the layers as below:

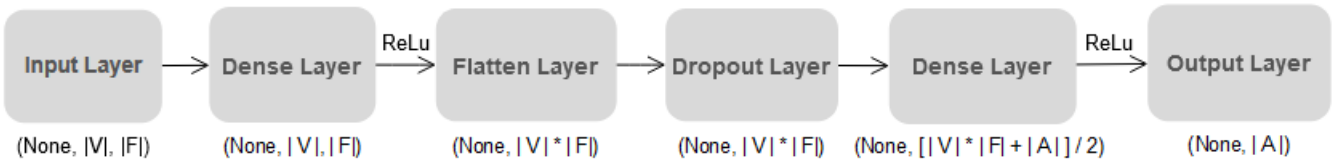

Figure 1: Architecture of the Q-Network

As explained earlier as well, the input layer will have the shape of the solution feature matrix. The inputs are fed to the first hidden layer of size $(|V|, |F|)$, namely a $|F|$-neuron layer for each of the V input vectors. We implement a ReLu activation. At the next step, the neuron matrix is flattened to produce $|V| * |F|$ neurons and a dropout operation is performed. Then, a dense layer is built on top of the flattened neurons with size of $\frac{|V|*|F|+A}{2}$ neurons and a ReLu activation is applied. Finally, we add an output layer of A neurons representing the Q-value vector. The activation of the output layer will be linear, since rewards can be positive as well as negative, so we don't want to constrain the output with any transformation.

Loss function and Network update. The loss function used to update the weights follows the Temporal Difference (TD) principle as the one presented in Mnih et al. (2015), where it is calculated as:

$$L_i(\theta_i) = (y_i - Q(s_i, a_i; \theta_i))^2 \tag{8}$$

where $y_i = r_i + * maxQ_{target}(s_{i+1}; \theta_t)$.

As explained above, during each exploration step only one action is executed. This implies that, albeit the network has predicted the Q-values for every state-action pair, the target value corresponds to a single pair at each step and therefore we are able to compute the loss only on the respective pair. However, we must also be consistent with the network architecture and assure that the target values have the same shape as the vectors of predicted values. Hence, we will transform the target value into a vector at the following way:

For A available actions the predicted Q-values will be:

$$y_{pred} = \begin{bmatrix} q_0 & q_1 & \dots & q_{A-2} & q_{A-1} \end{bmatrix}$$

Let's consider a to be the selected action on a certain step and y the respective target reward. We construct the A-length vector of target values using the below function:

$$q^*(i) = \begin{cases} e^y, & i = a \\ 0, & otherwise \end{cases} \tag{9}$$

This will result to a target vector as below:

$$y_{target} = \begin{bmatrix} 0 & 0 & \dots & e^{y} & \dots & 0 & 0 \end{bmatrix}$$

Having formed the above vectors, we can compute the loss defined in (8) by applying a series of calculations:

$$L_i(\theta_i) = [\ln(y_{target} * e^{-y_{pred}})]^2 \tag{10}$$

This way, when we perform a matrix multiplication between y_{target} and $e^{-y_{pred}}$, the resulting value will be e^{y-q_a}, since all other terms of the sum will be 0. From there, we apply the natural logarithm to remove the exponential and then by squaring the difference we are left with the calculation that we aimed for at the first place. The formulation delineated above facilitates the process of calculating the loss as well as the gradients of the network, since the whole procedure can be represented as a series of function applications and algebraic operations.

3. Computational results

This section summarizes the computational experiments and results obtained by the proposed hyper-heuristic. First of all, the state of the art approaches relatively to CTOP are described. Then, the standard parameter setting for the suggested algorithm is presented. Finally, analytic comparisons between the solution scores obtained by the standard setting of the proposed algorithm and the scores achieved by current state-of-the-art CTOP methodologies are drawn.

3.1. State-of-the-art approaches

The first paper by Archetti et al. (2009) proposed a VNS algorithm encapsulating two Tabu Search algorithms, one for exploring the feasible space and the other for allowing solutions to violate the capacity constraint and then repairing them to produce feasible solutions. They also introduced a set of 130 benchmark instances on which they deliver their algorithm performance results. Later on, Tarantilis et al. (2013) developed a Bi-Level Filter and Fan method that adequately combines two different heuristics that focus on different search spaces. The first heuristic applies a Filter and Fan method for the solution of the Knapsack sub-problem, while the second solves the route minimization sub-problem with the use of a Local Search methodology. Due to their proposal's dominant performance, the authors introduced a new set of 130 more complex instances and provided their results without comparison with the VNS method of Archetti et al. (2009). Most recently, Ben-Said et al. (2019) proposed a Greedy Randomized Adaptive Search Procedure algorithm to solve CTOP. The algorithmic framework was comprised of two techniques. In a lower level, an Evolutionary Local Search (ELS) framework was used for the exploration of the current region of a given solution space before advancing to another one. Controlling this method was a Tabu Search as well as a Simulated Annealing scheme that determined the region at which ELS would focus and apply the deeper exploration. The authors provided benchmark results on two proposed methodologies, one implementing Tabu Search and the other applying Simulated Annealing, named VSS-Tabu and VSS-SA respectively.

Unfortunately, we were not able to obtain access to the original set of smaller instances and so our computational results will include only the comparison regarding the newly generated set of instances. The proposed algorithm (*DQN-HH*) is coded in C# and the algorithm that trains the Deep Q Network is written in Python. The *CID* model is solved by Gurobi 9.5.2, whereas the *TSP* was tackled by the LKH TSP solver of Keld Helsgaun (Helsgaun 2000) (publicly available at http://webhotel4.ruc.dk/∼keld/research/LKH/). The overall algorithm was executed via a single thread to enable comparisons. A time limit of 60 seconds per *CID* call is imposed. The computational experiments were carried out on an Intel Core i7-8700 3.20 GHz CPU with 16 gigabyte RAM x64 Windows 10 machine. A summary of the methodologies implemented by the three previously published CTOP papers as well as the algorithm proposed in the present work is depicted in Table 1.

3.2. Standard Algorithmic Setting

The proposed algorithmic development contains various parameters which were tuned after performing preliminary experiments on selected instances. However, it should be noted that more extensive parameter calibration experiments

Table 1: Algorithmic frameworks for the CTOP problem

Reference	Acronym	Approach	Threads	CPU	Solver
Archetti et al. (2009)	VNS	Tabu Search	Default	Intel Pentium 4 CPU 1.60 GHz (1048 GB RAM)	CPLEX 9.0
Tarantilis et al. (2013)	BiF & F	Local Search	1	Intel Core2 Quad 2.83 GHz	-
Ben-Said et al. (2019)	VSS-Tabu & VSS-SA	GRASP	1	Intel Xeon 2.6 GHz	-
This project	DQN-HH	Hyper Heuristic	1	Intel Core i7-8700 3.20 GHz (16 GB RAM)	Gurobi 9.5.2

could reveal a set of parameter values that improve the overall performance of the algorithm. It should be noted that a dimension that was taken into account for the final adjustment of the parameters was the computational time reported by previous papers. Since we aimed to construct a methodology that would be competitive in terms of time complexity as well, we deliberately modified the parameter values in order to produce results at around the same time. In Table 2, the algorithm parameters are reported along with their standard values and grouped by algorithmic component.

Table 2: Standard parameter setting for the DQN-HH

Component	Parameter	Description	Standard Value
Local Search	max_iterations	Limit on total number of LS iterations	10000
	max_iter_unimproved	Maximum number of non improved LS iterations	200
CID	max_insertions	Maximum number of customer insertions allowed	10
	max_deletions	Maximum number of customer deletions allowed	10
TrainDQN	s	Number of parallel solutions guided by the simulation	10
	k	Number of hash functions calculating a state's hash code	50
	episodes	Number of simulation-training phases	30
	ep_duration	Number of steps in every episode simulation	500
	steps	Step frequency for the training phase	10
	γ	Q-value discount factor for the first episode	0.1
	τ	Size of replay buffer sample	$\min\{\|Buffer\|, 5000\}$
	b	Size of training batch	25
	epochs	Number of training epochs	3
	l	Learning rate	0.01

3.3. Comparison with state-of-the-art algorithms

For the execution of the computational experiments, the following process was followed: for every instance, we ran the training algorithm in order to train the DQ network on the specific instance and then applied the proposed algorithmic scheme a certain number of times. On each one of the instances we run 20 repetitions of the algorithmic framework. In Table 3 we present a summary of the relative performance of every algorithm in a cross table, the Bi-F & F of Tarantilis et al. (2013) that includes two implementations, a faster and a slower one, the VSS-Tabu and VSS-SA proposed by Ben-Said et al. (2019) and finally DQN-HH, our proposal. Every cell refers to the number of instances for which the row algorithm produces better results than the column algorithm. We notice that our approach does not manage to outperform any of the previous works and contributes the least to the best known solutions in literature. However, it is worth mentioning the nature of the instances for which it achieves improved results in comparison with the others. Table 4 presents the comparative computational results for the instances that our algorithm produces the best results. For every algorithmic framework three columns are shown: the objective found by the best execution of the algorithm (Z_{max}), the

percentage gap between the current objective and the best among the algorithms of the previous papers (RPE) and finally the execution time of the best repetition of the algorithm (TTB). The reported time for our case was calculated as the sum of the agent's training time on the specific instance and the execution time of the best repetition. We notice that for these instances the computational time of our algorithm is significantly larger; though, as it is shown in the analytical tables found in the Appendix, our methodology is on average faster than most methods and slower only in comparison with the fast implementation of BiF & F. By examining the results stated in the table, we observed that all these instances have something in common that distinguishes them from the rest: the vehicles parameter is significantly larger with values ranging from 14 to 24, whereas for all the remaining instances that number oscillates between 6 and 8. Therefore, we can deduce that our methodology achieves deeper exploration of the solution space when the combinatorics of the problem increase by the presence of a larger fleet of vehicles. The analytical tables with the complete results for every instance are presented in the Appendix.

Table 3: Instance improvement for every method

	BiF&F-f	BiF&F-s	VSS-Tabu	VSS-SA	DQN-HH
BiF&F-f	-	13	4	3	107
BiF&F-s	61	-	11	6	113
VSS-Tabu	91	76	-	27	118
VSS-SA	96	79	30	-	118
DQN-HH	20	13	10	10	-

Table 4: Analytical results for improved performance instances

Instance	BiF&F-f			BiF&F-s			VSS-Tabu			VSS-SA			DQN-HH		
	Zmax	RPE	TTB	*Zmax*	RPE	TTB	*Zmax*	RPE	TTB	*Zmax*	RPE	TTB	*Zmax*	RPE	TTB
01-337-14-345-720	4172	0	17	4172	0	17	4172	0	1	4172	0	2	4180	-0.192	502.065
02-385-16-350-713	4784	0	25	4784	0	25	4784	0	1	4784	0	3	4792	-0.167	625.565
03-433-18-330-675	5201	0	33	5201	0	32	5201	0	2	5201	0	3	5209	-0.154	704.52
04-481-20-335-713	5828	0	48	5828	0	47	5828	0	1	5828	0	3	5842	-0.240	798.165
05-529-22-340-705	6445	0	101	6445	0	98	6445	0	3	6445	0	5	6462	-0.264	1210.895
06-577-24-365-683	7071	0	94	7071	0	93	7071	0	1	7071	0	2	7082	-0.156	1552.05
07-361-15-335-668	4355	0	24	4355	0	23	4355	0	1	4355	0	3	4366	-0.253	476.5
08-433-18-350-675	5194	0	51	5194	0	49	5194	0	2	5194	0	4	5200	-0.116	1040.14
09-505-21-360-660	6183	0	103	6183	0	104	6183	0	3	6183	0	22	6196	-0.210	1076.43
10-577-24-375-675	7239	0	144	7239	0	144	7239	0	4	7239	0	7	7252	-0.180	1564.145

4. Conclusions

In the present work a novel hyper-heuristic algorithm is developed for the Capacitated Team Orienteering Problem (CTOP). CTOP is a multi-vehicle orienteering problem, thus it is aimed at maximizing the profit collected from customers subject to a maximum time duration. Every customer is associated with a profit and a demand whereas every vehicle has a maximum capacity of demand units that it can carry. Hence, from a theoretical standpoint, it can be perceived as a combination of two smaller classical problems. The Knapsack Problem, where an optimal subset of customers must be selected to be served taking into account the gain as well as the cost of serving a customer, and the Vehicle Routing Problem, according to which we must design an optimal fleet schedule that visits all customers without violating an upper cost bound.

The proposed algorithm is a Local Search development enhanced by a Deep Q Learning procedure. At the beginning of the algorithm execution, a neural network is trained to determine the optimal selection of a single operator among the available ones at each step aiming at the maximum objective increase after a large number of iterations. These operators constitute easy-to-implement low-level heuristics that perform simple operations such as removal or addition of customers to the solution. In addition, a couple of intense methods are introduced with a view of exploring the solution space in a

deeper level. The first performs multiple customer insertions and deletions by tackling a suitably defined Mixed Integer Programming model. The second method reduces the cost of every individual route of the solution by solving the Traveling Salesman Problem using a Lin-Kernighan based heuristic algorithm.

With respect to the computational results, it was shown that the algorithm did not manage to completely outperform the existing methodologies and produced worse scores for the majority of instances. Nonetheless, the few instances for which improvement was noted all included a larger number of vehicles than the rest. This element leads us to the conclusion that our algorithm can offer relatively better results for a certain type of instances and thus has the potential to contribute to the CTOP literature.

In terms of future work directions, it could be worth experimenting with the application of DRL methodologies in non-deterministic problems. In such cases, a suitably trained agent could possibly discover patterns and obtain a level of knowledge of the problem that would be opaque to an OR algorithm due to the lack of complete information. An example of a relevant problem would be the Dynamic Vehicle Routing Problem (DVRP) that consists in designing an optimal set of routes to satisfy a set of customers whose availability follows a general rule p. This means that we cannot visit a certain customer unless a condition p is met, which could be, for instance, that a customer becomes available only if a neighboring customer has been visited. In such a problem, the DRL agent could potentially explore the solution space and retrieve information that was not known in prior and can help a certain algorithm achieve better results. In general, DRL is a promising field that has the potential to offer at least competitive performance in comparison with the conventional OR techniques. During the last years a significant amount of research focus can be observed on similar methodologies, therefore it is expected that the relevant literature will advance and be enriched over the next years.

References

Archetti C., Feillet D., Hertz A., Speranza M. G. The capacitated team orienteering and profitable tour problems // Journal of the Operational Research Society. 2009. 60. 831–842.

Arnau Quim, Juan Angel, Serra Isabel. On the Use of Learnheuristics in Vehicle Routing Optimization Problems with Dynamic Inputs // Algorithms. XII 2018. 11, 12. 208.

Arulkumaran Kai, Deisenroth Marc Peter, Brundage Miles, Bharath Anil Anthony. Deep Reinforcement Learning: A Brief Survey // IEEE Signal Processing Magazine. XI 2017. 34, 6. 26–38.

Barto Andrew G., Sutton Richard S., Anderson Charles W. Neuronlike adaptive elements that can solve difficult learning control problems // IEEE Transactions on Systems, Man, and Cybernetics. IX 1983. SMC-13, 5. 834–846.

Bello Irwan, Pham Hieu, Le Quoc V., Norouzi Mohammad, Bengio Samy. Neural Combinatorial Optimization with Reinforcement Learning // CoRR. 2016. abs/1611.09940.

Ben-Said Asma, El-Hajj Racha, Moukrim Aziz. A variable space search heuristic for the Capacitated Team Orienteering Problem // Journal of Heuristics. 4 2019. 25. 273–303.

Burke Edmund K., Hyde Matthew, Kendall Graham, Ochoa Gabriela, Özcan Ender, Woodward John R. A Classification of Hyper-heuristic Approaches // International Series in Operations Research &amp Management Science. 2010. 449–468.

Campbell Ann M., Gendreau Michel, Thomas Barrett W. The orienteering problem with stochastic travel and service times // Annals of Operations Research. V 2011. 186, 1. 61–81.

Chao I-Ming, Golden Bruce L., Wasil Edward A. The team orienteering problem // European Journal of Operational Research. II 1996. 88, 3. 464–474.

Charikar Moses S. Similarity estimation techniques from rounding algorithms // Proceedings of the thiry-fourth annual ACM symposium on Theory of computing - STOC '02. 2002.

Cowling Peter, Kendall Graham, Soubeiga Eric. A Hyperheuristic Approach to Scheduling a Sales Summit // Lecture Notes in Computer Science. 2001. 176–190.

Dang Duc-Cuong, El-Hajj Racha, Moukrim Aziz. A Branch-and-Cut Algorithm for Solving the Team Orienteering Problem // Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems. 2013. 332–339.

Eberhart R., Kennedy J. A new optimizer using particle swarm theory // MHS'95. Proceedings of the Sixth International Symposium on Micro Machine and Human Science. 1995.

Garrido Pablo, Riff María Cristina. DVRP: a hard dynamic combinatorial optimisation problem tackled by an evolutionary hyper-heuristic // Journal of Heuristics. II 2010. 16, 6. 795–834.

Gomez Faustino, Schmidhuber Jürgen. Evolving Modular Fast-Weight Networks for Control // Lecture Notes in Computer Science. 2005. 383–389.

Hasselt Hado. Double Q-learning // Advances in Neural Information Processing Systems. 23. 2010.

Heess Nicolas, Wayne Greg, Silver David, Lillicrap Timothy P., Tassa Yuval, Erez Tom. Learning Continuous Control Policies by Stochastic Value Gradients // CoRR. 2015. abs/1510.09142.

Helsgaun Keld. An effective implementation of the Lin–Kernighan traveling salesman heuristic // European Journal of Operational Research. X 2000. 126, 1. 106–130.

Keshtkaran Morteza, Ziarati Koorush, Bettinelli Andrea, Vigo Daniele. Enhanced exact solution methods for the Team Orienteering Problem // International Journal of Production Research. VII 2015. 54, 2. 591–601.

Khalil Elias, Dai Hanjun, Zhang Yuyu, Dilkina Bistra, Song Le. Learning Combinatorial Optimization Algorithms over Graphs // Advances in Neural Information Processing Systems. 30. 2017.

Kool Wouter, Hoof Herke van, Welling Max. Attention, Learn to Solve Routing Problems! // International Conference on Learning Representations. 2019.

Koutník Jan, Cuccu Giuseppe, Schmidhuber Jürgen, Gomez Faustino. Evolving large-scale neural networks for vision-based reinforcement learning // Proceeding of the fifteenth annual conference on Genetic and evolutionary computation conference - GECCO '13. 2013.

Labadie Nacima, Mansini Renata, Melechovský Jan, Calvo Roberto Wolfler. The Team Orienteering Problem with Time Windows: An LP-based Granular Variable Neighborhood Search // European Journal of Operational Research. VII 2012. 220, 1. 15–27.

Liang Yun-Chia, Kulturel-Konak Sadan, Lo Min-Hua. A multiple-level variable neighborhood search approach to the orienteering problem // Journal of Industrial and Production Engineering. VI 2013. 30, 4. 238–247.

Lillicrap Timothy P., Hunt Jonathan J., Pritzel Alexander, Heess Nicolas, Erez Tom, Tassa Yuval, Silver David, Wierstra Daan. Continuous control with deep reinforcement learning. 2015.

Reinforcement learning for robots using neural networks. // . 1992.

Lin S., Kernighan B. An Effective Heuristic Algorithm for the Traveling-Salesman Problem // Oper. Res. 1973. 21. 498–516.

Lon Rinde R.S. van, Holvoet Tom, Berghe Greet Vanden, Wenseleers Tom, Branke Juergen. Evolutionary synthesis of multi-agent systems for dynamic dial-a-ride problems // Proceedings of the fourteenth international conference on Genetic and evolutionary computation conference companion - GECCO Companion '12. 2012.

Mısır Mustafa, Verbeeck Katja, Causmaecker Patrick De, Berghe Greet Vanden. An Intelligent Hyper-Heuristic Framework for CHeSC 2011 // Lecture Notes in Computer Science. 2012. 461–466.

Mnih Volodymyr, Kavukcuoglu Koray, Silver David, Rusu Andrei A., Veness Joel, Bellemare Marc G., Graves Alex, Riedmiller Martin, Fidjeland Andreas K., Ostrovski Georg, Petersen Stig, Beattie Charles, Sadik Amir, Antonoglou Ioannis, King Helen, Kumaran Dharshan, Wierstra Daan, Legg Shane, Hassabis Demis. Human-level control through deep reinforcement learning // Nature. 2 2015. 518. 529–533.

Nazari MohammadReza, Oroojlooy Afshin, Snyder Lawrence, Takac Martin. Reinforcement Learning for Solving the Vehicle Routing Problem // Advances in Neural Information Processing Systems. 31. 2018.

How can we define intrinsic motivation ? // . 07 2013.

Papapanagiotou V., Montemanni Roberto. Objective function evaluation methods for the orienteering problem with stochastic travel and service times // Journal of Applied Operational Research. 01 2014. 6. 16–29.

Qin Wei, Zhuang Zilong, Huang Zizhao, Huang Haozhe. A novel reinforcement learning-based hyper-heuristic for heterogeneous vehicle routing problem // Computers &amp Industrial Engineering. VI 2021. 156. 107252.

Runka Andrew. Evolving an edge selection formula for ant colony optimization // Proceedings of the 11th Annual conference on Genetic and evolutionary computation - GECCO '09. 2009.

Schulman John, Wolski Filip, Dhariwal Prafulla, Radford Alec, Klimov Oleg. Proximal Policy Optimization Algorithms // CoRR. 2017. abs/1707.06347.

Sevkli Zulal, Sevilgen F. Erdogan. Discrete Particle Swarm Optimization for the Orienteering Problem // IEEE Congress on Evolutionary Computation. VII 2010.

Sutton Richard S, Barto Andrew G. Reinforcement learning: An introduction. 2018.

Tang Haoran, Houthooft Rein, Foote Davis, Stooke Adam, Chen Xi, Duan Yan, Schulman John, Turck Filip De, Abbeel Pieter. #Exploration: A Study of Count-Based Exploration for Deep Reinforcement Learning // CoRR. 2016. abs/1611.04717.

Tarantilis C. D., Stavropoulou F., Repoussis P. P. The Capacitated Team Orienteering Problem: A Bi-level Filter-and-Fan method // European Journal of Operational Research. 1 2013. 224. 65–78.

Tsiligirides T. Heuristic methods applied to orienteering // Journal of the Operational Research Society. 1984. 35, 9. 797–809.

Vansteenwegen Pieter, Souffriau Wouter, Berghe Greet Vanden, Oudheusden Dirk Van. Iterated local search for the team orienteering problem with time windows // Computers &amp Operations Research. XII 2009. 36, 12. 3281–3290.

Vaswani Ashish, Shazeer Noam, Parmar Niki, Uszkoreit Jakob, Jones Llion, Gomez Aidan N, Kaiser Ł ukasz, Polosukhin Illia. Attention is All you Need // Advances in Neural Information Processing Systems. 30. 2017.

Wang Ziyu, Schaul Tom, Hessel Matteo, Hasselt Hado van, Lanctot Marc, Freitas Nando de. Dueling Network Architectures for Deep Reinforcement Learning. 2015.

Watkins Christopher J. C. H., Dayan Peter. Q-learning // Machine Learning. V 1992. 8, 3-4. 279–292.

Williams Ronald J. Simple statistical gradient-following algorithms for connectionist reinforcement learning // Machine Learning. V 1992. 8, 3-4. 229–256.

Wu Yaoxin, Song Wen, Cao Zhiguang, Zhang Jie, Lim Andrew. Learning Improvement Heuristics for Solving Routing Problems. 2019.

Appendix

Instance	Zbest	BiF&F-f			BiF&F-s			VSS-Tabu			VSS-SA			DQN-HH		
		Zmax	RPE	TTB	Zmax	RPE	TTB	Zmax	RPE	TTB	Zmax	RPE	TTB	Zmax	RPE	TTB
01-337-14-345-720	4172	4172	0.00	17	4172	0.00	17	4172	0.00	1	4172	0.00	2	4180	-0.19	502.065
02-385-16-350-713	4784	4784	0.00	25	4784	0.00	25	4784	0.00	1	4784	0.00	3	4792	-0.17	625.565
03-433-18-330-675	5201	5201	0.00	33	5201	0.00	32	5201	0.00	2	5201	0.00	3	5209	-0.15	704.52
04-481-20-335-713	5828	5828	0.00	48	5828	0.00	47	5828	0.00	1	5828	0.00	3	5842	-0.24	798.165
05-529-22-340-705	6445	6445	0.00	101	6445	0.00	98	6445	0.00	3	6445	0.00	5	6462	-0.26	1210.895
06-577-24-365-683	7071	7071	0.00	94	7071	0.00	93	7071	0.00	1	7071	0.00	2	7082	-0.16	1552.05
07-361-15-335-668	4355	4355	0.00	24	4355	0.00	23	4355	0.00	1	4355	0.00	3	4366	-0.25	476.5
08-433-18-350-675	5194	5194	0.00	51	5194	0.00	49	5194	0.00	2	5194	0.00	4	5200	-0.12	1040.14
09-505-21-360-660	6183	6183	0.00	103	6183	0.00	104	6183	0.00	3	6183	0.00	22	6196	-0.21	1076.43
10-577-24-375-675	7239	7239	0.00	144	7239	0.00	144	7239	0.00	4	7239	0.00	7	7252	-0.18	1564.145
01-337-6-75-100	460	459	0.22	10	460	0.00	30	460	0.00	16	460	0.00	17	457	0.65	452.7
02-385-6-75-100	547	543	0.73	22	543	0.73	35	547	0.00	29	547	0.00	44	539	1.46	433.05
03-433-6-75-100	599	598	0.17	148	599	0.00	127	599	0.00	313	599	0.00	115	593	1.00	448.95
04-481-6-75-100	535	524	2.06	2	528	1.31	78	535	0.00	55	535	0.00	32	534	0.19	498.15
05-529-6-75-100	493	493	0.00	11	493	0.00	11	493	0.00	54	493	0.00	15	490	0.61	519.9
06-577-6-75-100	592	588	0.68	215	592	0.00	494	592	0.00	31	592	0.00	132	583	1.52	548.4
07-361-6-75-100	546	546	0.00	8	546	0.00	8	546	0.00	3	546	0.00	3	546	0.00	468.6
08-433-6-75-100	530	527	0.57	16	527	0.57	25	530	0.00	42	529	0.19	30	527	0.57	516
09-505-6-75-100	531	528	0.56	15	530	0.19	30	531	0.00	19	531	0.00	23	530	0.19	576.6
10-577-6-75-100	574	570	0.70	4	570	0.70	4	574	0.00	3	574	0.00	6	569	0.87	551.25
01-337-7-75-100	513	512	0.19	16	513	0.00	85	513	0.00	6	513	0.00	8	501	2.34	532.5
02-385-7-75-100	626	624	0.32	27	623	0.48	171	626	0.00	99	626	0.00	47	615	1.76	519.75
03-433-7-75-100	690	689	0.14	76	689	0.14	337	690	0.00	159	690	0.00	133	683	1.01	589.5
04-481-7-75-100	613	598	2.45	79	599	2.28	93	613	0.00	70	613	0.00	64	603	1.63	581.85
05-529-7-75-100	560	560	0.00	19	560	0.00	19	560	0.00	66	560	0.00	32	554	1.07	576
06-577-7-75-100	680	680	0.00	402	680	0.00	1959	680	0.00	324	680	0.00	186	668	1.76	655.35
07-361-7-75-100	620	618	0.32	21	620	0.00	68	620	0.00	29	620	0.00	28	620	0.00	510.9
08-433-7-75-100	605	599	0.99	13	599	0.99	13	605	0.00	56	604	0.17	31	598	1.16	1060.5
09-505-7-75-100	603	603	0.00	40	603	0.00	144	603	0.00	53	603	0.00	31	599	0.66	606.15
10-577-7-75-100	652	650	0.31	9	650	0.31	9	652	0.00	82	652	0.00	52	645	1.07	635.1
01-337-8-75-100	566	557	1.59	20	566	0.00	95	566	0.00	28	566	0.00	26	552	2.47	574.65
02-385-8-75-100	701	694	1.00	7	694	1.00	7	701	0.00	79	699	0.29	37	691	1.43	609.6
03-433-8-75-100	780	775	0.64	1	777	0.38	1021	780	0.00	133	780	0.00	114	767	1.67	620.1

Instance	Zbest	BiF&F-f			BiF&F-s			VSS-Tabu			VSS-SA			DQN-HH		
		Zmax	RPE	TTB	Zmax	RPE	TTB	Zmax	RPE	TTB	Zmax	RPE	TTB	Zmax	RPE	TTB
04-481-8-75-100	687	673	2.04	19	685	0.29	308	687	0.00	198	687	0.00	488	677	1.46	657
05-529-8-75-100	623	623	0.00	227	622	0.16	303	623	0.00	99	623	0.00	30	616	1.12	629.4
06-577-8-75-100	768	765	0.39	106	767	0.13	257	768	0.00	362	767	0.13	223	754	1.82	726.6
07-361-8-75-100	689	683	0.87	12	689	0.00	99	689	0.00	47	689	0.00	22	687	0.29	1174.8
08-433-8-75-100	678	670	1.18	34	676	0.29	91	678	0.00	56	678	0.00	48	673	0.74	583.95
09-505-8-75-100	672	671	0.15	14	671	0.15	14	672	0.00	73	672	0.00	31	660	1.79	910.95
10-577-8-75-100	729	721	1.10	26	727	0.27	50	729	0.00	78	729	0.00	38	721	1.10	738.6
01-337-6-150-200	1107	1106	0.09	355	1104	0.27	1144	1107	0.00	600	1107	0.00	225	1085	1.99	662.4
02-385-6-150-200	1137	1125	1.06	1684	1129	0.70	3635	1137	0.00	793	1135	0.18	634	1122	1.32	674.7
03-433-6-150-200	1201	1197	0.33	912	1201	0.00	5868	1201	0.00	1242	1201	0.00	895	1174	2.25	704.7
04-481-6-150-200	1191	1186	0.42	22	1186	0.42	24	1191	0.00	1567	1191	0.00	620	1156	2.94	738.6
05-529-6-150-200	1171	1149	1.88	466	1149	1.88	2391	1171	0.00	1595	1169	0.17	793	1143	2.39	835.5
06-577-6-150-200	1233	1230	0.24	525	1230	0.24	4035	1233	0.00	1991	1233	0.00	699	1213	1.62	860.25
07-361-6-150-200	1144	1139	0.44	836	1141	0.26	486	1144	0.00	809	1143	0.09	268	1128	1.40	700.2
08-433-6-150-200	1131	1128	0.27	644	1122	0.80	1131	1131	0.00	1130	1131	0.00	503	1110	1.86	748.05
09-505-6-150-200	1106	1100	0.54	29	1102	0.36	1221	1106	0.00	1318	1105	0.09	843	1076	2.71	772.05
10-577-6-150-200	1198	1189	0.75	896	1191	0.58	4159	1198	0.00	1456	1196	0.17	930	1169	2.42	768.75
01-337-7-150-200	1267	1220	3.71	473	1221	3.63	456	1267	0.00	586	1262	0.39	316	1223	3.47	718.2
02-385-7-150-200	1299	1273	2.00	329	1290	0.69	5927	1299	0.00	1100	1298	0.08	370	1275	1.85	736.2
03-433-7-150-200	1383	1375	0.58	2192	1377	0.43	4301	1383	0.00	2034	1382	0.07	503	1340	3.11	839.55
04-481-7-150-200	1365	1357	0.59	506	1356	0.66	876	1365	0.00	1527	1365	0.00	697	1326	2.86	938.7
05-529-7-150-200	1348	1333	1.11	793	1344	0.30	10231	1348	0.00	1696	1346	0.15	581	1316	2.37	995.7
06-577-7-150-200	1421	1415	0.42	5602	1415	0.42	8870	1421	0.00	2415	1420	0.07	1119	1396	1.76	1056
07-361-7-150-200	1312	1277	2.67	55	1310	0.15	4280	1312	0.00	625	1312	0.00	323	1291	1.60	831.15
08-433-7-150-200	1295	1284	0.85	3147	1286	0.69	2201	1295	0.00	937	1294	0.08	585	1268	2.08	906
09-505-7-150-200	1267	1253	1.10	1853	1263	0.32	4602	1267	0.00	973	1265	0.16	653	1236	2.45	947.4
10-577-7-150-200	1377	1376	0.07	5514	1372	0.36	1221	1377	0.00	2355	1376	0.07	774	1350	1.96	1080.3
01-337-8-150-200	1404	1386	1.28	803	1389	1.07	1524	1404	0.00	830	1402	0.14	255	1357	3.35	888.75
02-385-8-150-200	1454	1436	1.24	563	1440	0.96	3271	1454	0.00	1073	1452	0.14	456	1424	2.06	1002.3
03-433-8-150-200	1557	1551	0.39	2622	1552	0.32	2752	1556	0.06	1672	1557	0.00	579	1510	3.02	951.6
04-481-8-150-200	1531	1524	0.46	294	1527	0.26	9275	1530	0.07	1455	1531	0.00	662	1487	2.87	1117.5
05-529-8-150-200	1514	1496	1.19	1063	1494	1.32	4557	1514	0.00	1363	1514	0.00	691	1478	2.38	1138.8
06-577-8-150-200	1607	1588	1.18	1277	1602	0.31	14037	1607	0.00	2963	1604	0.19	985	1569	2.36	2069.7

Instance	Zbest	BiF&F-f			BiF&F-s			VSS-Tabu			VSS-SA			DQN-HH		
		Zmax	RPE	TTB	Zmax	RPE	TTB	Zmax	RPE	TTB	Zmax	RPE	TTB	Zmax	RPE	TTB
07-361-8-150-200	1467	1450	1.16	296	1453	0.95	626	1465	0.14	828	1467	0.00	305	1439	1.91	917.7
08-433-8-150-200	1445	1427	1.25	5126	1436	0.62	2452	1445	0.00	1687	1443	0.14	479	1415	2.08	977.25
09-505-8-150-200	1421	1403	1.27	987	1413	0.56	7027	1421	0.00	2294	1418	0.21	626	1399	1.55	1132.05
10-577-8-150-200	1554	1543	0.71	1763	1548	0.39	4201	1554	0.00	2009	1554	0.00	721	1512	2.70	1292.7
01-337-6-200-400	1590	1584	0.38	54	1589	0.06	1773	1590	0.00	813	1590	0.00	613	1555	2.20	819.9
02-385-6-200-400	1603	1600	0.19	547	1599	0.25	2983	1601	0.12	1505	1603	0.00	1489	1563	2.50	928.35
03-433-6-200-400	1622	1621	0.06	233	1621	0.06	239	1622	0.00	2480	1622	0.00	1435	1576	2.84	1020
04-481-6-200-400	1642	1641	0.06	610	1642	0.00	6229	1641	0.06	2647	1642	0.00	1793	1608	2.07	1041.6
05-529-6-200-400	1656	1651	0.30	396	1651	0.30	3683	1656	0.00	4511	1655	0.06	1524	1616	2.42	1077.15
06-577-6-200-400	1686	1685	0.06	696	1686	0.00	1393	1686	0.00	6748	1686	0.00	5138	1654	1.90	1188.3
07-361-6-200-400	1614	1608	0.37	574	1612	0.12	2390	1614	0.00	848	1613	0.06	819	1572	2.60	946.95
08-433-6-200-400	1612	1610	0.12	724	1606	0.37	685	1612	0.00	2700	1612	0.00	3303	1560	3.23	1016.85
09-505-6-200-400	1622	1618	0.25	1923	1620	0.12	11069	1622	0.00	1990	1621	0.06	3474	1577	2.77	1062.9
10-577-6-200-400	1662	1661	0.06	6783	1662	0.00	12692	1662	0.00	6310	1662	0.00	3778	1618	2.65	1125
01-337-7-200-400	1820	1817	0.16	118	1817	0.16	120	1819	0.05	1677	1820	0.00	618	1780	2.20	1047.6
02-385-7-200-400	1840	1833	0.38	209	1837	0.16	4925	1839	0.05	991	1840	0.00	1465	1794	2.50	1157.25
03-433-7-200-400	1864	1864	0.00	456	1864	0.00	2600	1864	0.00	3805	1864	0.00	1241	1810	2.90	1248.3
04-481-7-200-400	1891	1890	0.05	6572	1889	0.11	4746	1890	0.05	2198	1891	0.00	1995	1844	2.49	1351.95
05-529-7-200-400	1913	1910	0.16	4424	1910	0.16	7882	1913	0.00	5123	1913	0.00	1671	1859	2.82	1212.75
06-577-7-200-400	1948	1947	0.05	2518	1947	0.05	8065	1947	0.05	4125	1948	0.00	2342	1915	1.69	1446.9
07-361-7-200-400	1844	1840	0.22	422	1843	0.05	3754	1844	0.00	808	1844	0.00	977	1796	2.60	1113.45
08-433-7-200-400	1850	1841	0.49	840	1848	0.11	10204	1848	0.11	1601	1850	0.00	1365	1786	3.46	1177.05
09-505-7-200-400	1873	1871	0.11	8573	1870	0.16	5774	1873	0.00	2662	1871	0.11	3101	1817	2.99	1868.55
10-577-7-200-400	1920	1919	0.05	4659	1919	0.05	1410	1919	0.05	6215	1920	0.00	3755	1869	2.66	1586.4
01-337-8-200-400	2041	2032	0.44	155	2032	0.44	1236	2039	0.10	1286	2041	0.00	763	1996	2.20	1313.1
02-385-8-200-400	2068	2064	0.19	1216	2065	0.15	5641	2066	0.10	2121	2068	0.00	1078	2020	2.32	1364.85
03-433-8-200-400	2098	2096	0.10	574	2097	0.05	3173	2097	0.05	2724	2098	0.00	1649	2037	2.91	1512.75
04-481-8-200-400	2131	2127	0.19	112	2127	0.19	103	2130	0.05	3317	2131	0.00	1627	2083	2.25	2416.8
05-529-8-200-400	2163	2156	0.32	1038	2155	0.37	7914	2162	0.05	3764	2163	0.00	2252	2085	3.61	1714.35
06-577-8-200-400	2205	2204	0.05	7385	2204	0.05	1658	2205	0.00	6426	2204	0.05	2289	2166	1.77	1843.5
07-361-8-200-400	2064	2063	0.05	1964	2063	0.05	3762	2063	0.05	889	2064	0.00	886	2012	2.52	1299.45
08-433-8-200-400	2074	2068	0.29	683	2070	0.19	3589	2072	0.10	1974	2074	0.00	2000	2008	3.18	1455
09-505-8-200-400	2118	2115	0.14	11022	2115	0.14	17380	2116	0.09	2693	2118	0.00	2515	2048	3.31	2425.8

Instance	Zbest	BiF&F-f			BiF&F-s			VSS-Tabu			VSS-SA			DQN-HH		
		Zmax	RPE	TTB	Zmax	RPE	TTB	Zmax	RPE	TTB	Zmax	RPE	TTB	Zmax	RPE	TTB
10-577-8-200-400	2173	2168	0.23	3876	2172	0.05	15678	2171	0.09	3326	2173	0.00	3197	2106	3.08	1715.55
01-337-6-345-720	2530	2530	0.00	136	2530	0.00	136	2530	0.00	178	2530	0.00	122	2495	1.38	1869.15
02-385-6-350-713	2614	2614	0.00	533	2614	0.00	752	2613	0.04	2703	2613	0.04	897	2581	1.26	2191.65
03-433-6-330-675	2520	2520	0.00	320	2520	0.00	319	2520	0.00	6320	2520	0.00	2246	2470	1.98	2103.45
04-481-6-335-713	2603	2603	0.00	993	2603	0.00	1844	2603	0.00	2755	2603	0.00	6341	2534	2.65	2276.1
05-529-6-340-705	2696	2696	0.00	2317	2696	0.00	6956	2696	0.00	5675	2696	0.00	6799	2603	3.45	2218.35
06-577-6-365-683	2932	2932	0.00	1798	2932	0.00	7050	2931	0.03	16945	2931	0.03	7305	2872	2.05	2619.6
07-361-6-335-668	2489	2489	0.00	183	2489	0.00	187	2489	0.00	2097	2489	0.00	262	2453	1.45	2065.2
08-433-6-350-675	2623	2621	0.08	8268	2621	0.08	11461	2622	0.04	3121	2623	0.00	3846	2557	2.52	2057.1
09-505-6-360-660	2782	2775	0.25	5661	2775	0.25	11315	2780	0.07	12326	2782	0.00	5697	2686	3.45	2699.85
10-577-6-375-675	2963	2960	0.10	6664	2959	0.13	10222	2962	0.03	8460	2963	0.00	10388	2870	3.14	2854.65
01-337-7-345-720	2860	2859	0.03	150	2859	0.03	153	2860	0.00	2235	2860	0.00	1518	2813	1.64	2510.85
02-385-7-350-713	2965	2964	0.03	226	2965	0.00	652	2965	0.00	2853	2965	0.00	1946	2923	1.42	2875.95
03-433-7-330-675	2860	2860	0.00	283	2860	0.00	270	2860	0.00	2530	2860	0.00	2845	2818	1.47	2605.95
04-481-7-335-713	2954	2954	0.00	555	2954	0.00	556	2954	0.00	4768	2954	0.00	1572	2902	1.76	2720.4
05-529-7-340-705	3075	3074	0.03	2349	3075	0.00	6881	3074	0.03	7201	3074	0.03	6259	2985	2.93	2932.05
06-577-7-365-683	3353	3351	0.06	1840	3353	0.00	7411	3351	0.06	11492	3352	0.03	10810	3279	2.21	2901.9
07-361-7-335-668	2814	2812	0.07	849	2814	0.00	3306	2812	0.07	2429	2813	0.04	2760	2764	1.78	2104.8
08-433-7-350-675	2973	2970	0.10	916	2971	0.07	1425	2972	0.03	5321	2973	0.00	2772	2898	2.52	2650.2
09-505-7-360-660	3163	3157	0.19	13277	3163	0.00	13864	3162	0.03	8331	3163	0.00	6937	3057	3.35	3193.05
10-577-7-375-675	3383	3380	0.09	4890	3381	0.06	14379	3383	0.00	11735	3383	0.00	8549	3291	2.72	3426.6
01-337-8-345-720	3159	3159	0.00	927	3159	0.00	1745	3158	0.03	1462	3159	0.00	1576	3123	1.14	3224.7
02-385-8-350-713	3291	3290	0.03	144	3291	0.00	583	3291	0.00	2850	3291	0.00	3018	3239	1.58	3576
03-433-8-330-675	3190	3190	0.00	455	3190	0.00	461	3190	0.00	4288	3190	0.00	3143	3139	1.60	3382.5
04-481-8-335-713	3289	3289	0.00	451	3289	0.00	426	3289	0.00	283	3289	0.00	314	3233	1.70	3469.35
05-529-8-340-705	3434	3432	0.06	1953	3434	0.00	7418	3434	0.00	4145	3434	0.00	4396	3344	2.62	3517.5
06-577-8-365-683	3749	3749	0.00	997	3749	0.00	1007	3748	0.03	14090	3748	0.03	9342	3721	0.75	2994.1875
07-361-8-335-668	3117	3116	0.03	187	3116	0.03	189	3117	0.00	3090	3117	0.00	2248	3058	1.89	2658.9
08-433-8-350-675	3303	3301	0.06	1834	3302	0.03	3455	3303	0.00	5691	3303	0.00	5120	3235	2.06	2331
09-505-8-360-660	3529	3510	0.54	2282	3525	0.11	14499	3528	0.03	7370	3529	0.00	11119	3418	3.15	5495.85
10-577-8-375-675	3783	3781	0.05	2784	3783	0.00	11663	3781	0.05	10154	3783	0.00	9865	3685	2.59	4392
Average			0.43	1365.11		0.27	3205.64		0.02	2360		0.03	1695		1.87	1403.67125