A RNN-Based Hyper-Heuristic for Combinatorial Problems

Emmanuel Kieffer[1], Gabriel Duflo[2], Grégoire Danoy[1,2], Sébastien Varrette[1], and Pascal Bouvry[1,2]

[1] Faculty of Science, Technology and Medicine, University of Luxembourg, Esch-sur-Alzette, Luxembourg
[2] Interdisciplinary Centre for Security, Reliability and Trust, University of Luxembourg, Esch-sur-Alzette, Luxembourg
`firstname.lastname@uni.lu`

Abstract. Designing efficient heuristics is a laborious and tedious task that generally requires a full understanding and knowledge of a given optimization problem. Hyper-heuristics have been mainly introduced to tackle this issue and are mostly relying on Genetic Programming and its variants. Many attempts in the literature have shown that an automatic training mechanism for heuristic learning is possible and can challenge human-based heuristics in terms of gap to optimality. In this work, we introduce a novel approach based on a recent work on Deep Symbolic Regression. We demonstrate that scoring functions can be trained using Recurrent Neural Networks to tackle a well-know combinatorial problem, i.e., the Multi-dimensional Knapsack. Experiments have been conducted on instances from the OR-Library and results show that the proposed modus operandi is an alternative and promising approach to human-based heuristics and classical heuristic generation approaches.

Keywords: Deep Symbolic Regression · Multi-dimensional Knapsack · Hyper-heuristics.

1 Introduction

Real-word combinatorial problems are typically $\mathcal{NP}$-hard and of large size, making them intractable with exact approaches from the Operations Research literature. Numerous non-exact approaches (e.g., heuristics, metaheuristics) have thus been proposed to provide solutions in polynomial time. Nonetheless, designing heuristics remains a difficult exercise requiring a lot of trials and can be difficult to generalize to large scale instances. The lack of guarantees can also be prohibitive for some decision makers. Hyper-heuristics have been therefore designed as a methodology assisting solution designers in creating heuristics using Evolutionary Learning. Similarly to what is done in machine learning to build classifiers or regressive models, one can "learn to optimize" a specific problem by training a constructive model on a large set of instances. Some successful attempts in the literature (e.g. [14], [2], [4]) relied on constructive hyper-heuristics.

The latter can be considered as a meta-algorithm that permits to engineer automatically heuristics using an existing set of instances. Hyper-heuristics essentially search through the space of heuristics or heuristic components instead of the space of solutions and use specific instances' data and properties (e.g., objective coefficients, columns for a mathematical problem) as inputs for the design of heuristics. As described in the next sections, hyper-heuristics have been historically applied to select existing heuristics and combine them together. Despite their very good results, they are constrained by the existing knowledge of a problem. This means that for a new problem with few existing heuristics, the chance to produce an efficient hyper-heuristic remains low as the search space is very restricted. On the contrary, constructive hyper-heuristic approaches assemble heuristics' components through evolution and have the advantage to spawn unseen ones. The limitations encountered by the original selective hyper-heuristics are thus removed with this constructive version. Historically, only Genetic Programming (GP) algorithms and their variants have been considered as constructive hyper-heuristics. In this work, we propose to generate heuristics and more precisely scoring functions based on a recent advance in Deep Symbolic Regression (DSR) [33]. Indeed, authors considered a Recurrent Neural Network (RNN) trained with Reinforcement Learning to provide probability distributions over symbolic expressions with the aim of solving regression problems. Learning symbolic expressions offers multiple advantages such as readability, interpretability and trustworthiness. The authors also note that the recent advances in Deep Neural Networks underexplore this aspect. We here propose to investigate the potential of this new approach to learn symbolic expressions as novel scoring functions for the Multi-dimensional Knapsack Problem (MKP). The latter has been widely study, hence our interest for it. We compare a GP-based hyper-heuristic against a RNN trained to solve MKP instances from the OR-library [12]. We demonstrate that the scoring functions obtained after training produce competitive results with state-of-the-art approaches and outperform the GP-based hyper-heuristic reference for the Multi-dimensional Knapsack.
The remainder of this article is organized as follows. The related work section details the classification of hyper-heuristics existing in the recent literature as well as the latest advances of symbolic regression. Section 3 introduces the MKP as well as some other resolution approaches. Section 4 introduces the proposed Deep Hyper-heuristic (DHH) which is subsequently compared to the reference GP-based hyper-heuristic on the MKP. Then, experimental setup and results are discussed in section 5 and 6 respectively. Finally, the last section provides our conclusions and proposes some possible perspectives.

2 Related work

Described as "heuristics to choose heuristics" by Cowling et al. [13], hyper-heuristics refer to approaches combining artificial intelligence methods to design heuristics. Contrary to algorithms searching in the space of solutions, hyper-heuristic algorithms search in the space of algorithms, i.e. heuristics, in order to determine the best heuristics combination to solve a problem. Burke et al. in [8] compared hyper-heuristics as "*off-the-peg*" methods which are generic approaches providing solutions of acceptable quality as opposed to *"made-to-measure"* techniques. This need of generalization is clearly related to machine learning approaches. Therefore, hyper-heuristics can be classified as learning algorithms and have been motivated by the following factors: the difficulty of maintaining problem-specific algorithms and the need of automating the design of algorithms. Two methodologies of hyper-heuristics rose from the literature: the first one is referred to as *heuristic selection* while the second one is described as *heuristic generation*.

Heuristic selection is the "legacy" approach which involves determining the best subset of heuristics solving a problem. Among these approaches, we can distinguish constructive and perturbation methods. Constructive methods assemble a solution step by step, starting from a partial or empty solution. The construction of a full solution is achieved through the selection and application of a heuristic to this partial solution. For this purpose, a pre-existing set of heuristics should be provided in order to determine the best heuristics to apply at a given state of the search. The resolution then stops when the solution is complete. Constructive methods have been applied for instance on vehicle routing [23], 2D packing [28], constraint satisfaction [32] and scheduling [22]. On the contrary, perturbation methods start from a valid solution and attempt to modify it using a pre-existing set of perturbation heuristics. At each step, one heuristic is selected from this set and applied to the solution. According to a specific acceptance strategy factor (e.g., deterministic or non-deterministic), the new solution is accepted or rejected. It is also possible to perturb multiple solutions at once but it has been seldom used in the literature. Scheduling [25], space allocation [5] and packing [6] are problems where such perturbation methods have been exploited.

More recently, a growing interest has been devoted to heuristic generation. The motivation behind this approach is the automatic generation mechanism which does not rely on a possible set of pre-existing heuristics. Instead of searching in the space of heuristics, the hyper-heuristic searches in the space of components, i.e., instance data. Building a complete heuristic is not a trivial task but it has been performed using Genetic Programming. In contrast to Genetic Algorithms (GA) where solution vectors are improved via genetic operators, GP algorithms evolve a population of programs until a certain stopping criterion is satisfied. Programs are expressed as tree structures which means that their length is not defined a priori contrary to GAs. The suitability of GP algorithms to produce heuristics has been outlined in a survey by Burke et al. [9]. The major advantage brought by GP algorithms is the possibility to automatize the assembly of building blocks, i.e. terminal sets and function sets emerging from knowledge gained on a problem. Concerning the MKP, this knowledge can be easily retrieved from the literature. Additionally, the dynamic length of the tree encoding is an advantage if some size limitations are implemented. Indeed, large programs will tend to have over-fitting symptoms meaning that the generated heuristics will be very efficient on the training instances but not on new ones. These are typically the same issues faced by machine learning models. GP-based hyper-heuristics encountered real successes in cutting and packing [10], function optimization [31] and other additional domains [18], [40], [30]. In addition, it is worth mentioning the recent approaches such as Cartesian GP and Grammar-based GP algorithms which are improvements of classical GP algorithms. Cartesian GP algorithms is an alternative form of GP algorithms encoding a graph representation of a computer program. Cartesian GP defines explicitly a size preventing bloat but can be very sensitive to parameters. In Grammar-based GP algorithms [7], a grammar in Backus-Naur Form (BNF) is considered to map linear genotypes to phenotype trees and have less structural difficulties than a classical GP algorithms.

Contrary to [33] which only considers regression problems, we extend the field of application of Deep Recurrent Networks to the task of learning scoring functions for a combinatorial problem such as the MKP. Nonetheless, it is worth mentioning recent advances in symbolic regression using deep neural networks although there are few attempts in the literature. For instance, a neural network implementation has been investigated in [39] as a pre-processing approach before using symbolic regression. In [36], authors have explored symbolic operators as activation functions while keeping neural networks differentiable. In [26], variational encoders have been considered for the first time to encode and decode parse trees using predefined grammar rules. Finally, [33] recently proposed a RNN to generate probability distributions over symbolic expressions. Authors have relied on pre-order traversal to build abstract syntax tree representing equations. They also illustrate and compare their approach with other frameworks and outperformed most of them. This is the reason why we rely on this last contribution to build a new type of hyper-heuristic. One should also note that the aforementioned works only investigated small scale regression problems and their suitability still needs to be demonstrated for large-scale problems. In this work, we partially answer this last question when extending the approach to hyper-heuristics.

3 Multi-dimensional 0-1 Knapsack

The Multi-dimensional 0-1 Knapsack (MKP) is a $\mathcal{NP}$-hard combinatorial problem which extends the well-know 0-1 Knapsack problem for multiple sacks. The objective is to find a subset of items maximizing the total profit and fitting into the m sacks. Each item j gives a profit p_j and occupies some space a_{ij} in the sack i. Each sack i has a maximum capacity b_i which should not be exceeded. The Multi-dimensional 0-1 Knapsack can be formally expressed as a 0-1 Integer Linear Program (ILP) as illustrated by Program 1.

More practically, the MKP is a resource allocation problem. It received a wide attention from many communities, including the Operations Research and Evolutionary Computing ones. Multiple heuristics and metaheuristics have been designed to tackle the MKP in addition to the existing exact approaches which

$$\text{maximize} \quad \sum_{j=1}^{n} p_j x_j \tag{1}$$

$$\text{subject to} \quad \sum_{j=1}^{n} a_{ij} x_j \leq b_i \quad \forall i \in \{1, ..., m\} \tag{2}$$

$$x_j \in \{0, 1\} \qquad \forall j \in \{1, ..., n\} \tag{3}$$

Program. 1. 0-1 ILP for the multi-dimensional knapsack

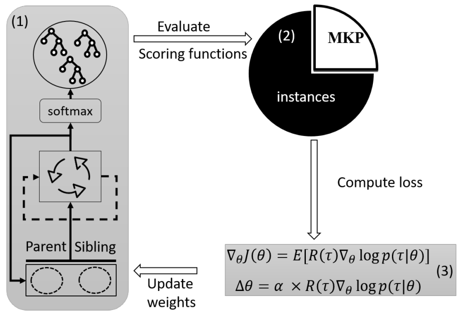
can only handle small instances. Among existing heuristics for the MKP, *greedy* heuristics are designed to be fast, i.e., work in polynomial time. Generally, these are constructive methods that can be categorized as primal or dual heuristics. A primal heuristic starts from a feasible solution and tries to improve the objective value while keeping the solution feasible. On the contrary, a dual heuristic starts from an upper-bound solution (in case of maximization), i.e., not feasible, and attempts to make it feasible while minimizing the impact on the objective value. For example, [37] considered a dual heuristic with a starting solution taking all items. Then, the heuristic removed items according to an increasing ratio until feasibility was reached. The ratio or score of each item j is computed as follows: $r_j = \frac{p_j}{\sum_i^m w_i a_{ij}}$. Weights w_i are sometimes omitted since they add a new level of complexity and are specific to the considered instances. Using Lagrangian relaxation, [29] improves the dual heuristic of [37].

Concerning primal heuristics, items are added as long as all constraints remain satisfied. In this case, items with the largest ratios have priority. These new heuristics using dual multipliers give insights about the variables to fix. Further improvements based on bound tightness [21], threshold acceptance [17] and noising approaches [11] have contributed to improve such heuristics. The interested reader may refer to the survey on MKP heuristics by Fréville [20].

Metaheuristics have also been considered to solve MKP instances. These are stochastic algorithms which successfully tackled many combinatorial optimization problems, including the MKP. A simulated annealing algorithm has been first employed in [16] where specific random moves should maintain feasibility during the search. Many diverse metaheuristics have been then considered to solve the MKP during the last decade including Ant Colony Optimization [19], Genetic algorithms [27], Memetic algorithms [12], Particle Swarm algorithms [24], Fish Swarm algorithms [3] and Bee Colony algorithms [38].

4 A RNN-based Hyper-heuristic

Contrary to most GP-based hyper-heuristics building full syntax trees and applying evolutionary operators to generate new ones, we propose hereafter to consider RNNs for this task. Based on the RNN architecture proposed in [33], we posit that Deep Recurrent Networks for symbolic expressions are perfectly suitable to learn *scoring functions* for a combinatorial problem such as the MKP. A scoring function takes as inputs information about an item to be added to the sacks. This function measures the pertinence of the given item j which is represented by the profit p_j and the column of the constraint matrix $A_{.j}$.



(1)
softmax
Parent
Sibling
Evaluate
Scoring functions
(2)
MKP
instances
Compute loss
$\nabla_\theta J(\theta) = E[R(\tau)\nabla_\theta \log p(\tau|\theta)]$
$\Delta\theta = \alpha \times R(\tau)\nabla_\theta \log p(\tau|\theta)$
(3)
Update
weights

Fig. 2. Workflow of the RNN-based Hyper-Heuristic (RHH): sampling (1), evaluation (2) and training (3).

Although the DSR approach proposed in [33] has the unique purpose to tackle symbolic regression problems, we propose to extend it in order to create a RNN-based Hyper-heuristic (RHH) generating sequences of tokens described in Table 1. A sequence can then be decoded into a symbolic expression represented as a binary tree, i.e., each node can only have at most one sibling. The sequence of tokens is produced by the RNN in a autoregressive manner, i.e., k^{th} token prediction depends on the previously obtained tokens. RNNs model efficiently time-varying information such as sequences. The RHH implementation presented hereafter provides a one-to-one mapping between the sampled symbolic expressions and the resulting scoring functions to solve MKP instances. This implementation can be decomposed into 3 main iterative steps as depicted in Fig. 2. The first one (1) is the sampling step in which a batch of symbolic expressions is produced. The second step (2) turns symbolic expressions into scoring functions which are subsequently evaluated using a greedy heuristic template to generate rewards. Finally, the last step (3) performs a one-step gradient ascent to update the embedded RNN's weights using Policy Gradient. All these steps are described in details in the following sections.

4.1 Sampling symbolic expressions (1)

Let us define T, the set of tokens which can be sampled. This set is equivalent to the union of terminal and non-terminal sets defined in GP algorithms. Table 1 provides a description of all tokens considered to generate scoring functions for the Multi-dimensional Knapsack. Among these tokens, we selected the entire column describing an item, the average difference between the capacity and the average resource consumption for item j, the maximal resources consumption and the total resource consumption of item j. In order to discriminate **good** items from **bad** ones, we also add as prior knowledge the solution of the LP relaxation which is clearly a very good feature for our learning purpose. The set shown in Table 1 is not exhaustive and could be easily completed with new features to discriminate more accurately profitable items from valueless ones.

Table 1. The set of tokens T which are components of scoring functions

Name	**Description**
Operators	
+	Add two inputs
-	Substract two inputs
*	Multiply two inputs
%	Divide two inputs with protection
Terminal sets/ Arguments	
p_j	Profit of the current item j
$d_j = \frac{\sum_i b_i - a_{ij}}{m}$	Average difference between the capacity and the resource consumption for item j
a_{1j}	Resource consumption of item j for sack 1
a_{2j}	Resource consumption of item j for sack 1
$a_{\ldots,j}$	...
a_{mj}	Resource consumption of item j for sack m
$s_j = \sum_i a_{ij}$	Total resource consumption of item j for sack i
$m_j = \max_i a_{ij}$	Max resource consumption of item j for sack i
$\bar{x}_j$	Solution value for item j after LP relaxation

The following describes how scoring functions are created from sampled symbolic expressions using a Recurrent Neural Network (RNN) as illustrated in Fig. 3. The first inputs provided to the networks are necessarily empty since the tree representation of the future symbolic expression is empty (see **step 0** in Fig. 3). Therefore, an empty token $< E >$ is added to the set of tokens listed in Table 1. At each step i, the next token τ_i^s is sampled according to the pre-order traversal. Its future location in the tree is thus known which means that the parent and sibling nodes can be identified and provided as inputs to the RNN cell. This is illustrated in Fig. 3 where the left-child of the root node will host the next token. Its parent is obviously the root node and it has no sibling yet. The RNN returns a logit vector L_i which is passed to a softmax layer. This very common layer permits to obtain a discrete probability distribution over all tokens at step i, i.e., $\sum_{k=1}^{|T|} p(\tau_i^k|\tau_{i-1}^s;\theta) = 1$. The sampled token τ_i^s ($\bar{x}_j$ in step 1) is then added to the tree. Please note that sampled tokens τ_i^s and τ_{i-1}^s are not necessarily connected in the resulting tree. In fact, they can be far from each other as depicted in Fig. 3. This is due to the hierarchical structure induced by the traversal. As a consequence, a sample token at step $i-1$ is not necessarily provided as input for step i. This is illustrated by **step 5** in Fig. 3. "Teacher forcing" is therefore considered to replace inputs by the true parent and sibling with regards to the traversal. Finally when the expression is complete, i.e., no empty leaves, one can define the log-likehood of the sample sequence of tokens τ, i.e., sampled symbolic expression, as follows: $\log p(\tau|\theta) = \sum_{i=1}^{|\tau|} \log p(\tau_i^s|\tau_{i-1}^s;\theta)$.

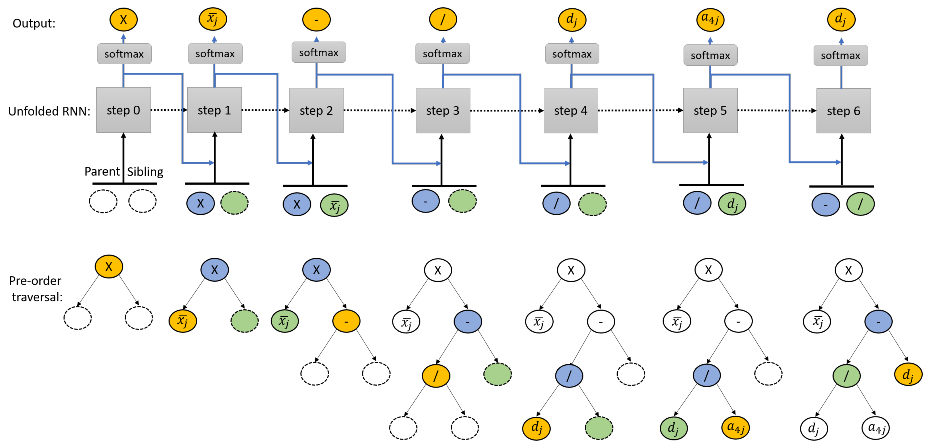


Fig. 3. Example of scoring function: $\bar{x}_j(\frac{d_j}{a_j^4} - d_j)$ generated from a sampled sequence of tokens (length=7). Blue circles represent parent inputs while green ones stand for sibling. Outputs are illustrated by yellow circles and are obtained after applying a softmax layer and sampling with regards to a discrete probability distribution over all possible tokens. Dashed circles represent missing token positions still need to be filled. At each step, parent and sibling tokens are presented to the RNN cell. The tree representation grows according to the pre-order traversal (recursively traverse left subtree first)

4.2 Evaluation of the resulting scoring functions (2)

A sampled symbolic expression can then be turned into a scoring function applied inside a so-called heuristic template (see Algorithm 1) in order to evaluate its relevance to provide an efficient insertion order of items into the sacks. The combination of this template and a scoring function characterizes a heuristic which is subsequently applied on a set of multiple MKP instances I, i.e., a training set. In this work, we consider a primal heuristic template starting from a feasible and trivial solution and selecting items ranked using a generated scoring function until sacks are full. Ranking is obtained by applying the scoring function on item data.

4.3 RNN training and gradient update (3)

In order to train the RNN to produce efficient scoring functions, a reward/fitness for the sequence of tokens $R(\tau)$ is proposed as follows: $R(\tau) = \frac{1}{1+\sum_{j=1}^{|I|} v_\tau^j}$ with

Algorithm 1 greedy_heuristic(instance,function)

```
1: value ← 0
2: solution ← [0,0,...,0]
3: sacks ← [0,0,...,0]
4: indexes ← sort(items,function)
5: while $indexes \neq \emptyset$ do
6:     index ← indexes.pop_head()
7:     if $sacks[i] + instance.A[i, index] \leq instance.rhs[i] \; \forall i \in \{1, ..., m\}$ then
8:         $solution[index] \leftarrow 1$
9:         $value \leftarrow value + instance.p[index]$
10:        for $i \in \{1, ..., m\}$ do
11:            $sacks[i] \leftarrow sacks[i] + instance.A[i, index]$
12:        end for
13:    end if
14: end while
15: return value
```

v_τ^j the solution value obtained by solving the j^{th} instance of I with the scoring function generated from the sequence of tokens τ.
The RNN is trained on batches of N sampled expressions $\mathcal{B} = \{\tau^{(k)}\}_{k=1}^{N}$ using **Policy Gradients** $\nabla_\theta J(\theta) = E_\tau \left[R(\tau)\nabla_\theta \log p(\tau|\theta)\right]$, . The loss function $\mathcal{L}(\theta)$ is therefore expressed in this way:

$$\mathcal{L}(\theta) = \frac{1}{N}\sum_{k=1}^{N} R(\tau^{(k)}) \log p(\tau^{(k)}|\theta) \tag{4}$$

Although solving MKP instances is deterministic, the spread of rewards can be very large. Consequently, the policy gradient will have high-variance. One way to mitigate this problem is to substract a baseline to rewards by some constant value, which is normally the mean of the rewards. The RNN training steps are described hereafter:

1. Initialise the network with random weights
2. Generate a batch $\mathcal{B}$ of N sampled expressions
3. Expressions are turned into scoring functions and injected into the heuristic template (see algorithm 1)
4. Heuristics are applied on a set $\mathcal{I}$ of training instances and rewards $R(\tau^k)$ are computed based on the resolution of instances.
5. The $\mathcal{L}(\theta) = \frac{1}{N}\sum_{k=1}^{N} R(\tau^{(k)}) \log p(\tau^{(k)}|\theta)$ is computed according to rewards and the log-likelihood
6. Perform a gradient ascent step update of the network's weights, maximizing the policy gradient
7. Repeat 2. until stopping criterion is met

The next section defines experimental setup to assess the proposed hyper-heuristic, namely RHH.

5 Experimental setup

The MKP instances from the OR-Library [3] have been considered. These have been originally introduced by [12] and include 270 instances classified according to the number of variables(items) $n \in \{100, 250, 500\}$, number of constraints $m \in \{5, 10, 30\}$ and tightness ratio $r \in \{0.25, 0.50, 0.75\}$.

In order to evaluate the efficiency of the trained heuristics, i.e., the combionation of the heuristic template and trained scoring functions, on these instances, we adopt as performance measure the %-gap (see equation 5) between a lower bound and an upper bound. Lower bounds are provided by the heuristics, i.e., $\underline{v}^h$, while the continuous LP relaxation, i.e., $\overline{v}_{lp}$, will be the reference upper bound. In addition, we multiply all gaps by 100.

$$\%\text{-}gap = 100 * \frac{\overline{v}_{lp} - \underline{v}^h}{\overline{v}_{lp}} \tag{5}$$

The RNN-based Hyper-heuristic (RHH) will be compared to a GP-based hyper-heuristic described in [14]. Table 2 details all the GP parameters and GP operators used by these authors. They performed 50 generations with a population size of 10000 scoring functions. Contrary to GAs, GP algorithms make a different use of the evolutionary operators. First of all, their probabilities should sum to 1. For example, in the case of Table 2, 85% of the solutions will mate with another one, 10% will face mutations and only 5% will be kept without any modifications for the next generation. In order to keep control of the size of each syntax tree, they prevent trees from having a depth greater than 17 nodes. The interested reader can refer to [14] for more details on the GP implementation.

Table 2. GP parameters

Generations	50
Population size	10000
Crossover Probability (CXPB)	0.85
Mutation Probability (MUTPB)	0.1
Reproduction Probability	0.05
Tree initialization method	Ramped half-and-half
Selection Method	Tournament selection with size=7
Depth limitation	17
Crossover Operator	One point crossover
Mutation Operator	Grow

Finally both approaches, i.e., RHH and the GP-based hyper-heuristic, follow the same protocol to train scoring functions. All instances have been divided into groups depending on the number of variables, the number of constraints

[3] http://people.brunel.ac.uk/ mastjjb/jeb/orlib/mknapinfo.html

and the tightness ratio. Both hyper-heuristics have been applied on all groups which contain ten instances each. Five random instances have been selected as training instances while the remaining five instances have been considered as validation instances. The reported %-gaps are only computed on the validation instances. For each group, five runs have been performed in order to obtain an average %-gap and the best scoring function has been recorded.

Table 3 lists all hyperparameters describing the RNN. A single layer with 32 units has been considered. A maximum of 500000 sampled expressions are generated to provide fair comparisons with the GP-based hyper-heuristic. The RNN is trained on batches of 1000 sampled expressions. The Adam optimiser is set up with a learning rate of 0.0005 and has been selected to optimise the RNN's weights.

Table 3. RNN parameters

Max sample	500000
Batch size	1000
RNN cell type	LSTM
Number of layers	1
Number of units	32
Optimizer	Adam
Learning rate	0.0005
Hidden state initializer	zeros

Experiments have been conducted on the High Performance Computing (HPC) platform of the University of Luxembourg. Each run was completed on a single core of an Intel Xeon E5-2680 v3 @ 2.5 GHz, 32Gb of RAM server, which was dedicated to this task.

6 Experimental results

The average %-gap obtained after 5 runs is provided in Table 4. The left part of this table represents the results reported in [14] while the right part corresponds to the RNN-based Hyper-Heuristic approach (RHH) proposed in this work.

Each row depicts a specific instance set **ORnXm** divided into groups of different tightness ratios. For example, the average %-gap obtained for the instance set **OR5x100** with tightness ratio 0.25 is **4.98** for the GP-based Hyper-heuristic approach. Gray shaded cells indicate that the average %-gap is better for the considered approach. For example, the average %-gap obtained for the instance set **OR5x100** with tightness ratio 0.25 is reported better for the RHH approach.

Table 4 shows us that each instance set **ORnXm** has a better average %-gap when solved with the RHH approach. When considering tightness ratios, we can observe that RHH outperforms all instances with $r = 0.25$ and $r = 0.5$ while this is not the case for $r = 0.75$. The tightness ratio defines the scarcity of capacities. The closer to 0 the tightness ratio the more constrained the instance. Indeed, a ratio $r = 0.25$ implies that about 25% of the items can be packed contrary to a ratio $r = 0.75$ where 75% of the items can be packed. These results show that the proposed RHH approach is able to handle more efficiently different levels of tightness.

Table 4. Average gaps (%) of the best found heuristics on the ORlib instances ordered by tightness ratio

	Original approach				**RHH**			
Instance set	0.25	0.50	0.75	Average	0.25	0.50	0.75	Average
OR5x100	4.98	2.05	1.36	***2.80***	1.58	2.02	2.33	***1.98***
OR5x250	3.08	1.66	0.77	***1.84***	0.49	0.47	0.85	***0.60***
OR5x500	2.38	1.64	0.71	***1.58***	0.26	0.22	0.51	***0.33***
OR10x100	7.39	3.54	2.26	***4.40***	2.30	2.34	3.38	***2.67***
OR10x250	4.43	2.78	1.15	***2.79***	1.01	0.66	1.25	***0.98***
OR10x500	3.77	1.97	0.99	***2.24***	0.38	0.35	0.57	***0.43***
OR30x100	8.67	4.70	2.43	***5.27***	3.41	2.06	6.01	***3.83***
OR30x250	5.73	3.25	1.70	***3.56***	1.33	1.20	2.34	***1.62***
OR30x500	4.80	2.54	1.40	***2.91***	0.80	0.56	0.95	***0.77***
All instances	***5.03***	***2.68***	***1.42***	***3.04***	***1.29***	***1.10***	***2.02***	***1.47***

The best scoring functions obtained with RHH are listed in Table 5. One can notice the presence of $\bar{x}_j$, i.e., the solution of the LP relaxation for variable x_j, in all resulting scoring functions. Interestingly, multiple scoring functions do not include the profit p_j (e.g., OR30x500-0.75). The size of each scoring function is rather reasonable and no "bloating" effect, generally experienced with GP algorithms, can be observed.

Table 5. Best scoring functions obtained for each benchmark

	Scoring funtions		
Instance set	0.25	0.50	0.75
OR5x100	$\bar{x}_j - (d_j + s_j) * \bar{x}_j^2 / p_j$	$((-d_j * s_j * \bar{x}_j + 1)/s_j - a_{1j})/(a_{3j} + m_j)$	$((a_{1j} + d_j)/a_{2j} - d_j) * \bar{x}_j$
OR5x250	$\bar{x}_j/(a_{1j} * \bar{x}_j - s_j)$	$\bar{x}_j/(-(a_{5j} + d_j)/m_j - a_{1j} - s_j)$	$-(-(a_{3j} - m_j * \bar{x}_j)/a_{1j} + d_j - s_j) * \bar{x}_j + d_j$
OR5x500	$-p_j - d_j * \bar{x}_j^2 + d_j$	$-(p_j + s_j * \bar{x}_j^2 - \bar{x}_j) * \bar{x}_j + a_{1j}$	$(-(d_j + d_j/p_j) * \bar{x}_j + a_{5j} + s_j + \bar{x}_j)/(s_j^2 + \bar{x}_j)$
OR10x100	$(a_{1j} - s_j) * \bar{x}_j$	$(-(p_j + a_{2j})) * \bar{x}_j + a_{2j}) * \bar{x}_j$	$((p_j + a_{9j})/s_j) - \bar{x}_j$
OR10x250	$-\bar{x}_j + 1/a_{1j}$	$(a_{10j}/s_j) - \bar{x}_j$	$(a_{2j} - a_{4j} - s_j)) * \bar{x}_j^2$
OR10x500	$(p_j - s_j) * \bar{x}_j^2$	$-p_j * d_j * \bar{x}_j + a_{4j}$	$\bar{x}_j/(-(a_{2j} + s_j) * d_j + a_{4j} + 2a_{5j})$
OR30x100	$p_j * \bar{x}_j/(-s_j + \bar{x}_j)$	$\bar{x}_j/(-a_{27j} - d_j + \bar{x}_j)$	$-p_j * \bar{x}_j + d_j$
OR30x250	$a_{12j} - s_j * \bar{x}_j$	$(-p_j + \bar{x}_j) * \bar{x}_j$	$p_j * (a_{24j} + a_{29j} - s_j) * \bar{x}_j$
OR30x500	$p_j * \bar{x}_j/(-d_j + s_j)$	$\bar{x}_j/(a_{7j} - s_j)$	$\bar{x}_j/(-d_j + s_j)$

Last but not least, Table 6 presents the %-gap obtained by different existing methods from the literature on the same benchmarks. The approach proposed in this work, i.e., RHH, obtains a good rank, i.e., 6th position which demonstrates the suitability of using recurrent neural architectures to assist in building heuristics.

Table 6. Comparisons with multiple existing approaches over all ORlib instances in terms of gap (%)

Type	Reference	%-gap
MIP	[15] (CPLEX 12.2)	0.52
MA	[12]	0.54
Selection HH	[15]	0.70
MA	[42]	0.92
Heuristic	[34]	1.37
RHH	**this work**	**1.47**
Heuristic	[21]	1.91
Metaheuristic	[35]	2.28
GHH	[14]	3.04
MIP	[12](CPLEX 4.0)	3.14
Heuristic	[1]	3.46
Heuristic	[41]	6.98
Heuristic	[29]	7.69

Hyper-heuristics or automatic generation of heuristics are not dedicated to provide the best results. They are general approaches which have been proposed to facilitate the generation of **good performing** and **fast** algorithms to solve problems. Table 6 shows that despite their general approach, they can provide better results than dedicated algorithms.

7 Conclusion and Perspectives

Traditionally, hyper-heuristics are GP-based approaches evolving heuristics represented by abstract syntax trees. In this paper, we proposed a new hyper-heuristic model based on deep symbolic expressions to automatically solve combinatorial problems such as the Multidimensional Knapsack. We tackled the generation of scoring functions to measure the pertinence of adding an item to the sacks. These functions therefore allow finding an inserting order in the sacks and provide reasonable educated guesses to solve the MKP. Contrary to the classical knapsack with a single constraint, it is not trivial to manually discover an efficient scoring procedure for multi-dimensional variants of the knapsack. After detailing the methodology, we compared a state-of-the-art GP hyper-heuristic versus the new deep hyper-heuristic approach. To measure the performance of both approaches and fairly confront them, validation instances which have been not presented to both hyper-heuristics during training served to compute a performance measure, i.e., %-gap. Results show that the proposed methodology relying on this recurrent neural architecture outperforms the classical GP-based hyper-heuristic on this problem. Training symbolic expressions with deep learning has the benefit to provide efficient predictions while keeping scoring functions explainable. Symbolic expressions can be easily analysed by experts contrary to a network providing only black-box scoring values which would be difficult to interpret. Future works will attempt to apply the proposed approach to "Column Generation (CG)", a well-known Operation Research technique, to solve large-scale problems. This would notably be helpful to cope with degeneracy.

References

1. Akçay, Y., Li, H., Xu, S.H.: Greedy algorithm for the general multidimensional knapsack problem. Ann Oper Res **150**(1), 17–29 (dec 2006). https://doi.org/10.1007/s10479-006-0150-4
2. Allen, S., Burke, E.K., Hyde, M., Kendall, G.: Evolving reusable 3d packing heuristics with genetic programming. In: Proceedings of the 11th Annual conference on Genetic and evolutionary computation - GECCO '09. ACM Press (2009). https://doi.org/10.1145/1569901.1570029
3. Azad, M.A.K., Rocha, A.M.A.C., Fernandes, E.M.G.P.: Solving large 0–1 multidimensional knapsack problems by a new simplified binary artificial fish swarm algorithm. J Math Model Algor **14**(3), 313–330 (feb 2015). https://doi.org/10.1007/s10852-015-9275-2
4. Bader-El-Den, M., Poli, R.: Generating SAT local-search heuristics using a GP hyper-heuristic framework. In: Lecture Notes in Computer Science, pp. 37–49. Springer Berlin Heidelberg (2007). https://doi.org/10.1007/978-3-540-79305-2_4
5. Bai, R., Burke, E.K., Kendall, G.: Heuristic, meta-heuristic and hyper-heuristic approaches for fresh produce inventory control and shelf space allocation. Journal of the Operational Research Society **59**(10), 1387–1397 (oct 2008). https://doi.org/10.1057/palgrave.jors.2602463
6. Bai, R., Blazewicz, J., Burke, E.K., Kendall, G., McCollum, B.: A simulated annealing hyper-heuristic methodology for flexible decision support. 4OR-Q J Oper Res **10**(1), 43–66 (nov 2011). https://doi.org/10.1007/s10288-011-0182-8
7. Brabazon, A., O'Neill, M., McGarraghy, S.: Grammar-based and developmental genetic programming. In: Natural Computing Algorithms, pp. 345–356. Springer Berlin Heidelberg (2015). https://doi.org/10.1007/978-3-662-43631-8_18
8. Burke, E.K., Gendreau, M., Hyde, M., Kendall, G., Ochoa, G., Özcan, E., Qu, R.: Hyper-heuristics: a survey of the state of the art. Journal of the Operational Research Society **64**(12), 1695–1724 (dec 2013). https://doi.org/10.1057/jors.2013.71
9. Burke, E.K., Hyde, M.R., Kendall, G., Ochoa, G., Ozcan, E., Woodward, J.R.: Exploring hyper-heuristic methodologies with genetic programming. In: Intelligent Systems Reference Library, pp. 177–201. Springer Berlin Heidelberg (2009). https://doi.org/10.1007/978-3-642-01799-5_6
10. Burke, E.K., Hyde, M.R., Kendall, G.: Grammatical evolution of local search heuristics. IEEE Transactions on Evolutionary Computation **16**(3), 406–417 (2012). https://doi.org/10.1109/tevc.2011.2160401
11. Charon, I., Hudry, O.: The noising method: a new method for combinatorial optimization. Operations Research Letters **14**(3), 133–137 (oct 1993). https://doi.org/10.1016/0167-6377(93)90023-a
12. Chu, P., Beasley, J.: A genetic algorithm for the multidimensional knapsack problem. Journal of Heuristics **4**(1), 63–86 (1998)
13. Cowling, P., Kendall, G., Soubeiga, E.: A hyperheuristic approach to scheduling a sales summit. In: Lecture Notes in Computer Science, pp. 176–190. Springer Berlin Heidelberg (2001). https://doi.org/10.1007/3-540-44629-x_11
14. Drake, J.H., Hyde, M., Khaled, I., Özcan, E.: A genetic programming hyper-heuristic for the multidimensional knapsack problem. Kybernetes **43**(9/10), 1500–1511 (March 2014). https://doi.org/10.1108/k-09-2013-0201

15. Drake, J.H., Özcan, E., Burke, E.K.: A case study of controlling crossover in a selection hyper-heuristic framework using the multidimensional knapsack problem. Evol. Comput. **24**(1), 113–141 (Mar 2016). https://doi.org/10.1162/evco_a_00145
16. Drexl, A.: A simulated annealing approach to the multiconstraint zero-one knapsack problem. Computing **40**(1), 1–8 (mar 1988). https://doi.org/10.1007/bf02242185
17. Dueck, G., Scheuer, T.: Threshold accepting: A general purpose optimization algorithm appearing superior to simulated annealing. Journal of Computational Physics **90**(1), 161 – 175 (1990). https://doi.org/10.1016/0021-9991(90)90201-b
18. Elyasaf, A., Hauptman, A., Sipper, M.: Evolutionary design of Free-Cell solvers. IEEE Trans. Comput. Intell. AI Games: Transactions on Computational Intelligence and AI in Games **4**(4), 270–281 (dec 2012). https://doi.org/10.1109/tciaig.2012.2210423
19. Fingler, H., Cáceres, E.N., Mongelli, H., Song, S.W.: A CUDA based solution to the multidimensional knapsack problem using the ant colony optimization. Procedia Computer Science **29**, 84–94 (2014). https://doi.org/10.1016/j.procs.2014.05.008
20. Fréville, A.: The multidimensional 0–1 knapsack problem: An overview. European Journal of Operational Research **155**(1), 1–21 (may 2004). https://doi.org/10.1016/s0377-2217(03)00274-1
21. Freville, A., Plateau, G.: An efficient preprocessing procedure for the multidimensional 0-1 knapsack problem. Discrete Applied Mathematics **49**(1-3), 189–212 (1994). https://doi.org/10.1016/0166-218x(94)90209-7
22. García-Villoria, A., Salhi, S., Corominas, A., Pastor, R.: Hyper-heuristic approaches for the response time variability problem. European Journal of Operational Research **211**(1), 160–169 (may 2011). https://doi.org/10.1016/j.ejor.2010.12.005
23. Garrido, P., Riff, M.C.: DVRP: a hard dynamic combinatorial optimisation problem tackled by an evolutionary hyper-heuristic. J Heuristics **16**(6), 795–834 (feb 2010). https://doi.org/10.1007/s10732-010-9126-2
24. Hembecker, F., Lopes, H.S., Jr, W.G.: Particle Swarm Optimization for the Multidimensional Knapsack Problem. In: Adaptive and Natural Computing Algorithms (ICANNGA 2007). pp. 358–365. Springer Berlin Heidelberg (2007)
25. Kendall, G.: Scheduling english football fixtures over holiday periods. Journal of the Operational Research Society **59**(6), 743–755 (jun 2008). https://doi.org/10.1057/palgrave.jors.2602382
26. Kusner, M.J., Paige, B., Hernández-Lobato, J.M.: Grammar variational autoencoder. In: Precup, D., Teh, Y.W. (eds.) Proceedings of the 34th International Conference on Machine Learning. Proceedings of Machine Learning Research, vol. 70, pp. 1945–1954. PMLR (06–11 Aug 2017)
27. Lai, G., Yuan, D., Yang, S.: A new hybrid combinatorial genetic algorithm for multidimensional knapsack problems. J Supercomput **70**(2), 930–945 (jul 2014). https://doi.org/10.1007/s11227-014-1268-9
28. López-Camacho, E., Terashima-Marín, H., Ross, P., Valenzuela-Rendón, M.: Problem-state representations in a hyper-heuristic approach for the 2d irregular BPP. In: Proceedings of the 12th annual conference on Genetic and evolutionary computation - GECCO '10. ACM Press (2010). https://doi.org/10.1145/1830483.1830539
29. Magazine, M., Oguz, O.: A heuristic algorithm for the multidimensional zero-one knapsack problem. European Journal of Operational Research **16**(3), 319–326 (jun 1984). https://doi.org/10.1016/0377-2217(84)90286-8

30. Nguyen, S., Zhang, M., Johnston, M.: A genetic programming based hyper-heuristic approach for combinatorial optimisation. In: Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation. pp. 1299–1306. GECCO '11, ACM, New York, NY, USA (2011). https://doi.org/10.1145/2001576.2001752
31. Oltean, M.: Evolving evolutionary algorithms using linear genetic programming. Evol. Comput. **13**(3), 387–410 (Sep 2005). https://doi.org/10.1162/1063656054794815
32. Ortiz-Bayliss, J.C., Ozcan, E., Parkes, A.J., Terashima-Marin, H.: Mapping the performance of heuristics for constraint satisfaction. In: IEEE Congress on Evolutionary Computation. IEEE (jul 2010). https://doi.org/10.1109/cec.2010.5585965
33. Petersen, B.K.: Deep symbolic regression: Recovering mathematical expressions from data via policy gradients. CoRR **abs/1912.04871** (2019), http://arxiv.org/abs/1912.04871
34. Pirkul, H.: A heuristic solution procedure for the multiconstraint zero-one knapsack problem. Naval Research Logistics **34**(2), 161–172 (apr 1987). https://doi.org/10.1002/1520-6750(198704)34:2¡161::aid-nav3220340203¿3.0.co;2-a
35. Qian, F., Ding, R.: Simulated annealing for the 0/1 multidimensional knapsack problem. Numerical Mathematics-English Series- **16**(10201026), 1–7 (2007)
36. Sahoo, S., Lampert, C., Martius, G.: Learning equations for extrapolation and control. In: Dy, J., Krause, A. (eds.) Proceedings of the 35th International Conference on Machine Learning. Proceedings of Machine Learning Research, vol. 80, pp. 4442–4450. PMLR (10–15 Jul 2018)
37. Senju, S., Toyoda, Y.: An approach to linear programming with 0–1 variables. Management Science **15**(4), B–196–B–207 (dec 1968). https://doi.org/10.1287/mnsc.15.4.b196
38. Sundar, S., Singh, A., Rossi, A.: An artificial bee colony algorithm for the 0–1 multidimensional knapsack problem. In: Contemporary Computing: Third International Conference, IC3 2010, Noida, India, August 9-11, 2010. Proceedings, Part I, pp. 141–151. Springer Berlin Heidelberg, Berlin, Heidelberg (2010). https://doi.org/10.1007/978-3-642-14834-7_14
39. Udrescu, S.M., Tegmark, M.: AI feynman: A physics-inspired method for symbolic regression. Sci. Adv. **6**(16) (apr 2020). https://doi.org/10.1126/sciadv.aay2631
40. Van Lon, R.R., Holvoet, T., Vanden Berghe, G., Wenseleers, T., Branke, J.: Evolutionary synthesis of multi-agent systems for dynamic dial-a-ride problems. In: Proceedings of the 14th Annual Conference Companion on Genetic and Evolutionary Computation. pp. 331–336. GECCO '12, ACM, New York, NY, USA (2012). https://doi.org/10.1145/2330784.2330832
41. Volgenant, A., Zoon, J.A.: An Improved Heuristic for Multidimensional 0-1 Knapsack Problems. Journal of the Operational Research Society **41**(10), 963–970 (1990). https://doi.org/10.2307/2583274, http://www.palgrave-journals.com/doifinder/10.1057/jors.1990.148
42. Özcan, E., Başaran, C.: A case study of memetic algorithms for constraint optimization. Soft Comput **13**(8-9), 871–882 (jul 2008). https://doi.org/10.1007/s00500-008-0354-4