A Novel Method to Solve Neural Knapsack Problems

Duanshun Li [*1] **Jing Liu** [*2] **Dongeun Lee** [3] **Ali Seyedmazloom** [4] **Giridhar Kaushik** [4] **Kookjin Lee** [5] **Noseong Park** [6]

Abstract

0-1 knapsack is of fundamental importance across many fields. In this paper, we present a game-theoretic method to solve 0-1 knapsack problems (KPs) where the number of items (products) is large and the values of items are not predetermined but decided by an external value assignment function (e.g., neural network in our case) during the optimization process. While existing papers are interested in *predicting* solutions with neural networks for classical KPs whose objective functions are mostly linear functions, we are interested in *solving* KPs whose objective functions are neural networks. In other words, we choose a subset of items that maximizes the sum of the values predicted by neural networks. Its key challenge is how to optimize the neural network-based non-linear KP objective with a budget constraint. Our solution is inspired by game-theoretic approaches in deep learning, e.g., generative adversarial networks. After formally defining our two-player game, we develop an adaptive gradient ascent method to solve it. In our experiments, our method successfully solves two neural network-based non-linear KPs and conventional linear KPs with 1 million items.

1. Introduction

The 0-1 knapsack problem (KP), which chooses an optimal subset of items (products) that maximizes an objective function under a budget constraint, is an NP-hard problem that frequently occurs in real world applications. According to a study, KPs are one of the top-20 most popular problems (Kellerer et al., 2004; Skiena, 1999). Most of existing linear and some special non-linear KPs can be solved by various algorithms (Kellerer et al., 2004; Martello & Toth, 1990; Garey & Johnson, 1979). Some of them are classical combinatorial optimization algorithms, whereas other works rely on deep learning methods (Bello et al., 2016; Gu & Hao, 2018; Hertrich & Skutella, 2021). Those deep learning models *predict* linear KP solutions after being trained. Note that we use the term ‘predict’ instead of ‘solve’ because given a linear KP instance they predict its solution. However, they do not consider non-linear KPs and moreover, their scalabiliy is not satisfactory either. They consider at most hundreds of items whereas we consider up to a million of items in our experiments.

In this paper, we consider a more complicated KP definition whose objective function consists of neural networks, which we call *neural knapsack problems*. Our definition belongs to non-linear KPs and has many applications in deep learning. We propose a method to solve such complicated KPs on deep learning platforms such as TensorFlow — we do not predict solutions but solve.

However, one difficulty in designing such a constrained optimization technique on top of deep learning platforms is how to consider the *hard* budget constraint — we must not violate the budget limitation. As such, our key contribution is to design a gradient-based constrained optimization method, which we call *adaptive gradient ascent*. Our method is optimized to solve a max-min game formulation devised by us. It is noted that the equilibrium state of our max-min game is equivalent to the optimal feasible solution of KP that does not violate the budget limitation and yields the biggest objective value. While the existence of equilibrium does not necessarily imply its achievement (see Thm. (1) and the discussion afterward), our method works well in practice. As a matter of fact, exact algorithms are known only for a few special non-linear KPs as will be discussed in Sec. 2. In our case, neural network-based objectives can be arbitrary non-linear and non-convex functions whose global convergence theorems are not known (Boyd & Vandenberghe, 2004; Bretthauer & Shetty, 2002; Fomeni & Letchford, 2014).

[*]The first two authors equally contributed and were advised by Noseong Park when he was at George Mason University. [1]University of Alberta, Edmonton, AB, Canada [2]Walmart Labs., Reston, VA, USA [3]Texas AM University-Commerce, Commerce, TX, USA [4]George Mason University, Fairfax, VA, USA [5]Arizona State University, Tempe, AZ, USA [6]Yonsei University, Seoul, South Korea. Correspondence to: Noseong Park <noseong@yonsei.ac.kr>.

Proceedings of the 38th International Conference on Machine Learning, PMLR 139, 2021. Copyright 2021 by the author(s).

Class	Objective	Related Work
LKP	$\sum_i v_i x_i$	(Pisinger, 2005)
QKP	$\sum_i v_i x_i + \sum_{i,j} v_{i,j} x_i x_j$	(Fomeni & Letchford, 2014)
NKP	$o(x_1, \cdots, x_n)$ where o is convex for minimization or concave for maximization	(Bretthauer & Shetty, 2002)
MDKP	$\sum_i v_i x_i / \sum_i c_i x_i$	(Cohen & Katzir, 2008)

Table 1: Various linear and non-linear KP classes and their objective definitions

We evaluate our method with two main deep learning-based experiments and one more secondary experiment with a linear KP benchmark set. First, we show that the point cloud resampling (PCR) (The CGAL Project, 2019) can be formulated as a non-linear KP and compare our method with other existing PCR methods. Second, we show that improving the transductive inference of graph convolutional networks (GCNs) can also be formulated as a non-linear KP. Our method improves the transductive inference accuracy of a state-of-the-art GCN model (Gao et al., 2018). Overall, our contributions can be summarized as follows:

1. Our non-linear KP definitions are much more complicated than others as we use neural network-based objectives. See our review in Sec. 2.
2. In deep learning platforms, it is not straightforward to consider the budget constraint of KPs. To resolve this, we design a max-min game (whose equilibrium state is equivalent to the optimal KP solution) and solve it using our proposed adaptive gradient ascent method.
3. Our adaptive gradient ascent method guarantees a decrease in the total cost after one iteration if any cost overrun. Therefore, a series of updates can eventually address the cost overrun situation and produce a feasible KP solution.
4. We show that our proposed method can be used in various applications including but not limited to the two deep learning applications and one more benchmark linear KP experiment that we will introduce in our experimental evaluation section.

2. Related Work

The definition of KP is, given a cost c_i of item i for all $1 \leq i \leq n$, a total budget B, and a value assignment function $o : \{0,1\}^n \rightarrow \mathbb{R}$, to decide $x_i \in \{0,1\}$, which denotes if we choose item i or not, for all i such that the sum of the value assignments is maximized and the total cost, denoted $\sum_{i=1}^{n} c_i x_i$, is not larger than B.

For example, the quadratic KP, one of the most popular non-linear KP definitions, maximizes $\sum_i v_i x_i + \sum_{i,j} v_{i,j} x_i x_j$ where v_i is the value (profit) of item i and $v_{i,j}$ is the synergistic profit when selecting both items i and j (Fomeni & Letchford, 2014). The most general research was done by Bretthauser et al., where they considered convex (for minimization) or concave (for maximization) functions for o (Bretthauer & Shetty, 2002). However, they only considered relatively simple functions that are not based on neural networks. We summarize popular KP definitions in Table 1.

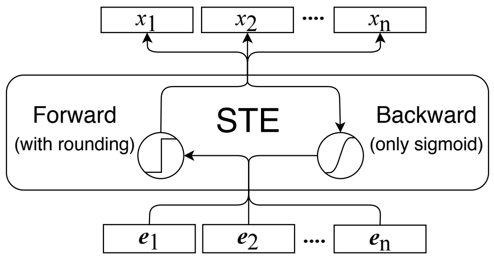
If o is fixed or has a specific property, one can design an efficient search method specific to o. In our case, however, o has no limitations and can be an arbitrary function (neural network). Therefore, it can be regarded as one of the hardest KPs to design a general search algorithm for.

Several other deep learning models *predict* the solutions of combinatorial optimization problems such as traveling salesman problem, minimum vertex cover, maximal independent set, and so forth (Vinyals et al., 2015; Bello et al., 2016; Zoph & Le, 2016; Gu & Hao, 2018; Khalil et al., 2017; Li et al., 2018; Gu & Yang, 2018; Yolcu & Poczos, 2019; Sato et al., 2019; Wu et al., 2019; Mena et al., 2018; Yu et al., 2018), which cannot be applied to solve our non-linear KPs because they solve unconstrained minimization problems or a specific type of problems different from our problem definition. Some works (Bello et al., 2016; Gu & Hao, 2018; Nomer et al., 2020) did experiments for linear KPs with at most hundreds of items. They did not consider non-linear KPs either.

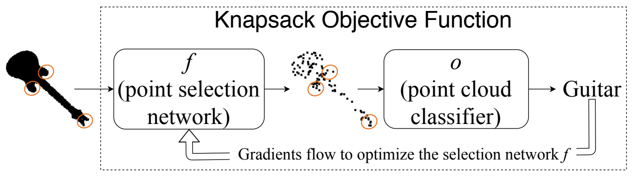
One more recent work is in (Hertrich & Skutella, 2021). Hertrich et al. showed that neural networks can predict linear KP solutions. Their main contribution is that they found a relation between the size of a KP instance to solve and the size of a neural network that can predict its solution.

3. Proposed Method

As discussed, o can be an application-dependent neural network in our case. We note that o requires a binary vector $\boldsymbol{x} = [x_1, x_2, \ldots, x_n]^\intercal$ which itself is a KP solution (boldface is used to denote vectors). Here we propose an application-agnostic neural network f generating the binary vector. Our neural network can be described by a function $f : \boldsymbol{\theta} \rightarrow \{0,1\}^n$, where $\boldsymbol{\theta}$ is a set of neural network parameters. In other words, our neural network f outputs a set of selected items (products) and does not require any inputs other than $\boldsymbol{\theta}$. The objective and constraint of KPs are



(a) Our item selection network f


Knapsack Objective Function
f (point selection network)
o (point cloud classifier)
Guitar
Gradients flow to optimize the selection network f

(b) PCR example

Figure 1: (a) We rely on the straight-through estimator (STE) or the pass-through estimator (PTE) to select items. The backward pass of STE, i.e., the pass with a sigmoid only, is differentiable. We directly feed e_i into the estimator. (b) Our PCR experiment example.

used to describe a training objective with which we optimize $\boldsymbol{\theta}$ using our novel gradient ascent method. As mentioned earlier, our neural network f does not predict, but solve KPs.

The architecture of f is shown in Fig. 1 (a). For each item i, we have a scalar parameter $e_i \in \mathbb{R}$ that will be optimized — in fact, these parameters are the only ones to optimize in our model, i.e., $\boldsymbol{\theta} = [e_1, e_2, \dots, e_n]$. The straight-through estimator (STE) or the pass-through estimator (PTE) produces $x_i \in \{0, 1\}$ from e_i, where $i = 1, 2, \dots, n$. In the straight-through estimator (STE),

$$x_i = \begin{cases} Round(Sigmoid(\tau e_i)), & \text{if forward-pass} \\ Sigmoid(\tau e_i), & \text{if backward-pass} \end{cases},$$

where $\tau \geq 1$ is a slope annealing parameter and we perform $\tau \leftarrow r^{\lfloor p/s \rfloor}\tau$ at epoch p with the change rate $r > 1$ and the step parameter s (Chung et al., 2016). *When τ becomes large enough, the slope of the sigmoid function becomes precipitous and close to the binary rounding function*. The pass-through estimator is defined in a simpler form and we refer to (Bengio et al., 2013). These two estimators are popular techniques to train binary stochastic neurons (Bengio et al., 2013). We note that *the differentiable backward pass is used for optimizing $\boldsymbol{\theta}$*.

We solve the following non-linear KP in Eq. (1) based on our proposed neural network f — we will describe shortly how we consider the budget constraint — and after optimizing $\boldsymbol{\theta}$, we can take $x_i \in \{0, 1\}$ to know which items have been selected with the forward pass of STE/PTE:

$$\begin{aligned} \max_{\boldsymbol{\theta}} \quad & o(f(\boldsymbol{\theta})) \\ \text{subject to} \quad & \boldsymbol{c}f(\boldsymbol{\theta}) \leq B, \end{aligned} \tag{1}$$

where $f(\boldsymbol{\theta}) = \boldsymbol{x}$, $\boldsymbol{c} = [c_1, c_2, \dots, c_n]$, and o is an application-dependent neural network. One can consider that Eq. (1) is based on a continuous relaxation of binary variables with STE/PTE minimizing the gap between the discrete and continuous variables incurred by the relaxation.

The objective function is a composite function of o and f, which makes our problem definition further complicated. The inner function f is application-agnostic and in any cases, we need it to produce the binary selection vector. The outer function o can be changed depending on applications. In our study, o is considered to be a pre-trained neural network[1] and our proposed neural network f produces the binary selection vector optimizing the outputs of o.

For instance, Fig. 1 (b) shows the objective function of PCR problem in our experiments, where o is a point cloud classification network. The solution by f optimizes the prediction by o so that selected points are still recognized as a guitar in the example. Note that the selected points successfully capture the key characteristics of the guitar. Therefore, we do not predict solutions but solve.

Complexity. The space complexity of f is $\mathcal{O}(n)$ where n is the number of items. Because of the space-efficient architecture, f incurs small GPU memory overhead. As f does not involve any matrix multiplications, its computational overhead is also $\mathcal{O}(n)$.

3.1. Game-Theoretic Approach

If we ignore the budget constraint, optimizing $\boldsymbol{\theta}$ to improve $o(\boldsymbol{x})$ can be done in TensorFlow — recall that the backward passes of STE and PTE are differentiable thus optimizing $\boldsymbol{\theta}$ with a gradient-based method is also feasible. However, we need to take into account the budget constraint in reality. To this end, we rely on the Karush–Kuhn–Tucker (KKT) condition-based approach, a popular method to solve both differentiable and non-differentiable non-linear optimization problems (Ruszczynski, 2006) — one can use subgradients for non-differentiable functions. Eq. (1) then can be rewritten as follows:

$$\max_{\boldsymbol{\theta}} \quad o(f(\boldsymbol{\theta})) - \lambda b(f(\boldsymbol{\theta})), \tag{2}$$

where $b(f(\boldsymbol{\theta})) = \boldsymbol{c}f(\boldsymbol{\theta}) - B = \boldsymbol{cx} - B$ is the cost overrun, and $\lambda \geq 0$ is the Lagrange multiplier.

[1] In some cases, both o and f can be trained together to find better solutions. For example, we use a pre-trained point cloud classification network in the first PCR experiment, whereas we train both a GCN and f together in the second experiment.

By properly setting λ, we can solve Eq. (2). If i) a closed-form of $\nabla o \circ f$ can be defined, ii) $o \circ f$ is concave, or iii) $o \circ f$ is one of some special non-concave functions, the optimal λ can be found (Bretthauer & Shetty, 2002; Wang et al., 2019). In our case, however, none of those three conditions can be assumed because o is an arbitrary neural network. To resolve this issue, we use the following max-min game to find a stable λ and an optimized $\boldsymbol{\theta}$:

$$\max_{\boldsymbol{\theta}} \min_{\lambda \geq 0} \quad o(f(\boldsymbol{\theta})) - \lambda b(f(\boldsymbol{\theta})) + \delta \lambda^2, \tag{3}$$

where $\delta\lambda^2 \geq 0$ is a regularization term to prevent λ from getting too large. When there is no such regularization term, the optimal λ^* for the inner minimization is ill-defined as follows: $\lambda^* = 0$, if $b(f(\boldsymbol{\theta})) \leq 0$; $\lambda^* \to \infty$, if $b(f(\boldsymbol{\theta})) > 0$. In particular, $\lambda^* \to \infty$ is problematic when the regularization term $\delta\lambda^2$ is missing. With our regularization, however, the optimal λ^* can be defined as follows:

Lemma 1. *Given a fixed $\boldsymbol{\theta}$, $\lambda^* = \max\left\{0, \frac{b(f(\boldsymbol{\theta}))}{2\delta}\right\}$ minimizes Eq.* (3).

Proof. Eq. (3) is a parabolic function ($\delta > 0$) w.r.t. λ after fixing $\boldsymbol{\theta}$; thus its global minimum is where the following partial derivative becomes zero.

$$\frac{\partial(o(f(\boldsymbol{\theta})) - \lambda b(f(\boldsymbol{\theta})) + \delta\lambda^2)}{\partial\lambda} = -b(f(\boldsymbol{\theta})) + 2\delta\lambda$$

Thus, $-b(f(\boldsymbol{\theta})) + 2\delta\lambda = 0$ yields $\lambda = \frac{b(f(\boldsymbol{\theta}))}{2\delta}$. By the constraint that $\lambda \geq 0$, $\lambda^* = \max\left\{0, \frac{b(f(\boldsymbol{\theta}))}{2\delta}\right\}$. □

After replacing λ with λ^*, Eq. (3) can be rewritten as follows:

$$\max_{\theta} o(f(\boldsymbol{\theta})) - \frac{\max\{0, b(f(\boldsymbol{\theta}))\}^2}{4\delta}. \tag{4}$$

Or equivalently,

$$\max_{\theta} o(f(\boldsymbol{\theta})) - \frac{1}{2}\beta R(b(f(\boldsymbol{\theta})))^2, \tag{5}$$

where $\beta = \frac{1}{2\delta}$, and the function $R : \mathbb{R} \to \mathbb{R}^+$ denotes the rectifier.

We solve Eq. (5) instead of Eq. (3) as the inner minimization is now eliminated, which is the main reason why we formulate Eq. (4) with the quadratic regularization term. Eq. (5) has the squared cost overrun penalty, which greatly drives the optimization process toward the zero cost overrun point — note that in KPs, optimal solutions are achieved when $b(f(\boldsymbol{\theta})) \lesssim 0$.

The rationale behind the proposed max-min game between $\boldsymbol{\theta}$ and λ is that $\boldsymbol{\theta}$ tries to maximize the objective while λ tries to suppress the selection of items to minimize the cost overrun term. However, by adding the regularization we try to limit λ within a reasonable range. Therefore, it will easily converge to a balancing point between the objective and the budget constraint.

Theorem 1. *The equilibrium state of the proposed max-min game corresponds to the optimal solution of Eq.* (1) *with a proper setting of β.*

Proof. In this theorem, we prove that $\boldsymbol{\theta}^*$ maximizes Eq. (5) if and only if it is the optimal KP solution of Eq. (1), with a proper setting of β.

Since Eq. (5) is equivalent to Eq. (3) with the optimal form of λ, we first prove that the optimal solution of the input KP instance maximizes Eq. (5).

Let $\boldsymbol{x}^* = f(\boldsymbol{\theta}^*)$ be the optimal KP solution. The optimal KP solution maximizes $o(f(\boldsymbol{\theta}^*))$ with $b(f(\boldsymbol{\theta}^*)) \leq 0$ (and as a result, $\beta R(b(f(\boldsymbol{\theta}^*)))^2 = 0$ by its definition). Therefore, we can make any $\boldsymbol{\theta}$ with $b(f(\boldsymbol{\theta})) > 0$ and $o(f(\boldsymbol{\theta})) > o(f(\boldsymbol{\theta}^*))$ sub-optimal in Eq. (5) by setting $\beta > \frac{2\left(o(f(\boldsymbol{\theta})) - o(f(\boldsymbol{\theta}^*))\right)}{b(f(\boldsymbol{\theta}))^2}$. In conclusion, only $\boldsymbol{\theta}^*$ maximizes Eq. (5) if β is large enough.

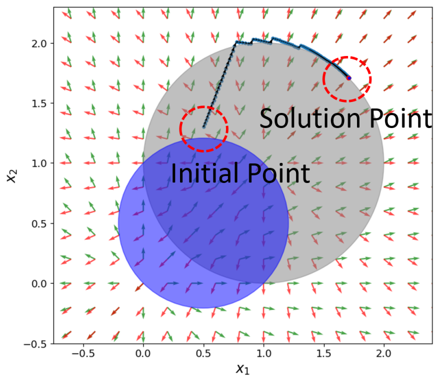
We then prove that the optimal solution of Eq. (5) also yields the optimal KP solution with the same configuration of β. If $\beta > \frac{2\left(o(f(\boldsymbol{\theta})) - o(f(\boldsymbol{\theta}^*))\right)}{b(f(\boldsymbol{\theta}))^2}$, we already have seen that any $\boldsymbol{\theta}$ with $b(f(\boldsymbol{\theta})) > 0$ and $o(f(\boldsymbol{\theta})) > o(f(\boldsymbol{\theta}^*))$ cannot be the optimal solution of Eq. (5). This implies that only $\boldsymbol{\theta}^*$ maximizes Eq. (5) and according to its definition, $\boldsymbol{\theta}^*$ can yield the optimal KP solution $\boldsymbol{x}^*$. □

The above equilibrium theorem shows i) the correctness of formulating our problem as a two-player game and ii) the importance of deciding β. Therefore, we propose the adaptive gradient ascent algorithm that dynamically controls β in the next subsection. Nevertheless, it should be noted that the equilibrium state is not always achievable even if it is well defined, which is also the case in generative adversarial networks (Goodfellow et al., 2014; Arjovsky & Bottou, 2017).

3.2. Adaptive Gradient Ascent Method

Because maximizing $L = o(f(\boldsymbol{\theta})) - \frac{1}{2}\beta R(b(f(\boldsymbol{\theta})))^2$, we need to use the gradient ascent method with $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \gamma\nabla L$, where $\gamma > 0$ is a learning rate. When $b(f(\boldsymbol{\theta})) \leq 0$, i.e., no cost overrun, we do not need to control β because of the rectifier R. However, we simply set $\beta = 0$ to enable more aggressive search for the KP objective without needing to consider the budget constraint.

When $b(f(\boldsymbol{\theta})) > 0$, however, we need to enforce that its steepest gradient ascent direction reduces the total cost. For



Algorithm 1 Adaptive gradient ascent

```
Input: $k$, $\gamma$
Output: $\boldsymbol{x}$
1  $tolerance \leftarrow k$; $\beta \leftarrow 0$; $best \leftarrow 0$; Initialize $\boldsymbol{\theta}$;
2  while not converged do
3      $\bar{\boldsymbol{\theta}} \leftarrow$ a $\xi$ portion of $\boldsymbol{\theta}$; /* Mini-batch selection */
4      $\bar{\boldsymbol{\theta}} \leftarrow \bar{\boldsymbol{\theta}} + \gamma\nabla L$; /* Gradient ascent */
5      $\boldsymbol{x} \leftarrow f(\boldsymbol{\theta})$; /* Use the forward-pass of STE/PTE */
6      if $b(\boldsymbol{x}) > 0$ or $o(\boldsymbol{x}) \leq best$ then
7          $tolerance \leftarrow tolerance - 1$;
8      else
9          $tolerance \leftarrow k$; $best \leftarrow o(\boldsymbol{x})$;
10     if $tolerance \leq 0$ then break;
11 return $\boldsymbol{x}$;
```

Figure 2: To show the efficacy of our adaptive gradient ascent, we maximize $x_1^2+x_2^2$ with a constraint $(x_1-1)^2+(x_2-1)^2 \leq 1$ from an initial point — note that we do not use our item selection network f but directly optimize $x_1, x_2 \in \mathbb{R}$ in this example. The adaptive gradient ascent converges to the optimal solution. Gray circle means there is no cost overrun and blue circle means $\mathbf{b}' \cdot \mathbf{o}' < 0$, i.e., their directions are opposite to each other. Green arrow represents $\mathbf{o}'$ and red arrow $\mathbf{b}'$. Note that our algorithm pulls the trajectory into the gray circle as soon as there is any small cost overrun.

ease of notation in this section, we simply use $b(\boldsymbol{\theta})$ and $o(\boldsymbol{\theta})$ to represent $b(f(\boldsymbol{\theta}))$ and $o(f(\boldsymbol{\theta}))$, and use $\boldsymbol{b}'$ and $\boldsymbol{o}'$ to represent their gradients w.r.t. $\boldsymbol{\theta}$. Using these notations, the steepest gradient ascent direction of L w.r.t. $\boldsymbol{\theta}$, when $b(\boldsymbol{\theta}) > 0$, will be as follows:

$$\nabla L = \boldsymbol{o}' - \beta b(\boldsymbol{\theta})\boldsymbol{b}'.$$

The following theorem shows the condition of β that decreases the total cost after one gradient ascent update.

Theorem 2. *For any $\boldsymbol{\theta}$ with $b(\boldsymbol{\theta}) > 0$, $b(\boldsymbol{\theta}+\gamma\nabla L) < b(\boldsymbol{\theta})$ with a sufficiently small $\gamma > 0$ if and only if $\beta > \frac{\boldsymbol{b}'\cdot\boldsymbol{o}'}{b(\boldsymbol{\theta})\boldsymbol{b}'\cdot\boldsymbol{b}'}$.*

Proof. We first prove that the left-hand side of the theorem derives the right-hand side. From $b(\boldsymbol{\theta}+\gamma(\boldsymbol{o}'-\beta b(\boldsymbol{\theta})\boldsymbol{b}')) < b(\boldsymbol{\theta})$ with a sufficiently small $\gamma > 0$, we have

$$\lim_{\gamma\to 0^+} \frac{b(\boldsymbol{\theta}+\gamma(\boldsymbol{o}'-\beta b(\boldsymbol{\theta})\boldsymbol{b}')) - b(\boldsymbol{\theta})}{\gamma} < 0.$$

Similarly, if we assume $b(\boldsymbol{\theta}+\gamma(\boldsymbol{o}'-\beta b(\boldsymbol{\theta})\boldsymbol{b}')) > b(\boldsymbol{\theta})$ with a sufficiently small $\gamma < 0$, we have

$$\lim_{\gamma\to 0^-} \frac{b(\boldsymbol{\theta}+\gamma(\boldsymbol{o}'-\beta b(\boldsymbol{\theta})\boldsymbol{b}')) - b(\boldsymbol{\theta})}{\gamma} < 0.$$

γ should be positive (resp. negative) if gradient ascent (resp. descent). We mention both of $\gamma \to 0^+$ and $\gamma \to 0^-$ in this proof so our theorem is the case for both the gradient ascent and descent, which makes our proof rigorous. These indicate that the directional derivative of $b(\boldsymbol{\theta})$ along $\boldsymbol{o}' - \beta b(\boldsymbol{\theta})\boldsymbol{b}'$ is negative.

Following the definition of directional derivative, we have

$$\begin{aligned}\nabla_{\boldsymbol{o}'-\beta b(\boldsymbol{\theta})\boldsymbol{b}'} b(\boldsymbol{\theta}) &\equiv \nabla b(\boldsymbol{\theta}) \cdot \big(\boldsymbol{o}' - \beta b(\boldsymbol{\theta})\boldsymbol{b}'\big) \\ &\equiv \boldsymbol{b}' \cdot \big(\boldsymbol{o}' - \beta b(\boldsymbol{\theta})\boldsymbol{b}'\big).\end{aligned}$$

Thus, we obtain

$$\boldsymbol{b}' \cdot \big(\boldsymbol{o}' - \beta b(\boldsymbol{\theta})\boldsymbol{b}'\big) < 0.$$

Then,

$$\beta > \frac{\boldsymbol{b}'\cdot\boldsymbol{o}'}{b(\boldsymbol{\theta})\boldsymbol{b}'\cdot\boldsymbol{b}'}.$$

The opposite direction from the right-hand side to the left-hand side is straightforward by reversing the deduction process. □

In addition, the following theorem shows the condition that the objective value does not decrease, i.e., $o(\boldsymbol{\theta}+\gamma\nabla L) \geq o(\boldsymbol{\theta})$, after one gradient ascent update.

Theorem 3. *For any $\boldsymbol{\theta}$ with $b(\boldsymbol{\theta}) > 0$, $o(\boldsymbol{\theta}+\gamma\nabla L) \geq o(\boldsymbol{\theta})$ with a sufficiently small $\gamma > 0$ if and only if* $\beta \begin{cases} \geq \frac{\boldsymbol{o}'\cdot\boldsymbol{o}'}{b(\boldsymbol{\theta})\boldsymbol{b}'\cdot\boldsymbol{o}'}, & \text{if } \boldsymbol{b}'\cdot\boldsymbol{o}' < 0 \\ \leq \frac{\boldsymbol{o}'\cdot\boldsymbol{o}'}{b(\boldsymbol{\theta})\boldsymbol{b}'\cdot\boldsymbol{o}'}, & \text{if } \boldsymbol{b}'\cdot\boldsymbol{o}' > 0 \end{cases}.$

Proof. This can be proved following the same procedure as Thm. (**??**); but one should be careful at the last step considering the sign of the dot product. □

Note that in Thm. (3) we do not consider the case that $\boldsymbol{b}'\cdot\boldsymbol{o}' = 0$ because $o(\boldsymbol{\theta}+\gamma\nabla L) \geq o(\boldsymbol{\theta})$ is automatically satisfied.

It leads to the best outcome if we can achieve both $b(\boldsymbol{\theta} + \gamma\nabla L) < b(\boldsymbol{\theta})$ and $o(\boldsymbol{\theta} + \gamma\nabla L) \geq o(\boldsymbol{\theta})$ after one gradient ascent update. Therefore, we need to combine the two theorems. From the above two theorems, we can collectively have the following conditions:

$$\begin{cases} \beta \geq 0 & \text{, if } \boldsymbol{b}' \cdot \boldsymbol{o}' < 0 \\ \beta > 0 & \text{, if } \boldsymbol{b}' \cdot \boldsymbol{o}' = 0 \\ \frac{\boldsymbol{b}' \cdot \boldsymbol{o}'}{b(\boldsymbol{\theta})\boldsymbol{b}' \cdot \boldsymbol{b}'} < \beta \leq \frac{\boldsymbol{o}' \cdot \boldsymbol{o}'}{b(\boldsymbol{\theta})\boldsymbol{b}' \cdot \boldsymbol{o}'} & \text{, if } \boldsymbol{b}' \cdot \boldsymbol{o}' > 0 \end{cases}.$$

If $\boldsymbol{b}' \cdot \boldsymbol{o}' < 0$ in the above condition, then $\beta \geq 0$ satisfies both Thm. (2) and Thm. (3). However, if it is not the case that $\frac{\boldsymbol{b}' \cdot \boldsymbol{o}'}{b(\boldsymbol{\theta})\boldsymbol{b}' \cdot \boldsymbol{b}'} < \frac{\boldsymbol{o}' \cdot \boldsymbol{o}'}{b(\boldsymbol{\theta})\boldsymbol{b}' \cdot \boldsymbol{o}'}$, then the decrease in the cost (Thm. (2)) and the non-decrease in the objective (Thm. (3)) cannot be fulfilled at the same time, in which case we rely only on Thm. (2). Considering all these conditions, β can have one of the following three values:

$$\beta = \begin{cases} 0 & \text{, if } \boldsymbol{b}' \cdot \boldsymbol{o}' < 0 \\ \frac{\boldsymbol{b}' \cdot \boldsymbol{o}'}{2b(\boldsymbol{\theta})\boldsymbol{b}' \cdot \boldsymbol{b}'} + \frac{\boldsymbol{o}' \cdot \boldsymbol{o}'}{2b(\boldsymbol{\theta})\boldsymbol{b}' \cdot \boldsymbol{o}'} & \text{, if } (\boldsymbol{b}' \cdot \boldsymbol{o}' > 0) \wedge \\ & \left(\frac{\boldsymbol{b}' \cdot \boldsymbol{o}'}{b(\boldsymbol{\theta})\boldsymbol{b}' \cdot \boldsymbol{b}'} < \frac{\boldsymbol{o}' \cdot \boldsymbol{o}'}{b(\boldsymbol{\theta})\boldsymbol{b}' \cdot \boldsymbol{o}'} \right) \\ \frac{\boldsymbol{b}' \cdot \boldsymbol{o}'}{b(\boldsymbol{\theta})\boldsymbol{b}' \cdot \boldsymbol{b}'} + \epsilon & \text{, otherwise} \end{cases} \tag{6}$$

where $\epsilon > 0$ is a small positive value we can analytically decide by the following theorem.

Theorem 4. *There will be no cost overrun after one gradient ascent update of $\boldsymbol{\theta}$ if $\epsilon = \frac{1}{\gamma \boldsymbol{b}' \cdot \boldsymbol{b}'}$ and $b(\boldsymbol{\theta})$ is linear.*

Proof. Let $b(\boldsymbol{\theta} + \gamma\nabla L) = 0$, i.e., there is no cost overrun after one gradient ascent update of $\boldsymbol{\theta}$. $b(\boldsymbol{\theta} + \gamma\nabla L) = 0$ implies that $b(\boldsymbol{\theta}) + \gamma\nabla L \cdot \boldsymbol{b}' \equiv b(\boldsymbol{\theta}) + \gamma(\boldsymbol{o}' - \beta b(\boldsymbol{\theta})\boldsymbol{b}') \cdot \boldsymbol{b}' = 0$, if $b(\boldsymbol{\theta})$ is linear, because the directional derivative $\gamma\nabla L \cdot \boldsymbol{b}'$ represents the rate of changes in $b(\boldsymbol{\theta})$ after one update.[2]

Replacing β with $\frac{\boldsymbol{b}' \cdot \boldsymbol{o}'}{b(\boldsymbol{\theta})\boldsymbol{b}' \cdot \boldsymbol{b}'} + \epsilon$, we have

$$b(\boldsymbol{\theta}) + \gamma \left[\boldsymbol{o}' - \left(\frac{\boldsymbol{b}' \cdot \boldsymbol{o}'}{b(\boldsymbol{\theta})\boldsymbol{b}' \cdot \boldsymbol{b}'} + \epsilon \right) b(\boldsymbol{\theta})\boldsymbol{b}' \right] \cdot \boldsymbol{b}' = 0.$$

Then, we have $\epsilon = \frac{1}{\gamma \boldsymbol{b}' \cdot \boldsymbol{b}'}$. □

The final proposed adaptive gradient ascent method is shown in Alg. (1). Let $\boldsymbol{\theta} = \{e_1, e_2, \ldots, e_n\}$ be the set of the one-dimensional (scalar) parameters to learn. At line 3, we choose a ξ portion of $\boldsymbol{\theta}$ as a mini-batch $\bar{\boldsymbol{\theta}}$ which is updated at line 4.

[2] $b(\boldsymbol{\theta}) + \gamma\nabla L \cdot \boldsymbol{b}'$ becomes 0 if $b(\boldsymbol{\theta})$ is linear. Because we use STE/PTE, however, b is not linear but concave w.r.t. $\boldsymbol{\theta}$, although it is a linear w.r.t. $\boldsymbol{x}$. As a heuristic, however, we use the derived ϵ and it sometimes requires more than one gradient ascent update. In any cases, however, the cost overrun will decrease. Our experiments show that it works well in practice.

Note that all deep learning platforms support if-else conditional statements; thus *L with β in Eq.* (6) *can be directly programmed*, and there is no need to control β from outside. Therefore, β is implicitly controlled during the update at line 4. Finally, line 10 is to check the early-stopping condition. If there are no improvements in the past k epochs, we stop the process. In Fig. 2, we show an example of optimizing a simple function with the proposed method.

4. Experiments

We test our proposed method in two deep learning experiments, resampling point clouds, and transductive inferences of GCNs, and one more benchmark linear KP experiment. We use Ubuntu 18.04 LTS, Python 3.6.6, NVIDIA Driver 417.22, CUDA 10, TensorFlow 1.14.0, Numpy 1.14.5, Scipy 1.1.0, and machines with Intel Core i9 CPU and NVIDIA RTX 2080 Ti.

4.1. Point Cloud Resampling

Point clouds are usually generated by a large set of collected (scanned) points on the external surface of objects from 3D scanners. Point cloud resampling (PCR) is to choose a subset of scanned points to reduce the space and time overheads of point processing algorithms. However, there are no well-defined metrics to evaluate and perform resampling. Each resampling algorithm relies on its own heuristic method. To this end, we solve the KP problem in Eq. (1) where o includes a point cloud classification model and B is the number of desired points in ratio — i.e., find a B portion of points that maximize the quality of resampling. We introduce our formal problem definition as follows:

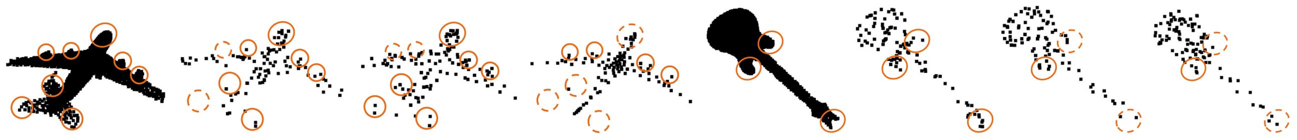
$$\begin{aligned} \min_{\boldsymbol{\theta}} \quad & \sigma(\ell, m(\boldsymbol{\theta}_m, \mathbf{P} \otimes f(\boldsymbol{\theta}))) \\ \text{subject to} \quad & \frac{\mathbf{1} \cdot f(\boldsymbol{\theta})}{n} \leq B, \end{aligned} \tag{7}$$

where $\mathbf{P} \in \mathbb{R}^{n \times 3}$ is a set (matrix) of n 3-dimensional points scanned from one object; m is a point cloud classification model, in particular PointNet (Qi et al., 2016) in our experiments; $\boldsymbol{\theta}_m$ is its model parameters; $f(\boldsymbol{\theta})$ outputs a column vector $\mathbf{x} \in \{0, 1\}^n$; $\mathbf{1} \cdot f(\boldsymbol{\theta})$ is a dot product to sum the elements of $\mathbf{x}$ generated by f; '$\otimes$' denotes the row-wise multiplication[3] to apply the selection vector $\mathbf{x}$ to $\mathbf{P}$; and σ is the cross-entropy loss created from the ground-truth label ℓ and the predicted label by m. Our problem definition is greatly inspired by *attention* where neural networks focus on a meaningful subset of information (Vaswani et al., 2017). Although Eq. (7) is a minimization problem, we can still apply the adaptive gradient ascent by maximizing its negative value.

[3] Because of the PointNet design, this row-wise multiplication is identical to selecting/deselecting points.

	Original (resampling ratio = 1.0)	Random (0.05)	Octree (0.06)	WLOP (0.05)	Grid (0.06)	Ours (0.05)
PointNet Micro F-1	0.8842	0.6353	0.7208	0.7070	0.7414	**0.8268**
PointNet Macro F-1	0.8304	0.5398	0.6344	0.6150	0.6607	**0.7457**

Table 2: PointNet accuracy after resampling points with various methods



(a) Original (b) Ours (ratio=5%, logit=40.8) (c) Grid (ratio=5.7%, logit=37.3) (d) WLOP (ratio=5%, logit=31.4) (e) Original (f) Ours (ratio=5%, logit=49.1) (g) Grid (ratio=4.3%, logit=42.5) (h) Random (ratio=5%, logit=46.1)

Figure 3: PCR Visualization. The logit values are extracted from PointNet. We remove Octree due to its relatively poor quality.

Datasets. We use the Princeton ModelNet40 (Zhirong Wu et al., 2015) which contains 12,308 samples from various different object classes, e.g., sofa, desk, chair, etc. 2,468 samples were reserved for our resampling test and others were used to train PointNet.

Baselines and Hyperparameters. We test four baselines: random resampling (Random) (CloudCompare Open Source Project Team, 2019), Octree (CloudCompare Open Source Project Team, 2019), weighted locally optimal projection (WLOP) (The CGAL Project, 2019), and grid resampling (Grid) (The CGAL Project, 2019). As there are no commonly-accepted evaluation metrics for PCR, we compare our KP-based PCR with the baselines using the classification performance by PointNet.

We set $B = 0.05$, $\xi = 0.1$, $\gamma = 0.0001$, and $k = 3,000$. We initialize e_i with the LeCun normal initializer (Sutskever et al., 2013) for our method. In particular, we test with a low resampling ratio $B = 0.05$, which is highly challenging. We use PTE to generate x_i from e_i. For those baselines, we set the same resampling ratio of 0.05. However, Octree and Grid sometimes resample more than 0.05 and their average resampling ratios are 0.06 as shown in Table 2.

Experimental Results. We feed each resampling result to PointNet and check if PointNet correctly recognizes its original class. In Table 2, we report the accuracy of each resampling method. Our method shows the best accuracy, which proves that our method optimizes well even with the complicated neural network-based objective. We also show some resampling results in Fig. 3 — more figures can be found in our supplementary file. In the figure, note that our results have the highest logit values. In all those airplane, guitar, and lamp examples, our method successfully captures all the important characteristics.

4.2. Graph Convolutional Networks

Let $G = (V, E)$ be a graph. When we know the ground-truth class labels for a subset $V' \subset V$, the transductive inference of GCNs is to infer other unknown vertex labels from the known subset V'. Transductive inference is popular in machine learning, e.g., label propagation, belief propagation, and so forth (Bishop, 2006). It can be described as minimizing $\sigma(\boldsymbol{\ell}, g(\boldsymbol{\theta}_g, \mathbf{A}))$, where g is a GCN model; $\boldsymbol{\theta}_g$ is its model parameters; $\mathbf{A} \in \{0,1\}^{|V|\times|V|}$ is the adjacency matrix of G; $\boldsymbol{\ell}$ is a set of ground-truth labels which we already know for $V' \subset V$; and σ is an evaluation metric such as cross-entropy loss[4]. We solve the following problem to improve the transductive inference of GCNs:

$$\begin{aligned} \min_{\boldsymbol{\theta}_g, \boldsymbol{\theta}} \quad & \sigma(\boldsymbol{\ell}, g(\boldsymbol{\theta}_g, abs(\mathbf{A} - f(\boldsymbol{\theta})))) \\ \text{subject to} \quad & \frac{\mathbf{1} \cdot vec(f(\boldsymbol{\theta}))}{|V| \times |V|} \le B, \end{aligned} \tag{8}$$

where $f(\boldsymbol{\theta})$ outputs a *flipping matrix* $\mathbf{F} \in \{0,1\}^{|V|\times|V|}$. Each element of the flipping matrix is a binary variable to decide whether we flip (invert) $a_{v,u} \in \mathbf{A}$ or not. $abs(\mathbf{A} - f(\boldsymbol{\theta}))$ means the flip operation in the problem definition and our method can correctly deal with the absolute function. One can consider the proposed KP problem is to find the optimal adjacency modification, given the modification ratio budget B, that leads to the highest inference performance. The value assignment function o in Eq. (1) corresponds to $\sigma(\boldsymbol{\ell}, g(\cdot))$ in this problem definition.

The rationale behind our problem definition is that graphs often include weak and/or missing connections that may

[4]This cross-entropy loss is defined only for V'. For other testing vertices $V \setminus V'$, we can infer their labels as side products of minimizing the loss for V' because vertices are connected.

Model	Cora	Citeseer	Model	Cora	Citeseer	Model	Cora	Citeseer
GAT	0.830	0.725	GLN	0.812	0.709	Chebyshev	0.812	0.698
AGNN	0.831	0.717	GCN	0.815	0.703	LNet	0.795	0.662
LGCN	0.833	0.730	AdaNet	0.804	0.687	FastGCN	0.798	0.688
GIN	0.776	0.661	SGC	0.810	0.719	Ours (LGCN + Flip)	**0.840**	**0.732**

Table 3: GCN transductive inference accuracy

disturb the inference process (Liben-Nowell & Kleinberg, 2003). By correcting them, we show that we can achieve better performance. We train $\boldsymbol{\theta}$ and $\boldsymbol{\theta}_g$ together in this experiment, which shows the flexibility in our method.

Datasets. We test two standard benchmark graphs: Cora and Citeseer. These datasets were used in many works such as (Yang et al., 2016; Kipf & Welling, 2016; Gao et al., 2018), to name a few.

Baselines and Hyperparameters. We compare with several state-of-the-art transductive inference models such as FastGCN (Chen et al., 2018), GAT (Veličković et al., 2017), GLN (Wu et al., 2019), AGNN (Wu et al., 2019), LNet (Liao et al., 2019), AdaNet (Liao et al., 2019), SGC (Wu et al., 2019), GCN (Kipf & Welling, 2016), and LGCN (Gao et al., 2018). We also compare with other non-GCN-based models such as Chebyshev (Defferrard et al., 2016). To test our method, we choose LGCN as the base GCN model g in Eq. (8).

We use the default hyperparameters of LGCN. For B, we test with $B = \{0.005, 0.001, 0.0001\}$. We use $\xi = 0.1$ as the mini-batch size. We set the initialization of e_i to zero for all i, γ to 0.01, and k to 1000. We use the STE-based item selection network with the slope annealing step size $s = 50$ and the change rate $r = 1.004$ to generate x_i from e_i. $\boldsymbol{\theta}$ and $\boldsymbol{\theta}_g$ are trained alternately. We use Adam to train $\boldsymbol{\theta}_g$ and the proposed adaptive gradient method for $\boldsymbol{\theta}$.

Experimental Results. In Table 3, we summarize results. While both LGCN and GAT show reliable performance, our method based on LGCN with the adjacency matrix modification shows the best performance in the two benchmark graphs. Theoretically, the accuracy of our method is lower bounded by the original LGCN as our method is reduced to the original LGCN when there are no flippings.

4.3. Benchmark Linear KP Instances

We also test with linear KP instances, whose problem formulations follow the equation below.

$$\begin{aligned} \max_{x_i \in \{0,1\}, 1 \le i \le n} \quad & \sum_{i=1}^{n} v_i x_i \\ \text{subject to} \quad & \sum_{i=1}^{n} c_i x_i \le B. \end{aligned} \tag{9}$$

In this case, the objective is not based on a neural network. We include this experiment, although our main interest lies in neural network-based KPs, since the comparison with other mathematical solvers for linear KP instances can highlight the efficacy of our proposed method.

Datasets. We use the benchmark KP instance generator created by Pisinger (Pisinger, 2005). It can generate many realistic KP instances where item values/costs and budgets are decided by the generator. We only generate large-scale instances where $n = 1,000,000$, as smaller-scale instances can be exactly solved with many other existing techniques.

These KP instances can be classified into six types depending on the characteristics of v_i and c_i for each item i. One representative type is *spanner* where v_i and c_i are multiples of a quite small set of numbers that is called *spanner set*. In Table 4, we use the spanner set $span(2, 10)$ where the two parameters 2 and 10 were chosen by Pisinger. We strictly follow his benchmark design.

In each type, item values and costs are determined following a special distribution/function. We describe the six types we used for our experiments as follows. We choose $R \ge 10^4$ for our experiments wherever applicable:

1. Uncorrelated Span(p, m): A set of p items are generated with costs in the interval [1,R] and uncorrelated values. The items (v_j, c_j) in the spanner set are normalized by diving the values and the costs with $m+1$. The n items are then constructed, by repeatedly choosing a item (v_j,c_j) from the spanner set and a multiplier a, randomly generated in the interval [1,m]. The constructed item has value and cost ($a \cdot v_j$, $a \cdot c_j$). It is suggested that hard instances are generated for smaller spanner sets (Pisinger, 2005) with $p = 2$ and $m = 10$.

2. Weakly Correlated Span(p, m): A similar method described in 1 is used with $p = 2$ and $m = 10$ while v_j and c_j are weakly correlated.

3. Strongly Correlated Span(p, m): A similar method described in 1 is used with $p = 2$ and $m = 10$ while v_j and c_j are strongly correlated.

4. Strongly correlated: Costs c_j are distributed in [1, R] and $v_j = c_j + R/10$, for some $R > 1$.

Method	Uncorr. Span(2,10)		Weak. Corr. Span(2,10)		Str. Corr. Span(2,10)	
	Is optimal?	Time	Is optimal?	Time	Is optimal?	Time
Combo	O	329 sec	O	3018 sec	O	19.13 sec
Gurobi	O	**2.61** sec	O	2.78 sec	O	**2.70** sec
Ours	O	4.46 sec	O	**1.70** sec	O	3.17 sec

Method	Strongly correlated		Inverse strongly correlated		Uncorrelated	
	Is optimal?	Time	Is optimal?	Time	Is optimal?	Time
Combo	O	$<$ **1** sec	O	$<$ **1** sec	N/A	Timeout
Gurobi	O	42.72sec	O	19.39sec	O	21.71 sec
Ours	O	60.16 sec	O	4.72 sec	O	$<$ **1** sec

Table 4: Results on benchmark KP instances with 6 types. Each type defines how to generate item values and costs. Due to the very large scale of objective values, we only show whether solution is optimal or not. 'Timeout' means it cannot be solved in an hour in our experiments. OR-Tools and LS are timed out in all cases and are not listed in this table.

5. Inverse strongly correlated: Values v_j are distributed in [1, R] and $c_j = v_j + R/10$ for some $R > 1$.

6. Uncorrelated: v_j and c_j are chosen randomly in [1,R] for some $R > 1$. In these instances, no correlation exists between the value and the cost of an item.

Baselines and Hyperparameters. We test four baselines: Combo (Martello et al., 1999), Gurobi (Gurobi Optimization, LLC, 2019), LocalSolver (LS) (LocalSolver, LLC, 2019), and OR-Tools by Google AI (Google, LLC, 2019).

All these baselines returned optimal solutions. They resort to many techniques such as dynamic programming, branch and bound, integer linear programming with LP-relaxation, Horowitz-Sahni decomposition, and so on. However, only few of them are scalable up to $n = 1,000,000$. Existing deep learning methods (Bello et al., 2016; Gu & Hao, 2018) are not scalable because their LSTM-based architectures cannot process a list of one million items. We give an hour for each solver to solve each instance. For our method, we use $\xi = 0.1$ as the mini-batch size. We set the initialization of e_i to zero for all i, γ to 0.1, and k to 100. We use STE with the slope annealing step size $s = 50$ and the change rate $r = 1.01$ to generate x_i from e_i. For baseline solvers, we rely on their automatic configurations.

Experimental Results. In Table 4, we summarize all the results. In general, Combo and Gurobi show better performance than other baselines except for the type 6, where Combo could not solve in an hour. OR-Tools and LS were all timed out. We think that Gurobi shows great performance considering its wide applicability — Gurobi is a versatile tool to solve (mixed integer) linear programming, quadratic programming, etc. Our proposed method solves all instances in less than about a minute. Moreover, our method and Gurobi are the only ones which solve all the instances. Due to the simplicity of linear KPs, our method could find optimal solutions in all types.

5. Conclusions & Future Work

To solve 0-1 KPs with neural network-based objective functions, we presented a max-min game and then developed a more efficient but equivalent way where the inner minimization can be completely removed. After that, our presented adaptive gradient ascent method is able to find feasible solutions with good objective values for large-scale KPs with up to a million of items. Our adaptive gradient ascent is designed to guarantee a decrease in the total cost if any cost overrun. We also showed that our method can be utilized for a couple of applications, including but not limited to PCR and the transductive inference of GCNs. In our application to PCR, we could achieve high accuracy, which was also corroborated by the GCN experimental results. We also showed that our presented method can be used to solve linear KPs. In comparison with classical solvers, our method shows comparable performance. In addition, we see that many deep learning-related problems can be modeled as KPs (although the effectiveness of modeling them as KPs is under-explored). Therefore, our work can inspire other related researchers.

One limitation in our work is that we do not support advanced optimization techniques yet. In particular, trusted region methods (Dauphin et al., 2014) are effective optimization techniques in increasing the quality of solutions and avoiding saddle points. This paper is our first work in this line of research and we plan to keep extending Thm. (3) in the future. Our overall framework should still work after redefining β in Thm. (3) for other optimization techniques.

Acknowledgements

This work was supported by the Institute of Information & Communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (No. 2020-0-01361, Artificial Intelligence Graduate School Program (Yonsei University)).

References

Arjovsky, M. and Bottou, L. Towards Principled Methods for Training Generative Adversarial Networks. In *ICLR*, 2017.

Bello, I., Pham, H., Le, Q. V., Norouzi, M., and Bengio, S. Neural combinatorial optimization with reinforcement learning. *CoRR*, abs/1611.09940, 2016.

Bengio, Y., Léonard, N., and Courville, A. C. Estimating or propagating gradients through stochastic neurons for conditional computation. *CoRR*, abs/1308.3432, 2013.

Bishop, C. M. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag, 2006.

Boyd, S. and Vandenberghe, L. *Convex Optimization*. Cambridge University Press, USA, 2004.

Bretthauer, K. M. and Shetty, B. The nonlinear knapsack problem - algorithms and applications. *European Journal of Operational Research*, 138(3):459–472, 2002.

Chen, J., Ma, T., and Xiao, C. Fastgcn: Fast learning with graph convolutional networks via importance sampling. *CoRR*, abs/1801.10247, 2018.

Chung, J., Ahn, S., and Bengio, Y. Hierarchical multiscale recurrent neural networks. *CoRR*, abs/1609.01704, 2016.

CloudCompare Open Source Project Team. Cloudcompare [gpl software], 2019. URL `http://www.cloudcompare.org/`.

Cohen, R. and Katzir, L. The generalized maximum coverage problem. *Inf. Process. Lett.*, 108(1):15–22, September 2008.

Dauphin, Y. N., Pascanu, R., Gulcehre, C., Cho, K., Ganguli, S., and Bengio, Y. Identifying and attacking the saddle point problem in high-dimensional non-convex optimization. In *NeurIPS*. 2014.

Defferrard, M., Bresson, X., and Vandergheynst, P. Convolutional neural networks on graphs with fast localized spectral filtering. *CoRR*, abs/1606.09375, 2016.

Fomeni, F. D. and Letchford, A. N. A dynamic programming heuristic for the quadratic knapsack problem. *INFORMS Journal on Computing*, 26(1):173–182, 2014.

Gao, H., Wang, Z., and Ji, S. Large-scale learnable graph convolutional networks. *CoRR*, abs/1808.03965, 2018.

Garey, M. R. and Johnson, D. S. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., 1979.

Goodfellow, I. J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. Generative Adversarial Networks. In *NeurIPS*, 2014.

Google, LLC. Google or-tools, 2019. URL `https://developers.google.com/optimization/`.

Gu, S. and Hao, T. A pointer network based deep learning algorithm for 0-1 Knapsack Problem. In *ICACI*, 2018.

Gu, S. and Yang, Y. A pointer network based deep learning algorithm for the max-cut problem. In *ICONIP*, 2018.

Gurobi Optimization, LLC. Gurobi optimizer reference manual, 2019. URL `http://www.gurobi.com`.

Hertrich, C. and Skutella, M. Provably good solutions to the knapsack problem via neural networks of bounded size. In *AAAI*, 2021.

Kellerer, H., Pferschy, U., and Pisinger, D. *Knapsack Problems*. Springer, Berlin, Germany, 2004.

Khalil, E., Dai, H., Zhang, Y., Dilkina, B., and Song, L. Learning combinatorial optimization algorithms over graphs. In *NeurIPS*. 2017.

Kipf, T. N. and Welling, M. Semi-supervised classification with graph convolutional networks. *CoRR*, abs/1609.02907, 2016.

Li, Z., Chen, Q., and Koltun, V. Combinatorial optimization with graph convolutional networks and guided tree search. In *NeurIPS*. 2018.

Liao, R., Zhao, Z., Urtasun, R., and Zemel, R. S. Lanczosnet: Multi-scale deep graph convolutional networks. *CoRR*, abs/1901.01484, 2019.

Liben-Nowell, D. and Kleinberg, J. The Link Prediction Problem for Social Networks. In *CIKM*, 2003.

LocalSolver, LLC. Localsolver, 2019. URL `https://www.localsolver.com/`.

Martello, S. and Toth, P. *Knapsack Problems: Algorithms and Computer Implementations*. John Wiley & Sons, Inc., 1990.

Martello, S., Pisinger, D., and Toth, P. Dynamic programming and strong bounds for the 0-1 knapsack problem. *Management Science*, 45(3):414–424, 1999.

Mena, G., Belanger, D., Linderman, S., and Snoek, J. Learning latent permutations with gumbel-sinkhorn networks. In *ICLR*, 2018.

Nomer, H. A. A., Alnowibet, K. A., Elsayed, A., and Mohamed, A. W. Neural knapsack: A neural network based solver for the knapsack problem. *IEEE Access*, 8:224200–224210, 2020.

Pisinger, D. Where are the hard knapsack problems? *Comput. Oper. Res.*, 32(9):2271–2284, September 2005.

Qi, C. R., Su, H., Mo, K., and Guibas, L. J. Pointnet: Deep learning on point sets for 3d classification and segmentation. *CoRR*, abs/1612.00593, 2016.

Ruszczynski, A. *Nonlinear Optimization*. Princeton University Press, 2006.

Sato, R., Yamada, M., and Kashima, H. Approximation ratios of graph neural networks for combinatorial problems. In *NeurIPS*. 2019.

Skiena, S. S. Who is interested in algorithms and why?: Lessons from the stony brook algorithms repository. *SIGACT News*, 30(3):65–74, September 1999.

Sutskever, I., Martens, J., Dahl, G., and Hinton, G. On the Importance of Initialization and Momentum in Deep Learning. In *ICML*, 2013.

The CGAL Project. *CGAL User and Reference Manual*. CGAL Editorial Board, 4.14 edition, 2019. URL `https://doc.cgal.org/4.14/Manual/packages.html`.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. Attention is all you need. *CoRR*, abs/1706.03762, 2017.

Veličković, P., Cucurull, G., Casanova, A., Romero, A., Liò, P., and Bengio, Y. Graph Attention Networks. art. arXiv:1710.10903, Oct 2017.

Vinyals, O., Fortunato, M., and Jaitly, N. Pointer networks. In *NeurIPS*. 2015.

Wang, Y., Yin, W., and Zeng, J. Global convergence of admm in nonconvex nonsmooth optimization. *Journal of Scientific Computing*, 78(1):29–63, Jan 2019.

Wu, F., Zhang, T., Jr., A. H. S., Fifty, C., Yu, T., and Weinberger, K. Q. Simplifying graph convolutional networks. *CoRR*, abs/1902.07153, 2019.

Wu, Z., Karimi, H. R., and Dang, C. A deterministic annealing neural network algorithm for the minimum concave cost transportation problem. *IEEE Transactions on Neural Networks and Learning Systems*, 2019.

Yang, Z., Cohen, W. W., and Salakhutdinov, R. Revisiting Semi-supervised Learning with Graph Embeddings. In *ICML*, 2016.

Yolcu, E. and Poczos, B. Learning local search heuristics for boolean satisfiability. In *NeurIPS*. 2019.

Yu, T., Yan, J., Wang, Y., Liu, W., and Li, B. Generalizing graph matching beyond quadratic assignment model. In *NeurIPS*, 2018.

Zhirong Wu, Song, S., Khosla, A., Fisher Yu, Linguang Zhang, Xiaoou Tang, and Xiao, J. 3d shapenets: A deep representation for volumetric shapes. In *CVPR*, 2015.

Zoph, B. and Le, Q. V. Neural architecture search with reinforcement learning. *CoRR*, abs/1611.01578, 2016.