An Efficient Combinatorial Optimization Model Using Learning-to-Rank Distillation

Honguk Woo¹*  , Hyunsung Lee²†, Sangwoo Cho¹†

¹Department of Computer Science and Engineering, Sungkyunkwan University
²Kakao Corporation
hwoo@skku.edu, iggy.ll@kakaocorp.com, jsw7460@skku.edu

Abstract

Recently, deep reinforcement learning (RL) has proven its feasibility in solving combinatorial optimization problems (COPs). The learning-to-rank techniques have been studied in the field of information retrieval. While several COPs can be formulated as the prioritization of input items, as is common in the information retrieval, it has not been fully explored how the learning-to-rank techniques can be incorporated into deep RL for COPs. In this paper, we present the learning-to-rank distillation-based COP framework, where a high-performance ranking policy obtained by RL for a COP can be distilled into a non-iterative, simple model, thereby achieving a low-latency COP solver. Specifically, we employ the approximated ranking distillation to render a score-based ranking model learnable via gradient descent. Furthermore, we use the efficient sequence sampling to improve the inference performance with a limited delay. With the framework, we demonstrate that a distilled model not only achieves comparable performance to its respective, high-performance RL, but also provides several times faster inferences. We evaluate the framework with several COPs such as priority-based task scheduling and multidimensional knapsack, demonstrating the benefits of the framework in terms of inference latency and performance.

Introduction

In the field of computer science, it is considered challenging to tackle combinatorial optimization problems (COPs) that are computationally intractable. While numerous heuristic approaches have been studied to provide polynomial-time solutions, they often require in-depth knowledge on problem-specific features and customization upon the changes of problem conditions. Furthermore, several heuristic approaches such as branching (Chu and Beasley 1998) and tabu-search (Glover 1989) to solving COPs explore combinatorial search spaces extensively, and thus render themselves limited in large scale problems.

Recently, deep learning techniques have proven their feasibility in addressing COPs, e.g., routing optimization (Kool, van Hoof, and Welling 2019), task scheduling (Lee et al. 2020), and knapsack problem (Gu and Hao 2018). For deep learning-based COP approaches, it is challenging to build a training dataset with optimal labels, because many COPs are computationally infeasible to find exact solutions. Reinforcement learning (RL) is considered viable for such problems as neural architecture search (Zoph and Le 2017), device placement (Mirhoseini et al. 2017), games (Silver et al. 2017) where collecting supervised labels is expensive or infeasible.

As the RL action space of COPs can be intractably large (e.g., 100! possible solutions for ranking 100-items), it is undesirable to use a single probability distribution on the whole action space. Thus, a sequential structure, in which a probability distribution of an item to be selected next is iteratively calculated to represent a one-step action, becomes a feasible mechanism to establish RL-based COP solvers, as have been recently studied in (Bello et al. 2017; Vinyals et al. 2019). The sequential structure is effective to produce a permutation comparable to optimal solutions, but it often suffers from long inference time due to its iterative nature. Therefore, it is not suitable to apply these approaches to the field of mission critical applications with strict service level objectives and time constraints. For example, task placement in SoC devices necessitates fast inferences in a few milliseconds, but the inferences by a complex model with sequential processing often take a few seconds, so it is rarely feasible to employ deep learning-based task placement in SoC (Ykman-Couvreur et al. 2006; Shojaei et al. 2009).

In this paper, we present RLRD, an RL-to-rank distillation framework to address COPs, which enables the low-latency inference in online system environments. To do so, we develop a novel ranking distillation method and focus on two COPs where each problem instance can be treated as establishing the optimal policy about ranking items or making priority orders. Specifically, we employ a differentiable relaxation scheme for sorting and ranking operations (Blondel et al. 2020) to expedite direct optimization of ranking objectives. It is combined with a problem-specific objective to formulate a distillation loss that corrects the rankings of input items, thus enabling the robust distillation of the ranking policy from sequential RL to a non-iterative, score-based ranking model. Furthermore, we explore the efficient sampling technique with Gumbel trick (Jang, Gu, and Poole 2017; Kool, van Hoof, and Welling 2020) on the scores generated by the distilled model to expedite the generation of
Through experiments, we demonstrate that a distilled model by the framework achieves the low-latency inference while maintaining comparable performance to its teacher. For example, compared to the high-performance teacher model with sequential RL, the distilled model makes inferences up to 65 times and 3.8 times faster, respectively. The Gumbel trick-based sequence sampling improves the performance of distilled models (e.g., 2% for the knapsack and task scheduling problems, while it shows only about 2.6% and 1.0% degradation in performance, respectively. The Gumbel trick-based sequence sampling improves the performance of distilled models (e.g., 2% for the knapsack) efficiently with relatively small inference delay.

The contributions of this paper are summarized as follows.

- We present the learning-based efficient COP framework RLRD that can solve various COPs, in which a low-latency COP model is enabled by the differentiable ranking (DiffRank)-based distillation and it can be boosted by Gumbel trick-based efficient sequence sampling.

- We test the framework with well-known COPs in system areas such as priority-based task scheduling in real-time systems and multidimensional knapsack in resource management, demonstrating the robustness of our framework approach to various problem conditions.

Our Approach

In this section, we describe the overall structure of the RLRD framework with two building blocks, the deep RL-based COP model structure, and the ranking distillation procedure.

Framework Structure

In general, retraining or fine-tuning is needed to adapt a deep learning-based COP model for varying conditions on problem requirements. The RLRD framework supports such model adaptation through knowledge distillation. As shown in Figure 1, (1) for a COP, a learning-to-rank (teacher) policy in the encoder-decoder model is first trained by sequential RL, and (2) it is then transferred through the DiffRank-based distillation to a student model with non-iterative ranking operations, according to a given deployment configuration, e.g., requirements on low-latency inference or model size. For instance, a scheduling policy is established by RL to make inferences on the priority order of a set of tasks running on a real-time multiprocessor platform, and then it is distilled into a low-latency model to make same inferences with some stringent delay requirement.

Reinforcement Learning-to-Rank

Here, we describe the encoder-decoder structure of our RL-to-Rank model for COPs, and explain how to train it. Our teacher model is based on a widely adopted attentive structure (Kool, van Hoof, and Welling 2019). In our model representation, we consider parameters $\theta$ (e.g., $W_{\theta}x + b_{\theta}$ for Affine transformation of vector $x$), and we often omit them for simplicity. In an RL-to-Rank model, an encoder takes the features of $N$-items as input, producing the embeddings for the $N$-items, and a decoder conducts ranking decisions iteratively on the embeddings, yielding a permutation for the $N$-items. This encoder-decoder model is end-to-end trained by RL.

COP Encoder. In the encoder, each item $x_i \in \mathbb{R}^d$ containing $d$ features is first converted into vector $h_i^{(0)}$ through the simple Affine transformation, $h_i^{(0)} = W x_i + b$. Then, for $N$-items, $(h \times N)$-matrix, $H^{(0)} = [h_1^{(0)}, \ldots, h_N^{(0)}]$ is passed into the $L$-attention layers, where each attention layer consists of a Multi-Head Attention layer (MHA) and a Feed Forward network (FF). Each sub-layer is computed with skip connection and Batch Normalization (BN). For $l \in \{1, \cdots, L\}$, $H^{(l)}$ are updated by

$$H^{(l)} = \text{BN}^{(l)}(X + \text{FF}^{(l)}(X)),$$

$$X = \text{BN}^{(l)}(H^{(l-1)} + \text{MHA}(H^{(l-1)}))$$

where

$$\text{MHA}(X) = W_G(\text{AM}_1(X) \odot \cdots \odot \text{AM}_{d_h}(X)),$$

$\odot$ is the concatenation of tensors, $d_h$ is a fixed positive integer, and $W_G$ is a learnable parameter. $\text{AM}$ is given by

$$\text{AM}_f(X) = W_V(X)\text{Softmax}\left(\frac{1}{d_h}(W_K(X)^T W_Q(X))\right)$$

where $W_Q, W_K$ and $W_V$ denote the layer-wise parameters for query, key and value (Vaswani et al. 2017). The result output $H^{(L)}$ in (1) is the embedding for the input $N$-items, which are used as input to a decoder in the following.

Ranking Decoder. With the embeddings $H^{(L)}$ for $N$-items from the encoder, the decoder sequentially selects items to obtain an $N$-sized permutation $\phi = [\phi_1, \phi_2, \ldots, \phi_N]$ where distinct integers $\phi_i \in \{1, 2, \ldots, N\}$
correspond to the indices of the $N$-items. That is, item $\phi_1$ is selected first, so it is assigned the highest ranking (priority), and item $\phi_2$ is assigned the second, and so on. Specifically, the decoder establishes a function to rank $N$-items stochastically,

$$
\mathbb{P}(\phi|H^{(L)}) = \prod_{t=2}^{N} \mathbb{P}(\phi_t|\phi_1, \ldots, \phi_{t-1}, H^{(L)})
$$

(4)

where $\mathbb{P}(\phi_t|\phi_1, \ldots, \phi_{t-1}, H^{(L)})$ represents a probability that item $\phi_t$ is assigned the $t$th rank.

From an RL formulation perspective, in (4), the information about $N$-items ($H^{(L)}$) including ranking-assigned items $[\phi_1, \ldots, \phi_{t-1}]$ until $t$ corresponds to state $o_t$, and selecting $\phi_t$ corresponds to action $a_t$. That is, a state contains a partial solution over all permutations and an action is a one-step inference to determine a next ranked item. Accordingly, the stochastic ranking function above can be rewritten as $\theta$-parameterized policy $\pi_\theta$ for each timestep $t$.

$$
\pi_\theta(o_{t+1} = \phi | o_t) = \mathbb{P}(a_t \mid o_t) = \mathbb{P}(\phi_t|\phi_1, \ldots, \phi_{t-1}, H^{(L)})
$$

(5)

This policy is learned based on problem-specific reward signals. To establish such policy $\pi_\theta$ via RL, we formulate a learnable score function of item $x_i$ upon state $o_t$, which is used to estimate $\mathbb{P}(a_t = i \mid o_t)$, e.g.,

$$
\text{SCORE}_{H^{(L)}(t-1)}(x_i) = \begin{cases} 10 \ast \tanh \left( \text{Att}(x_{(g)}(t^{(g)}), H^{(L)}) \right) & \text{if } i \notin \{\phi_1, \ldots, \phi_{t-1}\} \\ -\infty & \text{otherwise} \end{cases}
$$

(6)

where $t^{(g)}$ is the embedding of an item $(x_{\phi_{t-1}})$ selected at timestep $t-1$, and $x^{(g)}$ is a global vector obtained by

$$
x_{(g)} = t^{(t-1)} \odot \text{Att}(t^{(t-1)}, H^{(L)}).
$$

(7)

Note that $t^0$ is randomly initialized. To incorporate the alignment between $t^{(t-1)}$ and $H^{(L)}$ in $x_{(g)}$, we use Attention (Vaswani et al. 2017),

$$
\text{Att}(q, Y) = \sum_{i=1}^{N} \alpha_i y_i, \quad \alpha_i = \text{Softmax}(w_1, \ldots, w_N),
$$

(8)

$$
w_i = V_A \tanh W_A [y_i \odot q]
$$

for query $q$ and vectors $Y = [y_1, \ldots, y_N]$, where $V_A$ and $W_A$ are learnable parameters. Finally, we have the policy $\pi_\theta$ that calculates the ranking probability that the $i$th item is selected next upon state $o_t$.

$$
\pi_\theta(o_{t+1} = i) = \mathbb{P}(a_t = i \mid o_t) = \frac{e^{\text{SCORE}(x_i)}}{\sum_{k=1}^{N} e^{\text{SCORE}(x_k)}}.
$$

(9)

Training. For end-to-end training the encoder-decoder, we use the REINFORCE algorithm (Williams 1992), which is effective for episodic tasks, e.g., problems formulated as ranking $N$-items. Suppose that for a problem of $N$-items, we obtain an episode with

$$
T^{(\theta)} = (s_1, a_1, r_1, \ldots, s_N, a_N, r_N)
$$

(10)

that are acquired by policy $\pi_\theta$, where $s$, $a$ and $r$ are state, action and reward samples. We set the goal of model training to maximize the expected total reward by $\pi_\theta$,

$$
J(\theta) = \mathbb{E}_{\pi_\theta} \left( \sum_{t=1}^{N} \gamma^t r_t \right)
$$

(11)

where $\gamma$ is a discount rate, and use the policy gradient ascent.

$$
\theta \leftarrow \theta + \lambda \sum_{t=1}^{N} \nabla_{\theta} \log \pi_\theta(o_t, a_t) \left( G_t - b(t) \right)
$$

(12)

Note that $G_t = \sum_{k=t}^{N} \gamma^{k-t} r_{k+1}$ is a return, $\lambda$ is a learning rate, and $b(t)$ is a baseline used to accelerate the convergence of model training.

**Learning-to-Rank Distillation**

In the RLRD framework, the ranking decoder repeats $N$-times of selection to rank $N$-items through its sequential structure. While the decoder structure is intended to extract the relational features of items that have not been selected, the high computing complexity of iterative decoding renders difficulties in the application of the framework to mission-critical systems. To enable fast inferences without significant degradation in model performance, we employ knowledge distillation from an RL-to-rank model with iterative decoding to a simpler model. Specifically, we use a non-iterative, score-based ranking model as a student in knowledge distillation, which takes the features of $N$-items as input and directly produces a score vector for the $N$-items as output. A score vector is used to rank the $N$-items.

For $N$-items, the RL-to-rank model produces ranking vector as supervised label $y = [\phi_1, \phi_2, \ldots, \phi_N]$, and by distillation, the student model learns to produce such a score vector $s$ maximizing the similarity between $y$ and the corresponding ranking of $s$, say $\text{rank}(s)$. For example, given a score vector $s = [2.4, 1.3, 3.0, 0.1]$ for 4-items, it is sorted to $[3.0, 2.4, 1.3, 0.1]$, so $\text{rank}(s) = [2, 3, 1, 4]$. The ranking distillation loss is defined as

$$
\mathcal{L}(y, s) = \mathcal{L}_R(y, \text{rank}(s))
$$

(13)

where $\mathcal{L}_R$ is a differentiable evaluation metric for the similarity of two ranking vectors. We use mean squared error (MSE) for $\mathcal{L}_R$, because minimizing MSE of two ranking vectors is equivalent to maximizing the Spearman-rho correlation of two rankings $y$ and $\text{rank}(s)$.

**Differentiable Approximated Ranking.** To distill with the loss in (13) using gradient descent, the ranking function $\text{rank}$ needs to be differentiable with non-vanishing gradient. However, differentiating $\text{rank}$ has a problem of vanishing gradient because a slight shift of score $s$ does not usually affect the corresponding ranking. Thus, we revise the loss in (13) using an approximated ranking function having nonzero gradients in the same way of (Blondel et al. 2020).

Consider score $s = [s_1, \ldots, s_N]$ and $N$-permutation $\phi$ which is a bijection from $\{1, \ldots, N\}$ to itself. A descending sorted list of $s$ is represented as

$$
s_{\phi} = [s_{\phi(1)}, \ldots, s_{\phi(N)}]
$$

(14)
where \( s_{\phi(1)} \geq \cdots \geq s_{\phi(N)} \). Accordingly, the ranking function \( \text{rank} : \mathbb{R}^N \to \mathbb{R}^N \) is formalized as

\[
\text{rank}(s) = [\phi^{-1}(1), \cdots, \phi^{-1}(N)]
\]

where \( \phi^{-1} \) is an inverse of \( \phi \), which is also a permutation. For example, consider \( s = [2.4, 1.3, 3.0, 0.1] \). Its descending sort is \([3.0, 2.4, 1.3, 0.1] \), so we have \( \phi(1) = 3, \phi(2) = 1, \phi(3) = 2 \) and \( \phi(4) = 4 \). Accordingly, we have \( \text{rank}(s) = [2, 3, 1, 4] \).

To implement the DiffRank, a function \( \text{rk}_e \) is used, which approximates \( \text{rank} \) in a differential way with nonzero gradients such as

\[
\text{rk}_e(s) = \text{ArgMin} \left\{ \frac{1}{2} \| x + s^2 \| : x \in \mathcal{P}(\tau) \right\}. \tag{16}
\]

Here \( \mathcal{P}(\tau) \) is called a permutahedron, which is a convex hull generated by the permutation \( \tau = [N, N-1, \cdots, 1] \) with \( \epsilon > 0 \). As explained in (Blondel et al. 2020), the function \( \text{rk}_e \) converges to \( \text{rank} \) as \( \epsilon \to 0 \), while it always preserves the order of \( \text{rank}(s) \). That is, given \( s_i \) in (14) and \( \text{rk}_e(s_i) = [\psi_1, \cdots, \psi_N] \), we have \( \psi_{\phi(1)} \leq \cdots \leq \psi_{\phi(N)} \).

In addition, we also consider a problem-specific loss. For example, in the knapsack problem, an entire set of items can be partitioned into two groups, one for selected items and the other for not selected items. We can penalize the difference of partitions obtained from label \( y \) and target output score \( s \) by the function \( \mathcal{L}_P \). Finally, the total loss is given by

\[
\mathcal{L}(y, s) = \alpha \mathcal{L}_R(y, \text{rk}_e(s)) + (1 - \alpha)\mathcal{L}_P(y, s)
\]

where \( \alpha \in [0, 1] \). The overall distillation procedure is illustrated in Algorithm 1.

Here, we present the explicit nonvanishing gradient form of our ranking loss function \( \mathcal{L}_R \), whose proof can be found in Appendix A.

**Proposition 1.** Fix \( r = [r_1, \cdots, r_n] \in \mathbb{R}^n \). Let \( \text{rk}_e : \mathbb{R}^n \to \mathbb{R}^n \) as in (16) and \( L : \mathbb{R}^n \to \mathbb{R}^n \) where \( L(y) = \frac{1}{2} \| y - r \|_2^2 \). Let \( g = L \circ \text{rk}_e \),

\[
\text{rk}_e(s) = [\tilde{r}_1, \cdots, \tilde{r}_n], \tag{18}
\]

and \( e_i = \tilde{r}_i - r_i \). Then, we have

\[
\frac{\partial g}{\partial s}(s) = -\frac{1}{\epsilon} [1, \cdots, 1] e_i \tag{19}
\]

\[
\begin{pmatrix}
\frac{1}{n} I_{k_1} & 0 & \cdots & 0 \\
0 & \frac{1}{n} I_{k_2} & \cdots & 0 \\
\vdots & \vdots & \ddots & \vdots \\
0 & 0 & \cdots & \frac{1}{n_{m}} I_{k_m}
\end{pmatrix}
\]

where \( \phi \) is a matrix multiplication, \( k_1 + \cdots + k_m = n \). \( I_{k_i} \) is a square matrix whose entries are \( 1 \) with size \( k_1 \times k_1 \), and \( \phi \) is an \( n \)-permutation. Here, for any matrix \( M \), \( M_\phi \) denotes row and column permutation of \( M \) according to \( \phi \).

**Algorithm 1: Learning-to-rank distillation**

1: Load (teacher) RL policy \( \pi \)
2: Initialize parameter \( \theta_s \) of student model
3: Input : Sample batch \( B \) and learning rate \( \lambda_s \)
4: for \( i \sim N \) do
5: \( \nabla \theta_s \leftarrow 0 \)
6: for Itemset \( I \in B \) do
7: Get rank = \( \{\phi(1), \cdots, \phi(n)\} \) from \( \pi \)
8: Get score \( s = [s_1, \cdots, s_n] \) from student model.
9: Calculate \( \text{rk}_e(s) = [\psi(1), \cdots, \psi(N)] \) using (16)
10: Calculate loss \( J(\theta_s) = L(L, \tilde{R}) \) using (17)
11: \( \nabla \theta_s \leftarrow 0 + \nabla \theta_s + \nabla J(\theta_s) \)
12: end for
13: \( \theta_s \leftarrow \theta_s + \lambda_s \nabla \theta_s \)
14: end for

**Efficient Sequence Sampling.** As explained, we use a score vector in (14) to obtain its corresponding ranking vector deterministically. On the other hand, if we treat such a score vector as an un-normalized log-probability of a categorical distribution on \( N \)-items, we can randomly sample from the distribution using the score vector without replacement and obtain a ranking vector for the \( N \)-items. Here, the condition of without replacement specifies that the distribution is renormalized so that it sums up to 1.0 for each time to sample an item. This \( N \)-times drawing and normalization increases the inference time. Therefore, to compute rankings rapidly, we exploit Gumbel trick (Gumbel 1954; Maddison, Tarlow, and Minka 2014).

Given score \( s \), consider the random variable \( S = s + Z \) where

\[
Z = -\log(-\log(\text{Unif}(0, 1))) \tag{21}
\]

and suppose that \( S \) is sorted by \( \phi \), as in (14). Note that \( \phi(1), \phi(2), \cdots, \phi(N) \) are random variables.

**Theorem 1.** Appendix A in (Kool, van Hoof, and Welling 2020). For each \( k \in \{1, 2, \cdots, N\} \),

\[
\mathcal{P}(\phi(1) = i_1, \cdots, \phi(k) = i_k) = \prod_{j=1}^{k} \frac{\exp(s_{i_j})}{\sum_{i \in R^*_j} \exp(s_i)} \tag{22}
\]

where \( R^*_j = \{1, 2, \cdots, N\} - \{i_1, \cdots, i_{j-1}\} \).

This sampling method reduces the complexity to obtain each ranking vector instance from quadratic (sequentially sampling each of \( N \)-items on a categorical distribution) to log-linear (sorting a perturbed list) of the number of items \( N \), improving the efficiency of our model significantly.

**Experiments**

In this section, we evaluate our framework with multidimensional knapsack problem (MDKP) and global fixed-priority task scheduling (GFPS) (Davis et al. 2016). The problem details including RL formulation, data generation, and model training can be found in Appendix B and C.
Multidimensional Knapsack Problem (MDKP)
Given values and $k$-dimensional weights of $N$-items, in MDKP, each item is either selected or not for a knapsack with $k$-dimensional capacities to get the maximum total value of selected items.

For evaluation, we use the performance (the achieved value in a knapsack) by GLOP implemented in the OR-tools (Perron and Furnon 2019-7-19) as a baseline. We compare several models in our framework. RL is the RL-to-rank teacher model, and RD is the distilled student model. RL-S is the RL model variant with sampling, and RD-G is the RD model variant with Gumbel trick-based sequence sampling. In RL-S, the one-step action in (9) is conducted stochastically, while in RL, it is conducted greedily. For both RL-S and RD-G, we set the number of ranking sampling to 30 for each item set, and report the the highest score among samples. In addition, we test the Greedy method that exploits the mean ratio of item weight and value.

Model Performance. Table 1 shows the performance in MDKP, where GAP denotes the performance ratio to the baseline GLOP and Time denotes the inference time.

<table>
<thead>
<tr>
<th>$N$</th>
<th>$k$</th>
<th>$w$</th>
<th>$\alpha$</th>
<th>GLOP - Time</th>
<th>Greedy</th>
<th>RL</th>
<th>RL-S</th>
<th>RD</th>
<th>RD-G</th>
</tr>
</thead>
<tbody>
<tr>
<td>50</td>
<td>3</td>
<td>0</td>
<td>0.9</td>
<td>100% 0.00606s</td>
<td>97.9% 0.00003s</td>
<td>99.7% 0.0706s</td>
<td>98.8% 2.0051s</td>
<td>97.5% 0.00291s</td>
<td>100.1% 0.0152s</td>
</tr>
<tr>
<td>100</td>
<td>10</td>
<td>0</td>
<td>0.9</td>
<td>100% 0.00355s</td>
<td>99.9% 0.00003s</td>
<td>99.6% 0.0656s</td>
<td>99.9% 1.8844s</td>
<td>98.7% 0.00291s</td>
<td>100.7% 0.0152s</td>
</tr>
</tbody>
</table>

Table 1: The Evaluation of MDKP. For each method, Gap denotes the performance (average achieved value) ratio of the method to GLOP, and Time denotes the average inference time for a problem instance. $N$ and $k$ denote the number of items and the size of knapsack resource dimensions, respectively. $w$ denotes the sampling range of item weight on $[1, w]$, and $\alpha$ denotes the correlation of weight and value of items. The performance is averaged for a testing dataset of 500 item sets.

- The performance of Greedy degrades drastically in the case of $\alpha=0$.9. This is because the weight-value ratio for items becomes less useful when the correlation is high. Unlike Greedy, our models show stable performance for both high and low correlation cases.

Priority Assignment Problem for GFPS
For a set of $N$-periodic tasks, in GFPS, each task is assigned a priority (an integer from 1 to $N$) to be scheduled. GFPS with a priority order (or a ranking list) can schedule the $m$ highest-priority tasks in each time slot upon a platform comprised of $m$-processors, with the goal of not incurring any deadline violation of the periodic tasks.

For evaluation, we need to choose a schedulability test for GFPS, that determines whether a task set is deemed schedulable by GFPS with a priority order. We target a schedulability test called RTA-LC (Guan et al. 2009; Davis and Burns 2011) which has been known to perform superior to the others in terms of covering schedulable task sets. We compare our models with Audsley’s Optimal Priority Assignment (OPA) (Audsley 1991, 2001) with the state-of-the-art OPA-compatible DA-LC test (Davis and Burns 2011), which is known to have the highest performance compared to other heuristic algorithms. Same as those in MDKP, we denote our models as RL, RL-S, RD, and RD-G. For both RL-S and RD-G, we limit the number of ranking samples to 10 for each task set.

Model Performance. Table 2 shows the performance in the schedulability ratio of GFPS with respect to different task set utilization settings on an $m$-processor platform and $N$-tasks.

- Our models all show better performance than OPA, indicating the superiority of the RLRD framework. The performance gap is relatively large on the intermediate utilization ranges, because those ranges can provide more opportunities to optimize with a better strategy. For example, when $m=8$, $N=64$ and Util=6.3, RL and RD show...
The performance difference of RD and its teacher RL is about 1% on average, while the inference time of RD is decreased (improved) by 3.8 times. This clarifies the benefit of the ranking distillation.

As the utilization (Util) increases, the inference time of RL-S and RD-G becomes longer, due to multiple executions of the schedulability test up to the predefined limit (i.e., 10 times). On the other hand, the inference time of OPA decreases for large utilization; the loop of OPA is terminated when a task cannot satisfy its deadline with the assumption that other priority-unassigned tasks have higher priorities than that task.

• RD-G shows comparable performance to, and often achieves slight higher performance than RL-S. This is the opposite pattern of MDKP where RL-S achieves the best performance. While direct comparison is not much valid due to different sampling methods, we notice the possibility that a distilled student can perform better than its teacher for some cases, and the similar patterns are observed in (Tang and Wang 2018; Kim and Rush 2016).

### Analysis on Distillation

#### Effects of Iterative Decoding

To verify the feasibility of distillation from sequential RL to a score-based ranking model, we measure the difference of the outputs by iterative decoding and greedy sampling. In the case when the decoder generates the ranking distribution at timestep $t$ and takes action $a_i = t$ as in (9), by masking the $i$th component of the distribution and renormalizing it, we can obtain a renormalized distribution $P_{RE}$. In addition, consider another probability distribution $P_{DE}$ generated by the decoder at $t + 1$.

Figure 2 illustrates the difference of the distributions in terms of KL-divergence on three specific COPs, previously explained MDKP and GFPS as well as Traveling Salesman Problem (TSP). As shown, MDKP and GFPS maintain a low divergence value, implying that the ranking result of decoding with many iterations can be approximated by decoding with no or few iterations. Unlike MDKP and GFPS, TSP shows a large divergence value. This implies that many decoding iterations are required to obtain an optimal path. Indeed, in TSP, we obtain good performance by RL (e.g. 2% better than a heuristic method), but we hardly achieve comparable performance to RL when we test RD. The experiment and performance for TSP can be found in Appendix D.

#### Effects of Distillation Loss

To evaluate the effectiveness of DiffRank-based distillation, we implement other ranking metrics such as a pairwise metric in (Burges et al. 2005) and a listwise metric in (Cao et al. 2007) and test them in the framework as a distillation loss.
Table 3: Ranking Loss Comparison in MDKP.

<table>
<thead>
<tr>
<th>N</th>
<th>k</th>
<th>w</th>
<th>α</th>
<th>RD</th>
<th>Pairwise</th>
<th>Listwise</th>
</tr>
</thead>
<tbody>
<tr>
<td>50</td>
<td>5</td>
<td>200</td>
<td>0</td>
<td>100%</td>
<td>99.4%</td>
<td>69.8%</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0.9</td>
<td>100%</td>
<td>99.1%</td>
<td>98.5%</td>
</tr>
<tr>
<td>100</td>
<td>10</td>
<td>200</td>
<td>0</td>
<td>100%</td>
<td>98.8%</td>
<td>80.1%</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0.9</td>
<td>100%</td>
<td>100.1%</td>
<td>99.5%</td>
</tr>
<tr>
<td>150</td>
<td>15</td>
<td>200</td>
<td>0</td>
<td>100%</td>
<td>93.4%</td>
<td>86.8%</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0.9</td>
<td>100%</td>
<td>99.6%</td>
<td>99.6%</td>
</tr>
</tbody>
</table>

Table 4: Ranking Loss Comparison in GFPS.

<table>
<thead>
<tr>
<th>m</th>
<th>N</th>
<th>Util</th>
<th>RD</th>
<th>Pairwise</th>
<th>Listwise</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>32</td>
<td>3.1</td>
<td>100%</td>
<td>90.1%</td>
<td>72.8%</td>
</tr>
<tr>
<td>6</td>
<td>48</td>
<td>4.6</td>
<td>100%</td>
<td>91.3%</td>
<td>59.6%</td>
</tr>
<tr>
<td>8</td>
<td>64</td>
<td>6.0</td>
<td>100%</td>
<td>91.6%</td>
<td>49.2%</td>
</tr>
</tbody>
</table>

Table 3 and 4 show the performance in MDKP and GFPS, respectively, achieved by different distillation loss functions, where RD denotes our distilled model trained with DiffRank-based distillation loss, and the performance of the other two is represented as the ratio to RD. Note that they all use the same RL model as a teacher in this experiment.

As shown, RD achieves consistently better performance than the others for most cases. Unlike RD, the other methods commonly show data-dependent performance patterns. The pairwise method (with pairwise distillation loss) achieves performance similar to or slightly lower than RD in MDKP but shows much lower performance than RD in GFPS. The listwise method shows the worst performance for many cases in both MDKP and GFPS except for the cases of $\alpha = 0.9$ in MDKP. These results are consistent with the implication in Figure 2 such that GFPS has larger divergence than MDKP and thus GFPS is more difficult to distill, giving a large performance gain to RD.

**Related Work**

Advanced deep neural networks combined with RL algorithms showed the capability to address various COPs in a data-driven manner with less problem-specific customization. In (Bello et al. 2017), the pointer network was introduced to solve TSP and other geometric COPs, and in (Kool, van Hoof, and Welling 2019), a transformer model was incorporated for more generalization. Besides the pointer network, a temporal difference based model showed positive results in the Job-Shop problem (Zhang and Dietterich 1995), and deep RL-based approaches such as Q-learning solvers (Afshar et al. 2020) were explored for the knapsack problem (Afshar et al. 2020). Several attempts have been made to address practical cases formulated in the knapsack problem, e.g., maximizing user engagements under business constraints (Agarwal et al. 2015; Gupta et al. 2016).

Particularly, in the field of computer systems and resource management, there have been several works using deep RL to tackle system optimization under multiple, heterogeneous resource constraints in the form of COPs, e.g., cluster resource management (Mao et al. 2016, 2019), compiler optimization (Chen et al. 2018). While we leverage deep RL techniques to address COPs in the same vein as those prior works, we focus on efficient, low-latency COP models.

The ranking problems such as prioritizing input items based on some scores have been studied in the field of information retrieval and recommendation systems. A neural network based rank optimizer using a pairwise loss function was first introduced in (Burges et al. 2005), and other ranking objective functions were developed to optimize relevant metrics with sophisticated network structures. For example, Bayesian Personalized Ranking (Rendle et al. 2009) is known to maximize the AUC score of given item rankings with labeled data. However, although these approaches can bypass the non-differentiability of ranking operations, the optimization is limited to some predefined objectives such as NDCG or AUC; thus, it is difficult to apply them to COPs because the objectives do not completely align with the COP objectives. To optimize arbitrary objectives involving non-differentiable operations such as ranking or sorting, several works focused on smoothing nondifferentiable ranking operations (Grover et al. 2019; Blondel et al. 2020). They are commonly intended to make arbitrary objectives differentiable by employing relaxed sorting operations.

Knowledge distillation based on the teacher-student paradigm has been an important topic in machine learning to build a compressed model (Hinton, Vinyals, and Dean 2015) and showed many successful practices in image classification (Touvron et al. 2021) and natural language processing (Kim and Rush 2016; Jiao et al. 2020). However, knowledge distillation to ranking models has not been fully studied. A ranking distillation for recommendation system was introduced in (Tang and Wang 2018), and recently, a general distillation framework RankDistil (Reddi et al. 2021) was presented with several loss functions and optimization schemes specific to top-K ranking problems. These works exploited pairwise objectives and sampling based heuristics to distill a ranking model, but rarely focused on arbitrary objectives and sequential models, which are required to address various COPs. The distillation of sequential models was investigated in several works (Kim and Rush 2016; Jiao et al. 2020). However, to the best of our knowledge, our work is the first to explore the distillation from sequential RL models to score-based ranking models.

**Conclusion**

In this paper, we presented a distillation-based COP framework by which an efficient model with high-performance is achieved. Through experiments, we demonstrate that it is feasible to distill the ranking policy of deep RL to a score-based ranking model without compromising performance, thereby enabling the low-latency inference on COPs.

The direction of our future work is to adapt our RL-based encoder-decoder model and distillation procedure for various COPs with different consistency degrees between embeddings and decoding results and to explore meta learning for fast adaptation across different problem conditions.
Acknowledgements

We would like to thank anonymous reviewers for their valuable comments and suggestions.

This work was supported by the Institute for Information and Communications Technology Planning and Evaluation (IITP) under Grant 2021-0-00875 and 2021-0-00900, by the ICT Creative Consilience Program supervised by the IITP under Grant IITP-2020-0-01821, by Kakao I Research Supporting Program, and by Samsung Electronics.

References


