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

Honguk Woo[1]*, **Hyunsung Lee**[2] †, **Sangwoo Cho**[1] †

[1]Department of Computer Science and Engineering, Sungkyunkwan University
[2]Kakao 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

*Honguk Woo is the corresponding author.
†Lead authors, equally contributed.

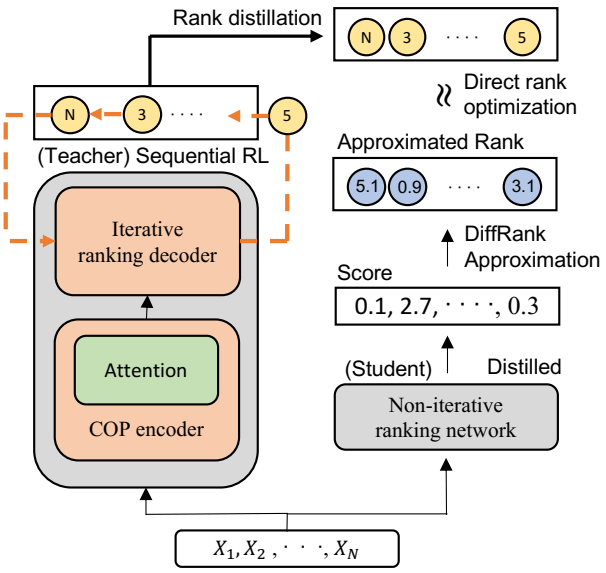

Figure 1: Overview of RLRD framework

sequence samples and improve the model performance with an inference time limitation.

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 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 production system 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 θ (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, $h_i^{(0)} = Wx_i + b$. Then, for N-items, $(h \times N)$-matrix, $H^{(0)} = [h_1^{(0)}, \cdots, h_N^{(0)}]$ 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

$$\begin{aligned} H^{(l)} &= \mathbf{BN}^{(l)}(X + \mathbf{FF}^{(l)}(X)), \\ X &= \mathbf{BN}^{(l)}(H^{(l-1)} + \mathbf{MHA}(H^{(l-1)})) \end{aligned} \tag{1}$$

where

$$\mathbf{MHA}(X) = W_G(\mathbf{AM}_1(X) \odot \cdots \odot \mathbf{AM}_{d_h}(X)), \tag{2}$$

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

$$\mathbf{AM}_j(X) = W_V(X)\mathbf{Softmax}(\frac{1}{d_h}(W_K(X)^T W_Q(X))) \tag{3}$$

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 ϕ_1 is selected first, so it is assigned the highest ranking (priority), and item ϕ_2 is assigned the second, and so on. Specifically, the decoder establishes a function to rank N-items stochastically,

$$\mathbb{P}(\phi|H^{(L)}) = \prod_{t=2}^{N} \mathbb{P}(\phi_t|\phi_1, \ldots, \phi_{t-1}, H^{(L)}) \quad (4)$$

where $\mathbb{P}(\phi_t|\phi_1, \ldots, \phi_{t-1}, H^{(L)})$ represents a probability that item ϕ_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 ϕ_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 θ-parameterized policy π_θ for each timestep t.

$$\pi_\theta(o_t, a_t) = \mathbb{P}(a_t \mid o_t) = \mathbb{P}(\phi_t|\phi_1, \ldots, \phi_{t-1}, H^{(L)}) \quad (5)$$

This policy is learned based on problem-specific reward signals. To establish such policy π_θ 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.,

$$\begin{aligned} &\mathbf{SCORE}_{H^{(L)}, l^{(t-1)}}(x_i) \\ &= \begin{cases} 10 * \mathbf{tanh}\left(\mathbf{Att}(x^{(g)}, H^{(L)})_i\right) & \text{if } i \notin [\phi_1, \cdots, \phi_{t-1}) \\ -\infty & \text{otherwise} \end{cases} \end{aligned} \quad (6)$$

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

$$x^{(g)} = l^{(t-1)} \odot \mathbf{Att}(l^{(t-1)}, H^{(L)}). \quad (7)$$

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

$$\begin{aligned} \mathbf{Att}(q, Y) = \sum_{i=1}^{N} \alpha_i y_i, \;\; \alpha_i = \mathbf{Softmax}(w_1, \cdots, w_N), \\ w_i = V_A \, \mathbf{tanh} \, W_A[y_i \odot q] \end{aligned} \quad (8)$$

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

$$\pi_\theta(o_t, a_t = i) = \mathbb{P}(a_t = i \mid o_t) = \frac{e^{\mathbf{SCORE}(x_i)}}{\sum_{k=1}^{N} e^{\mathbf{SCORE}(x_k)}}. \quad (9)$$

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

$$T^{(\theta)} = (s_1, a_1, r_1, \cdots, s_N, a_N, r_N) \quad (10)$$

that are acquired by policy π_θ, 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 π_θ,

$$J(\theta) = \mathbb{E}_{\pi_\theta}\left(\sum_{t=1}^{N} \gamma^t r_t\right) \quad (11)$$

where γ is a discount rate, and use the policy gradient ascent.

$$\theta \leftarrow \theta + \lambda \sum_{t=1}^{N} \nabla_\theta \log \pi_\theta(o_t, a_t) \left(G_t - b(t)\right) \quad (12)$$

Note that $G_t = \sum_{k=t}^{N} \gamma^{k-t} r_{k+1}$ is a return, λ 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 $\mathbf{y} = [\phi_1, \phi_2, \ldots, \phi_N]$, and by distillation, the student model learns to produce such a score vector $\mathbf{s}$ maximizing the similarity between $\mathbf{y}$ and the corresponding ranking of $\mathbf{s}$, say $\mathbf{rank}(\mathbf{s})$. For example, given a score vector $\mathbf{s} = [2.4, 1.3, 3.0, 0.1]$ for 4-items, it is sorted to $[3.0, 2.4, 1.3, 0.1]$, so $\mathbf{rank}(\mathbf{s}) = [2, 3, 1, 4]$. The ranking distillation loss is defined as

$$\mathcal{L}(\mathbf{y}, \mathbf{s}) = \mathcal{L}_R\left(\mathbf{y}, \mathbf{rank}(\mathbf{s})\right) \quad (13)$$

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 $\mathbf{y}$ and $\mathbf{rank}(\mathbf{s})$.

Differentiable Approximated Ranking. To distill with the loss in (13) using gradient descent, the ranking function **rank** needs to be differentiable with non-vanishing gradient. However, differentiating **rank** has a problem of vanishing gradient because a slight shift of score $\mathbf{s}$ 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 $\mathbf{s} = [s_1, \cdots, s_N]$ and N-permutation ϕ which is a bijection from $\{1, \cdots, N\}$ to itself. A descending sorted list of $\mathbf{s}$ is represented as

$$\mathbf{s}_\phi = [s_{\phi(1)}, \cdots, s_{\phi(N)}] \quad (14)$$

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

$$\mathbf{rank}(\mathbf{s}) = [\phi^{-1}(1), \cdots, \phi^{-1}(N)] \tag{15}$$

where ϕ^{-1} is an inverse of ϕ, which is also a permutation. For example, consider $\mathbf{s} = [2.4, 1.3, 3.0, 0.1]$. 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 $\mathbf{rank}(\mathbf{s}) = [2, 3, 1, 4]$.

To implement the DiffRank, a function $\tilde{\mathbf{rk}}_\epsilon$ is used, which approximates **rank** in a differential way with nonzero gradients such as

$$\tilde{\mathbf{rk}}_\epsilon(\mathbf{s}) = \text{ArgMin}\left\{\frac{1}{2}\|\mathbf{x} + \frac{\mathbf{s}}{\epsilon}^2\| \mid \mathbf{x} \in \mathcal{P}(\tau)\right\}. \tag{16}$$

Here $\mathcal{P}(\tau)$ is called a perumutahedron, 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 $\tilde{\mathbf{rk}}_\epsilon$ converges to **rank** as $\epsilon \to 0$, while it always preserves the order of $\mathbf{rank}(\mathbf{s})$. That is, given $\mathbf{s}_\phi$ in (14) and $\tilde{\mathbf{rk}}_\epsilon(s) = [\psi_1, \cdots, \psi_N]$, 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 $\mathbf{y}$ and target output score $\mathbf{s}$ by the function $\mathcal{L}_P$. Finally the total loss is given by

$$\mathcal{L}(\mathbf{y}, \mathbf{s}) = \alpha\, \mathcal{L}_R\left(\mathbf{y}, \tilde{\mathbf{rk}}_\epsilon(\mathbf{s})\right) + (1-\alpha)\mathcal{L}_P(\mathbf{y}, \mathbf{s}) \tag{17}$$

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 $\mathcal{L}_R$, where its proof can be found in Appendix A.

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

$$\tilde{\boldsymbol{rk}}_\epsilon(\boldsymbol{s}) = [\tilde{r}_1, \cdots, \tilde{r}_n], \tag{18}$$

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

$$\frac{\partial g}{\partial \boldsymbol{x}}(\boldsymbol{s}) = -\frac{\boldsymbol{I}}{\epsilon}[e_1, \cdots, e_n] \tag{19}$$

$$@\left(\boldsymbol{I} - \begin{bmatrix} \frac{1}{k_1}\boldsymbol{1}_{k_1} & \boldsymbol{0} & \cdots & \boldsymbol{0} \\ \boldsymbol{0} & \frac{1}{k_2}\boldsymbol{1}_{k_2} & \cdots & \boldsymbol{0} \\ \vdots & \vdots & \vdots & \vdots \\ \boldsymbol{0} & \boldsymbol{0} & \cdots & \frac{1}{k_m}\boldsymbol{1}_{k_m} \end{bmatrix}\right)_\phi \tag{20}$$

where @ is a matrix multiplication, $k_1 + \cdots + k_m = n$, $\boldsymbol{1}_{k_j}$ is a square matrix whose entries are all 1 with size $k_j \times k_j$, and ϕ is an n-permutation. Here, for any matrix M, M_ϕ denotes row and column permutation of M according to ϕ.

Algorithm 1: Learning-to-rank distillation

```
Load (teacher) RL policy π
Initialize parameter θ_s of student model
Input : Sample batch B and learning rate λ_s.
for 1 ∼ N_s do
    ∇θ_s ⟵ 0.
    for Itemset I ∈ B do
        Get rank = {φ(1), ··· , φ(n)} from π
        Get score s = {s_1, ··· , s_n} from student model.
        Calculate rk̃_ε(s) = {ψ̃(1), ··· , ψ̃(n)} using (16)
        Calculate loss J(θ_s) = L(L, R̃) using (17)
        ∇θ_s ⟵ ∇θ_s + ∇J(θ_s)
    end for
    θ_s ⟵ θ_s + λ_s ∇θ_s
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 $\mathbf{s}$, consider the random variable $\mathbf{S} = \mathbf{s} + Z$ where

$$Z = -\log\left(-\log(\mathbf{Unif}(0, 1))\right), \tag{21}$$

and suppose that $\mathbf{S}$ is sorted by ϕ, 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 $k \in \{1, 2, \cdots, N\}$,*

$$\mathbb{P}(\phi(1) = i_1, \cdots, \phi(k) = i_k) = \prod_{j=1}^{k} \frac{\exp(s_{i_l})}{\sum_{l \in R_j^*} \exp(s_l)} \tag{22}$$

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.

N	k	w	α	GLOP		Greedy		RL		RL-S		RD		RD-G	
				-	Time	Gap	Time	Gap	Time	Gap	Time	Gap	Time	Gap	Time
50	3	200	0	100%	0.0060s	97.9%	0.0003s	99.7%	0.0706s	98.8%	2.0051s	97.5%	0.0029s	100.1%	0.0152s
			0.9	100%	0.0063s	87.6%	0.0003s	100.2%	0.0675s	104.5%	1.8844s	97.7%	0.0030s	102.2%	0.0154s
		500	0	100%	0.0066s	97.8%	0.0003s	99.4%	0.0686s	99.6%	1.9768s	97.4%	0.0029s	99.7%	0.0159s
			0.9	100%	0.0064s	81.4%	0.0004s	101.5%	0.0687s	105.4%	1.8840s	97.9%	0.0030s	101.5%	0.0150s
100	10	200	0	100%	0.0950s	101.3%	0.0005s	102.2%	0.4444s	101.9%	12.4996s	100.5%	0.0060s	102.7%	0.0198s
			0.9	100%	0.0435s	93.4%	0.0004s	103.2%	0.4443s	106.9%	12.3932s	99.2%	0.0086s	102.6%	0.0222s
		500	0	100%	0.1046s	100.9%	0.0005s	100.6%	0.4363s	101.1%	12.3392s	98.9%	0.0088s	101.6%	0.0214s
			0.9	100%	0.0436s	90.2%	0.0004s	103.5%	0.4381s	107.1%	12.3211s	100.4%	0.0059s	104.6%	0.0198s
150	15	200	0	100%	0.2494s	102.0%	0.0007s	102.9%	0.6370s	102.8%	17.8975s	100.7%	0.0090s	102.5%	0.0328s
			0.9	100%	0.0885s	96.7%	0.0006s	103.4%	0.5380s	106.4%	16.2406s	99.7%	0.0090s	102.3%	0.0290s
		500	0	100%	0.2497s	101.8%	0.0007s	99.1%	0.6488s	100.1%	19.4974s	96.6%	0.0093s	98.6%	0.0244s
			0.9	100%	0.0425s	92.5%	0.0006s	103.7%	0.5289s	107.0%	14.0924s	100.7%	0.0088s	103.9%	0.0292s

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 α denotes the correlation of weight and value of items. The performance is averaged for a testing dataset of 500 item sets.

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.

- RD shows comparable performance to its respective high-performance teacher RL, with insignificant degradation of average 2.6% for all the cases. More importantly, RD achieves efficient, low-latency inferences, e.g., 23 and 65 times faster inferences than RL for N=50 and N=150 cases, respectively.
- RD-G outperforms RL by 0.3% on average and also achieves 4.4 and 20 times faster inferences than RL for N=50 and N=150 cases, respectively. Moreover, RD-G shows 2% higher performance than RD, while its inference time is increased by 3.7 times.
- RL-S shows 1.8% higher performance than RL model. However, unlike RD-G, the inference time of RL-S is increased linearly to the number of ranking samples (i.e., about 30 times increase for 30 samples).
- As N increases, all methods shows longer inference time, but the increment gap of GLOP is much larger than RL and RD. For example, as N increases from 50 to 150 when α=0, the inference time of GLOP is increased by 39 times, while RL and RD shows 9.3 and 3.1 times increments, respectively.
- The performance of Greedy degrades drastically in the case of α=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

m	N	Util	OPA		RL		RL-S		RD		RD-G	
			Ratio	Time	Ratio	Time	Ratio	Time	Ratio	Time	Ratio	Time
4	32	3.0	78.1%	0.3531s	87.5%	0.0616s	89.4%	0.0697s	86.5%	0.0145s	90.1%	0.0263s
		3.1	63.5%	0.3592s	74.8%	0.0599s	77.6%	0.0840s	73.8%	0.0139s	78.9%	0.0390s
		3.2	44.9%	0.3487s	56.9%	0.0623s	60.1%	0.0991s	56.0%	0.0140s	61.2%	0.0509s
		3.3	26.6%	0.3528s	35.7%	0.0621s	38.5%	0.9123s	35.8%	0.0131s	39.2%	0.0620s
6	48	4.4	84.2%	0.4701s	92.5%	0.1021s	94.3%	0.1153s	91.76%	0.0298s	94.3%	0.0406s
		4.6	61.9%	0.4600s	78.4%	0.1057s	78.4%	0.1508s	74.6%	0.0308s	79.3%	0.0728s
		4.8	33.2%	0.4287s	46.5%	0.1082s	50.4%	0.1967s	45.4%	0.0290s	50.8%	0.1123s
		5.0	11.5%	0.3888s	15.7%	0.1010s	18.1%	0.2474s	18.0%	0.0256s	20.2%	0.1615s
8	64	5.7	92.9%	0.6686s	97.8%	0.1437s	98.6%	0.1596s	97.5%	0.0502s	98.5%	0.0537s
		6.0	72.9%	0.6460s	86.5%	0.1490s	89.9%	0.2043s	85.0%	0.0364s	88.7%	0.0907s
		6.3	37.6%	0.5798s	53.5%	0.1559s	57.5%	0.2800s	52.5%	0.0509s	57.7%	0.1695s
		6.6	10.4%	0.4806s	15.1%	0.1488s	17.7%	0.4093s	17.0%	0.0390s	19.6%	0.2715s

Table 2: The Evaluation of GFPS. For each method, Ratio denotes the performance in the schedulability ratio ($\frac{\text{num of schedulable task sets}}{\text{num. of task sets}}$), and Time denotes the average inference time for a problem instance. m and N denote the number of processors and the number of tasks, respectively, i.e., scheduling N-tasks on an m-processor platform. Util denotes the task set utilization, i.e., the sum of task utilization ($\sum \frac{\text{task exe. time}}{\text{task period}}$). The performance is averaged for a testing dataset of 5,000 task sets.

15.9% and 11.9% higher schedulability ratio than OPA, respectively, while when m=8, N=64 and Util=5.7, their gain is 4.9% and 4.6%, respectively.

- 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 $a_t = i$ as in (9), by masking the ith 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.

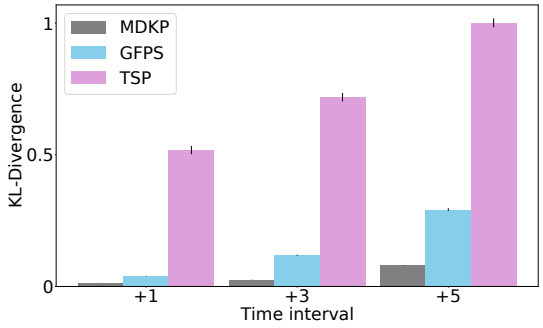

Figure 2: The KL divergence at different timesteps t and $t+\delta$ is used to estimate the difference of decoder driven ranking distribution (P_{DE}) and renormalized ranking distribution (P_{RE}), where time intervals $\delta \in \{1, 3, 5\}$. Scales are normalized.

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.

N	k	w	α	RD	Pairwise	Listwise
50	5	200	0	100%	99.4%	69.8%
			0.9	100%	99.1%	98.5%
100	10	200	0	100%	98.8%	80.1%
			0.9	100%	100.1%	99.2%
150	15	200	0	100%	95.4%	86.8%
			0.9	100%	99.6%	99.6%

Table 3: Ranking Loss Comparison in MDKP.

m	N	Util	RD	Pairwise	Listwise
4	32	3.1	100%	90.1%	72.8%
6	48	4.6	100%	91.3%	59.6%
8	64	6.0	100%	91.6%	49.2%

Table 4: Ranking Loss Comparison in GFPS.

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 also 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

Afshar, R. R.; Zhang, Y.; Firat, M.; and Kaymak, U. 2020. A State Aggregation Approach for Solving Knapsack Problem with Deep Reinforcement Learning. In *Proceedings of the 12th Asian Conference on Machine Learning*, volume 129, 81–96.

Agarwal, D.; Chatterjee, S.; Yang, Y.; and Zhang, L. 2015. Constrained Optimization for Homepage Relevance. In *Proceedings of the 24th International Conference on World Wide Web Companion*, 375–384.

Audsley, N. C. 1991. *Optimal priority assignment and feasibility of static priority tasks with arbitrary start times*. Citeseer.

Audsley, N. C. 2001. On priority assignment in fixed priority scheduling. *Inf. Process. Lett.*, 79(1): 39–44.

Bello, I.; Pham, H.; Le, Q. V.; Norouzi, M.; and Bengio, S. 2017. Neural Combinatorial Optimization with Reinforcement Learning. In *Proceedings of the 5th International Conference on Learning Representations (ICLR)*.

Blondel, M.; Teboul, O.; Berthet, Q.; and Djolonga, J. 2020. Fast Differentiable Sorting and Ranking. In *Proceedings of the 37th International Conference on Machine Learning (ICML)*, volume 119, 950–959.

Burges, C. J. C.; Shaked, T.; Renshaw, E.; Lazier, A.; Deeds, M.; Hamilton, N.; and Hullender, G. N. 2005. Learning to rank using gradient descent. In *Proceedings of the 22nd International Conference on Machine Learning (ICML)*, volume 119, 89–96.

Cao, Z.; Qin, T.; Liu, T.; Tsai, M.; and Li, H. 2007. Learning to rank: from pairwise approach to listwise approach. In *Proceedings of the 24th International Conference on Machine Learning (ICML)*, volume 227, 129–136.

Chen, T.; Moreau, T.; Jiang, Z.; Zheng, L.; Yan, E. Q.; Shen, H.; Cowan, M.; Wang, L.; Hu, Y.; Ceze, L.; Guestrin, C.; and Krishnamurthy, A. 2018. TVM: An Automated End-to-End Optimizing Compiler for Deep Learning. In *Proceedings of the 13th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 578–594.

Chu, P. C.; and Beasley, J. E. 1998. A genetic algorithm for the multidimensional knapsack problem. *Journal of heuristics*, 4(1): 63–86.

Davis, R. I.; and Burns, A. 2011. Improved priority assignment for global fixed priority pre-emptive scheduling in multiprocessor real-time systems. *Real Time Syst.*, 47(1): 1–40.

Davis, R. I.; Cucu-Grosjean, L.; Bertogna, M.; and Burns, A. 2016. A review of priority assignment in real-time systems. *J. Syst. Archit.*, 65: 64–82.

Glover, F. W. 1989. Tabu Search - Part I. *INFORMS J. Comput.*, 1(3): 190–206.

Grover, A.; Wang, E.; Zweig, A.; and Ermon, S. 2019. Stochastic Optimization of Sorting Networks via Continuous Relaxations. In *Proceedings of the 7th International Conference on Learning Representations (ICLR)*.

Gu, S.; and Hao, T. 2018. A pointer network based deep learning algorithm for 0-1 Knapsack Problem. In *Proceedings of the 10th International Conference on Advanced Computational Intelligence (ICACI)*, 473–477.

Guan, N.; Stigge, M.; Yi, W.; and Yu, G. 2009. New Response Time Bounds for Fixed Priority Multiprocessor Scheduling. In *Proceedings of the 30th IEEE Real-Time Systems Symposium (RTSS)*, 387–397.

Gumbel, E. 1954. The maxima of the mean largest value and of the range. *The Annals of Mathematical Statistics*, 76–84.

Gupta, R.; Liang, G.; Tseng, H.; Vijay, R. K. H.; Chen, X.; and Rosales, R. 2016. Email Volume Optimization at LinkedIn. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 97–106.

Hinton, G. E.; Vinyals, O.; and Dean, J. 2015. Distilling the Knowledge in a Neural Network. *ArXiv*, abs/1503.02531.

Jang, E.; Gu, S.; and Poole, B. 2017. Categorical Reparameterization with Gumbel-Softmax. In *Proceedings of the 5th International Conference on Learning Representations (ICLR)*.

Jiao, X.; Yin, Y.; Shang, L.; Jiang, X.; Chen, X.; Li, L.; Wang, F.; and Liu, Q. 2020. TinyBERT: Distilling BERT for Natural Language Understanding. In *Proceedings of the Association for Computational Linguistics (EMNLP)*, 4163–4174.

Kim, Y.; and Rush, A. M. 2016. Sequence-Level Knowledge Distillation. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 1317–1327.

Kool, W.; van Hoof, H.; and Welling, M. 2019. Attention, Learn to Solve Routing Problems! In *Proceedings of the 7th International Conference on Learning Representations, (ICLR)*.

Kool, W.; van Hoof, H.; and Welling, M. 2020. Estimating Gradients for Discrete Random Variables by Sampling without Replacement. In *Proceedings of the 8th International Conference on Learning Representations (ICLR)*.

Lee, H.; Lee, J.; Yeom, I.; and Woo, H. 2020. Panda: Reinforcement Learning-Based Priority Assignment for Multi-Processor Real-Time Scheduling. *IEEE Access*, 8: 185570–185583.

Maddison, C. J.; Tarlow, D.; and Minka, T. 2014. A* sampling. *arXiv preprint arXiv:1411.0030*.

Mao, H.; Alizadeh, M.; Menache, I.; and Kandula, S. 2016. Resource Management with Deep Reinforcement Learning.

In *Proceedings of the 15th ACM Workshop on Hot Topics in Networks*, 50–56.

Mao, H.; Schwarzkopf, M.; Venkatakrishnan, S. B.; Meng, Z.; and Alizadeh, M. 2019. Learning scheduling algorithms for data processing clusters. In *Proceedings of the ACM Special Interest Group on Data Communication*, 270–288.

Mirhoseini, A.; Pham, H.; Le, Q. V.; Steiner, B.; Larsen, R.; Zhou, Y.; Kumar, N.; Norouzi, M.; Bengio, S.; and Dean, J. 2017. Device Placement Optimization with Reinforcement Learning. In *Proceedings of the 34th International Conference on Machine Learning (ICML)*, volume 70, 2430–2439.

Perron, L.; and Furnon, V. 2019-7-19. OR-Tools.

Reddi, S. J.; Pasumarthi, R. K.; Menon, A. K.; Rawat, A. S.; Yu, F. X.; Kim, S.; Veit, A.; and Kumar, S. 2021. RankDistil: Knowledge Distillation for Ranking. In *Proceedings of the 24th International Conference on Artificial Intelligence and Statistics (AISTATS)*, volume 130, 2368–2376.

Rendle, S.; Freudenthaler, C.; Gantner, Z.; and Schmidt-Thieme, L. 2009. BPR: Bayesian Personalized Ranking from Implicit Feedback. In *Proceedings of the 25th Conference on Uncertainty in Artificial Intelligence*, 452–461.

Shojaei, H.; Ghamarian, A.; Basten, T.; Geilen, M.; Stuijk, S.; and Hoes, R. 2009. A parameterized compositional multi-dimensional multiple-choice knapsack heuristic for CMP run-time management. In *Proceedings of the 46th Annual Design Automation Conference*, 917–922.

Silver, D.; Schrittwieser, J.; Simonyan, K.; Antonoglou, I.; Huang, A.; Guez, A.; Hubert, T.; Baker, L.; Lai, M.; Bolton, A.; et al. 2017. Mastering the game of go without human knowledge. *nature*, 550(7676): 354–359.

Tang, J.; and Wang, K. 2018. Ranking Distillation: Learning Compact Ranking Models With High Performance for Recommender System. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD)*, 2289–2298.

Touvron, H.; Cord, M.; Douze, M.; Massa, F.; Sablayrolles, A.; and Jegou, H. 2021. Training data-efficient image transformers & distillation through attention. In *Proceedings of the 38th International Conference on Machine Learning (ICML)*, volume 139, 10347–10357.

Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A. N.; Kaiser, L.; and Polosukhin, I. 2017. Attention is All you Need. In *Proceedings of the advances in Neural Information Processing Systems (NeurIPS)*, 5998–6008.

Vinyals, O.; Babuschkin, I.; Czarnecki, W. M.; Mathieu, M.; Dudzik, A.; Chung, J.; Choi, D. H.; Powell, R.; Ewalds, T.; Georgiev, P.; Oh, J.; Horgan, D.; Kroiss, M.; Danihelka, I.; Huang, A.; Sifre, L.; Cai, T.; Agapiou, J. P.; Jaderberg, M.; Vezhnevets, A. S.; Leblond, R.; Pohlen, T.; Dalibard, V.; Budden, D.; Sulsky, Y.; Molloy, J.; Paine, T. L.; Gülçehre, Ç.; Wang, Z.; Pfaff, T.; Wu, Y.; Ring, R.; Yogatama, D.; Wünsch, D.; McKinney, K.; Smith, O.; Schaul, T.; Lillicrap, T. P.; Kavukcuoglu, K.; Hassabis, D.; Apps, C.; and Silver, D. 2019. Grandmaster level in StarCraft II using multi-agent reinforcement learning. *Nat.*, 575(7782): 350–354.

Williams, R. J. 1992. Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning. *Mach. Learn.*, 8: 229–256.

Ykman-Couvreur, C.; Nollet, V.; Catthoor, F.; and Corporaal, H. 2006. Fast Multi-Dimension Multi-Choice Knapsack Heuristic for MP-SoC Run-Time Management. In *Proceedings of the 4th international Symposium on System-on-Chip (SoC)*, 1–4.

Zhang, W.; and Dietterich, T. G. 1995. A Reinforcement Learning Approach to job-shop Scheduling. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI)*, 1114–1120.

Zoph, B.; and Le, Q. V. 2017. Neural Architecture Search with Reinforcement Learning. In *Proceedings of the 5th International Conference on Learning Representations (ICLR)*.