A Loss Decomposition

Proofs of proposition 1: We consider the loss $L_j^i(\Theta^i)$ for task T_j^i at time t_2 and take the first order Taylor expansion for $t_1 < t_2$:

$$\begin{aligned} L_j^i(\Theta^i(t_2)) &= L_j^i(\Theta(t_2)) = L_j^i(\Theta(t_1)) + \nabla^T L_j^i(\Psi^i(t_1))(\Theta(t_2) - \Theta(t_1)) \\ &= L_j^i(\Theta^i(t_1)) - \nabla^T L_j^i(\Psi^i(t_1)) \sum_{t=t_1}^{t_2} \eta_t \nabla_{\Theta^i} L(\Theta(t)) \end{aligned} \tag{9}$$

where $\Psi^i(t_1)$ is some vector lying between $\Theta^i(t_1)$ and $\Theta^i(t_2)$. Suppose task T_j^i is selected for c_j^i times between time step t_1 and t_2, we then study the term in the case of all tasks are selected at time step t:

$$\begin{aligned} \nabla_{\Theta^i} L(\Theta(t)) &= (\nabla^T_{\theta^{\text{share}}} L(\Theta(t)), \nabla^T_{\theta^i} L(\Theta(t)))^T \\ &= (\sum_{p=1}^{K} \sum_{q=1}^{n_p} \nabla^T_{\theta^{\text{share}}} L_q^p(\Theta^p(t)), \sum_{j=1}^{n_i} \nabla^T_{\theta^i} L_j^i(\Theta(t)))^T \\ &= \underbrace{(\nabla^T_{\theta^{\text{share}}} L_j^i(\Theta^i(t)), \nabla^T_{\theta^i} L_j^i(\Theta(t)))^T}_{(a)\text{ gradients of training task } T_j^i} + \underbrace{(\sum_{\substack{q=1 \\ q\neq j}}^{n_i} \nabla^T_{\theta^{\text{share}}} L_q^i(\Theta^i(t)), \sum_{\substack{q=1 \\ q\neq j}}^{n_i} \nabla^T_{\theta^i} L_q^i(\Theta(t)))^T}_{(b)\text{ gradients of training task } \{T_q^i, q \neq i\}} \\ &\quad + \underbrace{(\sum_{\substack{p=1 \\ p\neq i}}^{K} \sum_{q=1}^{n_p} \nabla^T_{\theta^{\text{share}}} L_q^p(\Theta^p(t)), \mathbf{0})^T}_{(c)\text{ gradients of training task } \{T_q^p, p \neq i\}} \\ &= \nabla L_j^i(\Theta^i(t)) + \sum_{\substack{q=1 \\ q\neq j}}^{n_i} \nabla L_q^i(\Theta^i(t)) + (\sum_{\substack{p=1 \\ p\neq i}}^{K} \sum_{q=1}^{n_p} \nabla^T_{\theta^{\text{share}}} L_q^p(\Theta^p(t)), \mathbf{0})^T. \end{aligned} \tag{10}$$

The terms $(a), (b)$ and (c) in Eq. 10 mean the gradients leading by the training of task T_j^i, the same kind of COP $\{T_q^i, q \neq j\}$ and other kinds of COPs $\{T^p, p \neq i\}$, respectively. After combining Eq. 9 and 10, we obtain

$$\begin{aligned} & L_j^i(\Theta^i(t_2)) - L_j^i(\Theta^i(t_1)) \\ &= \underbrace{-(\nabla^T L_j^i(\Psi^i(t_1)) \sum_{t=t_1}^{t_2} \mathbb{1}(a_t = T_j^i) \eta_t \nabla L_j^i(\Theta^i(t))}_{(a)\text{ effects of training task } T_j^i\text{: } e_j^i(t_1 \to t_2)} + \underbrace{\nabla^T L_j^i(\Psi^i(t_1)) \sum_{\substack{q=1 \\ q\neq j}}^{n_i} \sum_{t=t_1}^{t_2} \mathbb{1}(a_t = T_q^i) \eta_t \nabla L_q^i(\Theta^i(t))}_{(b)\text{ effects of training task } \{T_q^i, q \neq j\}\text{: } \{e_q^i((t_1 \to t_2)), q \neq j\}} \\ &\quad + \underbrace{\nabla^T_{\theta^{\text{share}}} L_j^i(\Psi^i(t_1)) \sum_{\substack{p=1 \\ p\neq i}}^{K} \sum_{q=1}^{n_p} \sum_{t=t_1}^{t_2} \mathbb{1}(a_t = T_q^p) \eta_t \nabla^T_{\theta^{\text{share}}} L_q^p(\Theta^p(t)))}_{(c)\text{ effects of training task } \{T_q^p, p \neq i\}\text{:} \{e_q^p(t_1 \to t_2), q=1,2,\ldots,n_p, p \neq i\}}, \end{aligned} \tag{11}$$

where $\mathbb{1}(a_t = T_j^i)$ is the indicator function which is introduced here because we only select one task at each time step, taking 1 if selecting task T_j^i at time step t, 0 otherwise. □

Adam optimizer [45] is more widely used and popular in practice than standard gradient descent . Accordingly, we derive the loss decomposition for Adam optimizer in a manner consistent with the

previous method. We first summarize the update rule of Adam as follows:

$$\begin{aligned}\Theta(t) &= \Theta(t-1) + \alpha \frac{\sqrt{\sum_{i=1}^{t-1} \beta_2^{t-i}}}{\sum_{i=1}^{t-1} \beta_1^{t-i}} \frac{\sum_{i=1}^{t} \beta_1^{t-i} g_i}{\sqrt{\sum_{i=1}^{t} \beta_2^{t-i} ||g_i||^2 + \epsilon}} \\ &= \Theta(t-1) + \eta_t \sum_{i=1}^{t} \beta_1^{t-i} g_i\end{aligned}$$

where $g_i = \nabla J(\Theta_{i-1})$ and $g_0 = \mathbf{0}, \eta_t = \frac{\sqrt{\sum_{i=1}^{t-1} \beta_2^{t-i}}}{\sum_{i=1}^{t-1} \beta_1^{t-i}} \frac{1}{\sqrt{\sum_{i=1}^{t} \beta_2^{t-i} ||g_i||^2 + \epsilon}}, \eta_i, i = 1, 2$ are exponential average parameters for the first and second order gradients. Our assumption is that sharing the second moment term correction for all tasks can be easily implemented by using a single optimizer during training.

Given that the update is predicated on the optimization trajectory's history, we can use comparable calculations in gradient descent to infer Adam's contribution breakdown. Starting at the same point:

$$\begin{aligned}L_j^i(\Theta^i(t_2)) &= L_j^i(\Theta^i(t_1)) + \nabla^T L_j^i(\Psi^i(t_1))(\Theta^i(t_2) - \Theta^i(t_1)) \\ &= L_j^i(\Theta^i(t_1)) - \nabla^T L_j^i(\Psi^i(t_1)) \sum_{t=t_1}^{t_2} \eta_t \sum_{k=1}^{t} \beta_1^{t-k} \nabla L(\Theta^i(k-1)),\end{aligned}$$

then taking Eq. 10 into $\nabla L(\Theta^i(k-1))$, we have

$$\begin{aligned}&L_j^i(\Theta^i(t_2)) - L_j^i(\Theta^i(t_1)) \\ =& \underbrace{-(\nabla^T L_j^i(\Psi^i(t_1)) \sum_{t=t_1}^{t_2} \mathbb{1}(a_t = T_j^i) \eta_t \sum_{k=1}^{t} \beta_1^{t-k} \nabla L_j^i(\Theta^i(k-1))}_{(a) \text{ effects of training task } T_j^i\text{: } e_j^i(t_1 \to t_2)} \\ &+ \underbrace{\nabla^T L_j^i(\Psi^i(t_1)) \sum_{\substack{q=1 \\ q \neq j}}^{n_i} \sum_{t=t_1}^{t_2} \mathbb{1}(a_t = T_q^i) \eta_t \sum_{k=1}^{t} \beta_1^{t-k} \nabla L_q^i(\Theta^i(k-1))}_{(b) \text{ effects of training task } \{T_q^i, q \neq j\}\text{: } \{e_q^i((t_1 \to t_2)), q \neq j\}} \\ &+ \underbrace{\nabla_{\theta^{\text{share}}}^T L_j^i(\Psi^i(t_1)) \sum_{\substack{p=1 \\ p \neq i}}^{K} \sum_{q=1}^{n_p} \sum_{t=t_1}^{t_2} \mathbb{1}(a_t = T_q^p) \eta_t \sum_{k=1}^{t} \beta_1^{t-k} \nabla_{\theta^{\text{share}}} L_q^p(\Theta^p(k-1)))}_{(c) \text{ effects of training task } \{T_q^p, p \neq i\}\text{:} \{e_q^p(t_1 \to t_2), q=1,2,\ldots,n_p, p \neq i\}},\end{aligned} \tag{12}$$

Three similar parts are obtained finally.

B Problem Description

Traveling Salesman Problem (TSP) - The objective is to determine the shortest possible route that visits each location once and returns to the original location. In this study, we limit our consideration to the two-dimensional euclidean case, where the information for each location is presented as $(x_i, y_i) \in \mathbb{R}^2$ sampled from the unit square.

Vehicle Routing Problem (VRP) - The Capacitated VRP (CVRP) [46] consists of a depot node and several demand nodes. The vehicle begins and ends at the depot node, travels through multiple routes to satisfy all the demand nodes, and the total demand for each route must not exceed the vehicle capacity. The goal of the CVRP is to minimize the total cost of the routes while adhering to all constraints.

Orienteering Problem (OP) - The Orienteering Problem (OP) is a variant of the Traveling Salesman Problem (TSP). Instead of visiting all the nodes, the objective is to maximize the total prize of visited

nodes within a total distance constraint. Unlike the TSP and the Vehicle Routing Problem (VRP), the OP does not require selecting all nodes.

Knapsack Problem (KP) - The Knapsack Problem strives to decide which items with various weights and values to be placed into a knapsack with limited capacity fully. The objective is to attain the maximum total value of the selected items while not surpassing the knapsack's limit.

C Experimental Settings

Model structure - We adopt the same model structures as in POMO [4] to build our model. To train various COPs in a unified model, we use a separate MLP on top of the model for each problem, which we call *Header*. This header facilitates correlation of input features with different dimensions. For TSP, we use two-dimensional coordinates, $\{(x_i, y_i), i = 1, 2, ..., N\}$, as input, while CVRP and OP have additional constraints on customer demand and vehicle capacity, in addition to two-dimensional coordinates. Hence, their input dimensions are 3 and 3, respectively. Moreover, in OP, the prize is assigned based on the distance between the node and the depot node, following the setting in AM [3]. The KP takes two-dimensional inputs, $\{(w_i, v_i), i = 1, 2, ..., N\}$, with w_i and v_i representing the weight and value of each item, respectively. As such, we introduce four kinds of *Header* to embed features with different dimensions to 128. The embeddings obtained from the *Header* are then passed through a shared *Encoder*, composed of six encoder layers based on the Transformer [47]. Finally, we employ four type-specific *Decoder*s, one for each COP, to make decisions in a sequential manner. The shared *Encoder* has the bulk of the model's capacity because the *Header* and *Decoder* are lightweight 1-layer MLPs. Furthermore, when solving a specific COP, we only need to use the relevant *Encoder*, *Header*, and *Decoder* for evaluation. Since the model size is precisely the same, the inference time required is similar to that of single-task learning.

Hyperparameters - In each epoch, we process a total of 100×1000 instances with a batch size of 512. The POMO size is equal to the problem scale, except for KP-200, where it is 100. We optimize the model using Adam [45] with a learning rate of 1e-4 and weight decay of 1e-6. The training of the model involves 1000 epochs in the standard setting. The learning rate is decreased by 1e-1 at the 900th epoch. During the first epoch, we use the bandit algorithm to explore at the beginning of the training process. We then collect gradient information by updating the bandit algorithm with every 12 batches of data. The model is trained using 8 Nvidia Tesla A100 GPUs in parallel, and the evaluations are done on a single NVIDIA GeForce RTX 3090.

Approximation of gradients - We use $\nabla^T L_j^i(\Psi^i(t_1))$ to calculate the reward information for training task T_j^i. To approximate it, we use the average gradient from the equation:

$$\frac{1}{\sum_{t=t_1}^{t_2} \mathbb{1}(a_t = T_j^i)} \sum_{t=t_1}^{t_2} \mathbb{1}(a_t = T_j^i) \nabla L_j^i(\Theta^i(t)).$$

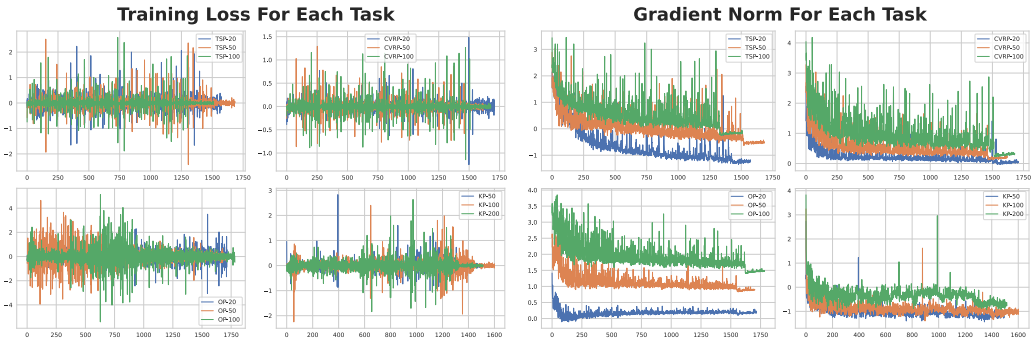
Another issue is the approximation of $\nabla L_q^i(\Theta^i(t))$ in Eq. 4 and $\nabla_{\theta^{\text{share}}} L_q^p(\Theta^p(t))$ in Eq. 5 when tasks T_q^i and T_q^p are not selected during the update interval. To obtain an approximation, we use the most recent gradient information collected from the last time they were selected to train. This approximation is necessary because training task T_j^i can change the values of Θ^i and θ^{share}, which can affect other training tasks. Considering all these changes is necessary to accurately measure the influences of training T_j^i on other tasks.

Bandit settings - We utilized the open-source repository [48] for implementing the bandit algorithms in this study with default settings.

D Loss and Gradient Norm of Each Task

One intuitive method of measuring the effect of training is to calculate the ratio of losses between adjacent training sessions. These ratios can be used to calculate training rewards for each corresponding task. However, as shown in Figure 5a, this method of calculating rewards is not effective because they are not sufficiently distinct to guide the training process properly.

Computing the inner products of corresponding gradients to analyze how training one task affects the others can lead to a misleading calculation of rewards and training process. Figure 5b visualizes



(a) Training loss for each task. (b) Gradient norm for each task.

Figure 5: Training loss and gradient norm for each task in the log-scale.

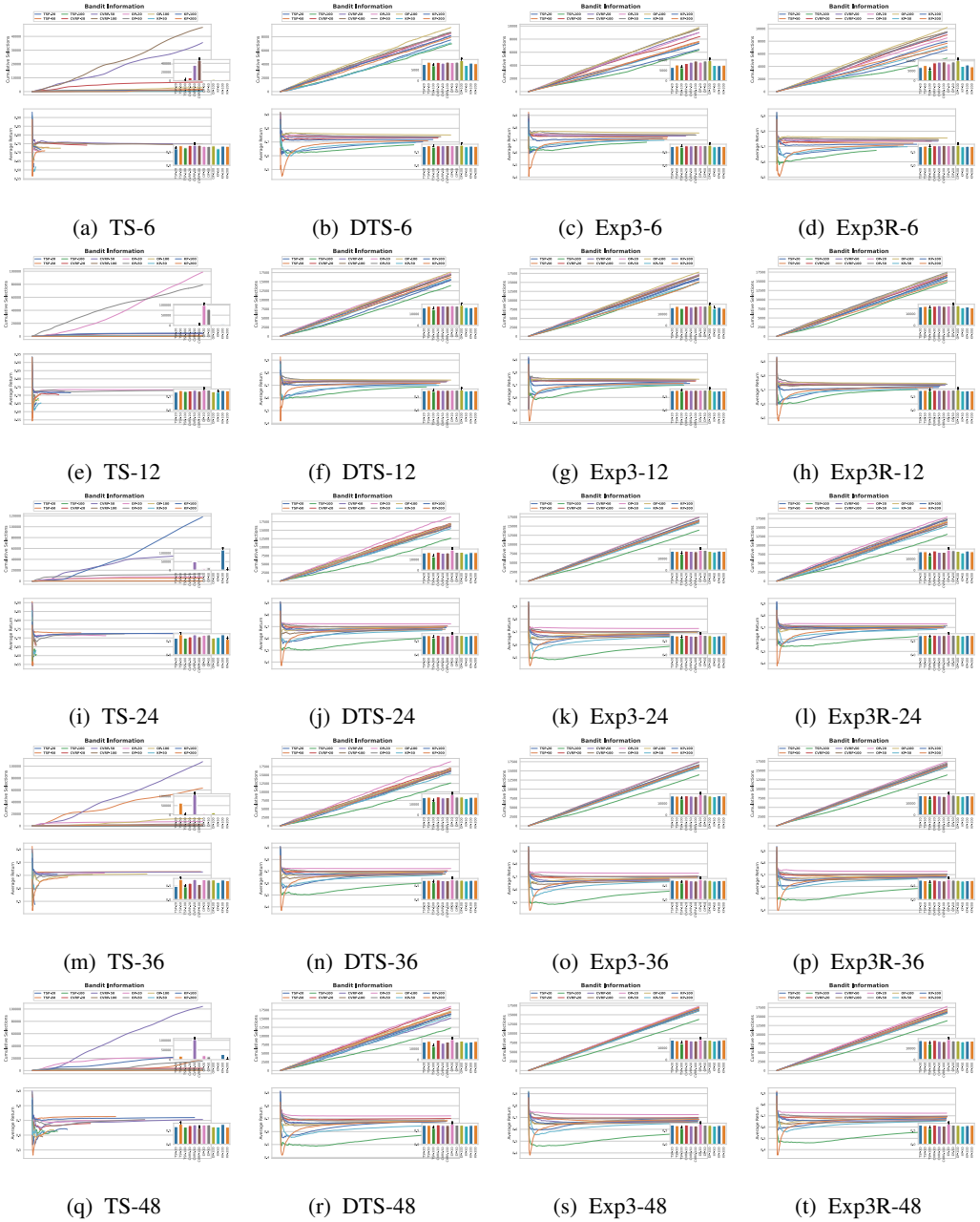
gradient norms for each task in the logarithmic scale. We observe that the gradient norms are not in the same scale, which becomes problematic when jointly training different COP types. In such cases, the rewards of certain COP types (such as CVRP in our experiments) may dominate the rewards of other types.

E Demonstration of the Bandit Algorithms

This section presents detailed information on various bandit algorithms, as shown in Fig. 6, including the selection count and average return for each task. It is evident that TS algorithm dominates in all 12 tasks, leading to poor performance on tasks where training is limited. In contrast, other bandit algorithms maintain balance across all tasks, resulting in better average results.

F Further Results on The Bandit Algorithm Selection and Update Frequency

In Section 4.3, we examine the impact of bandit algorithms and update frequency on 12 tasks, specifically on the average optimality gap. We also analyze the effect of these two factors on the influence matrix, which is presented in this section. For ease of understanding, a visual aid is included in Figure 7. By combining the results from Figure 3 and Figure 7, we can infer that influence matrices derived from DTS, Exp3, and Exp3R with an update frequency of 6 and 12 comply with the rule specified in Section 4.2. However, the TS algorithm disregards this rule due to its inability to handle adversaries and changing environments. Moreover, when the update frequency is increased, the approximation of the influence matrix is impaired due to the lazy update of bandit algorithms. As a result, utilizing the number of tasks as the update frequency appears to be a sound decision, as it not only improves performance but also enhances interpretability.



(a) TS-6 (b) DTS-6 (c) Exp3-6 (d) Exp3R-6

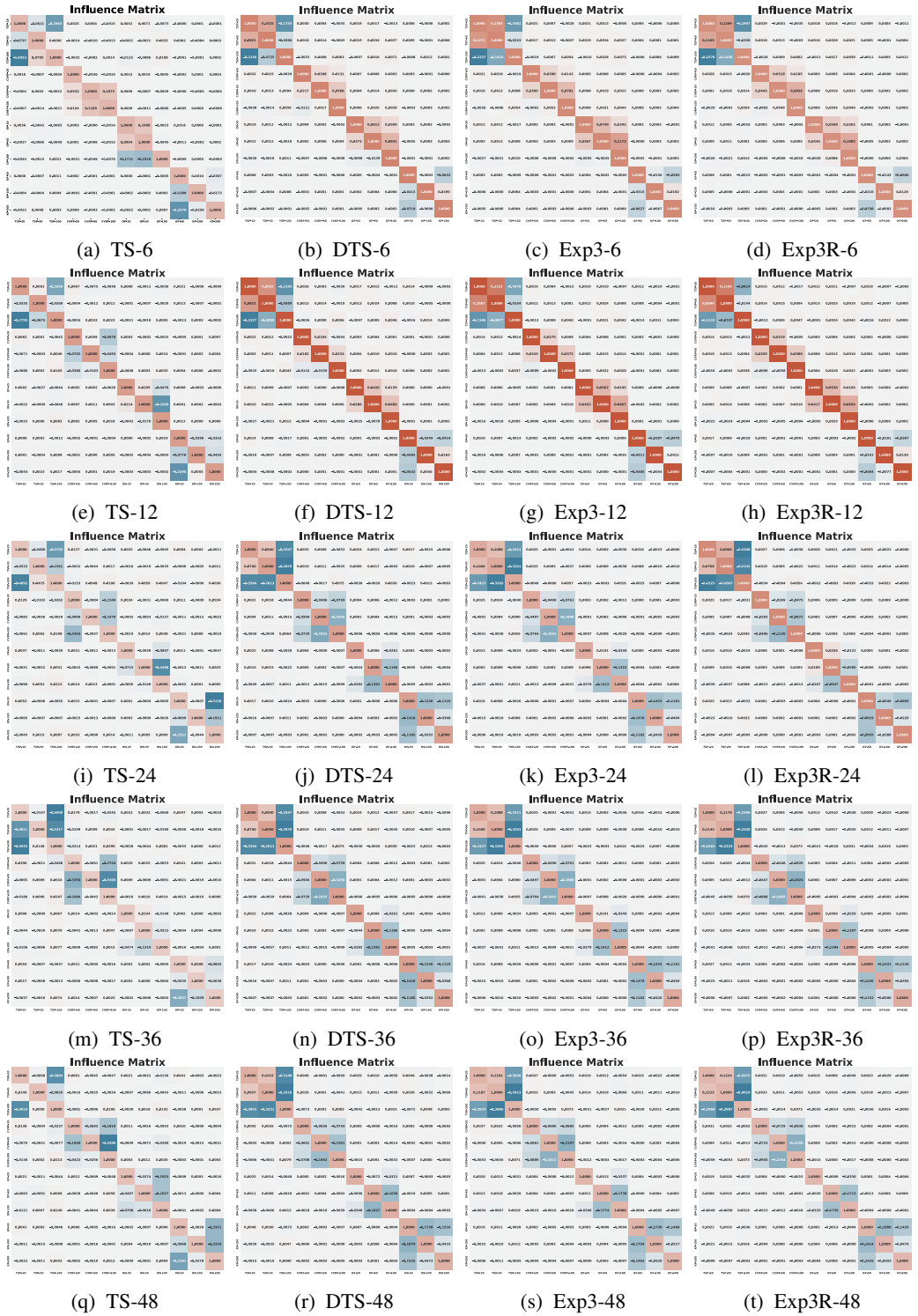
(e) TS-12 (f) DTS-12 (g) Exp3-12 (h) Exp3R-12

(i) TS-24 (j) DTS-24 (k) Exp3-24 (l) Exp3R-24

(m) TS-36 (n) DTS-36 (o) Exp3-36 (p) Exp3R-36

(q) TS-48 (r) DTS-48 (s) Exp3-48 (t) Exp3R-48

Figure 6: Further results of the bandit information. The caption of each subfigure "A-B" means the influence matrix obtained by algorithm A with update frequency B.



(a) TS-6 (b) DTS-6 (c) Exp3-6 (d) Exp3R-6

(e) TS-12 (f) DTS-12 (g) Exp3-12 (h) Exp3R-12

(i) TS-24 (j) DTS-24 (k) Exp3-24 (l) Exp3R-24

(m) TS-36 (n) DTS-36 (o) Exp3-36 (p) Exp3R-36

(q) TS-48 (r) DTS-48 (s) Exp3-48 (t) Exp3R-48

Figure 7: Further results of the influence matrix on the selection of bandit algorithms and update frequency. The caption of each subfigure "A-B" means the influence matrix obtained by algorithm A with update frequency B.