# WeaveNet: A Differentiable Solver for Non-linear Assignment Problems

**Anonymous authors**
Paper under double-blind review

## Abstract

Assignment, a task to match a limited number of elements, is a fundamental problem in informatics. Traditionally, non-linear assignment is discussed as a combinatorial optimization problem with its calculation complexity. On the other hand, it is often a sub-problem of image processing tasks, such as 3D point cloud matching. This paper proposes WeaveNet, a differentiable solver for diverse non-linear assignment problems. Traditional graph convolutional networks (GCNs) suffer from an over-smoothing problem when characterizing nodes with their relationship. WeaveNet overcomes this problem by forwarding edge-wise features at each layer rather than aggregated node features. To deal with the exponentially large input space of combinatorial optimization problems, we designed WeaveNet to be highly parameter efficient while characterizing edges through stacked *set-encoder* with *cross-concatenation* operations. Experimental results show that WeaveNet approximates two *strongly NP-hard* variants of stable matching in a comparative performance with the gold standard hand-crafted algorithms under the limited size of problem instances. We have also confirmed that it can boost 3D point cloud matching performance significantly.

## 1 Introduction

Assignment problems are defined on a bipartite graph $\mathcal{K}_{N_1,N_2}(\mathbb{V}_1, \mathbb{V}_2, \mathbb{A})$, where $N_1 = |\mathbb{V}_1|$, $N_2 = |\mathbb{V}_2|$ $(N_1 \geq N_2)$, and $\mathbb{A} = \{\{v, w\} | v \in \mathbb{V}_1, w \in \mathbb{V}_2\}$. On the graph, non-linear assignment is a problem to find $\mathbb{M} \subset \mathbb{A}$ that satisfies non-linear conditions and objective functions. A typical traditional example is stable matching (Gale & Shapley, 1962), which was firstly applied to a hospital-student assignment problem and nowadays applied to wider social problems such as kidney exchange (Roth et al., 2007). On the other hand, it is known that adding a certain kind of objective functions make the stable matching problem *strongly NP-hard* (Kato, 1993; McDermid & Irving, 2014).

Not limited to the family of stable matching problem, non-linear assignment problem is essentially involved in many practical applications (Zeng et al., 2021; Fu et al., 2021; He et al., 2021; Gao et al., 2021; El Banani et al., 2021; Wiles et al., 2021). A typical example is 3D point cloud matching (Zeng et al., 2021; Fu et al., 2021), where $\mathbb{V}_1$ and $\mathbb{V}_2$ are point clouds observed in independent frames. Each element in $\mathbb{V}_{1/2}$ is expressed by a latent feature extracted by a non-linear encoder. Then, in the assignment part, weights on $\mathbb{A}$ are calculated as an adjacency matrix. The matrix is binarized to $\mathbb{M}$ through some kinds of linear assignment operations. As a result, the latent space is optimized to embed each feature into a linearly assignable position, which is hard since the encoder does not refer $\mathbb{V}_2$ ($\mathbb{V}_1$) when embedding elements in $\mathbb{V}_1$ ($\mathbb{V}_2$).

This paper aims to develop a fully differentiable approximate solver for general non-linear assignment problems. As such solver, we propose a novel network architecture, *WeaveNet*. The contribution of this paper is five-fold.

1. We propose a novel non-linear assignment solver, WeaveNet, which achieves high generalization ability than the previous differentiable solvers.
2. Weavenet is the first fully-symmetric solver, which mathematically ensures fair treatment between $\mathbb{V}_1$ and $\mathbb{V}_2$. We also provide its asymmetric variant for highly biased situations.

3. We provide a scalable testing protocol with *strongly NP-hard* variants of stable matching. We also propose an evaluation metric, *pseudo fairness scores*, that enables direct comparison with hand-crafted algorithms.
4. WeaveNet is the first differentiable solver that achieved a comparative performance with the SoTA hand-crafted algorithm with no error correction process although the problem size is limited to $N_1 \leq 30$.
5. We proposed to combine a differentiable solver to CorrNet3D (Zeng et al., 2021) and demonstrated the impact of WeaveNet on 3D point cloud matching.

## 2 Related work

### 2.1 Combinatorial optimization with loss minimization

There have been attempts to search for a good solution by loss minimization for each problem instance. Gold & Rangarajan (1996) proposed a method named SoftAssign, which relaxes loss for the quadratic assignment problem. Later, its step-wise landscape of loss function was pointed by Lozano & Escolano (2013) with a kernel-based approach that smoothens the landscape. Another group of attempts is to search solutions with a pre-trained deep learning model and an error correction process (Wang et al., 2019; Selsam et al., 2019). The error correction requires complete input information. However, such information is often unobservable in real applications (e.g., the latent feature inputs or online assignment problems).

Our goal is to develop a deep learning model that reaches a good solution without error correction, and there are two such previous studies. Li (2019) tried to approximate the stable matching problem with multi-layer perceptrons (MLPs) in a fully unsupervised manner. Gibbons et al. (2019) tried to approximate the weapon-target assignment (WTA) problem with their original architecture, deep bipartite assignment (DBA). Both models have insufficient generalization ability and under-perform to hand-crafted algorithms even when $N \leq 20$.

### 2.2 Non-linear assignment problems in machine learning applications

Since visual observation contains multiple elements of interest in each frame, many computer vision applications involve bipartite matching as a sub-problem. A representative task is 3D point cloud matching (Zeng et al., 2021; Fu et al., 2021), but also structure-from-motion (El Banani et al., 2021), co-attention calculation (Wiles et al., 2021), graph matching (Gao et al., 2021), and multiple object tracking (He et al., 2021) are all potential applications of non-linear assignment solver.

Due to the lack of solvers that work with latent feature inputs, any of the above recent methods once summarize the latent features into an affinity matrix $\boldsymbol{M}$, which is binarized into a set of edges $\mathbb{M}$; however, $\mathbb{M}$ cannot be a good one-to-one matching when we independently select the $\arg\max$ element for each row or column. Hence, we need to make $\boldsymbol{M}$ be doubly stochastic. Sinkhorn operation (Mena et al., 2018), de-smoothing (Zeng et al., 2021), and the Hungarian algorithm (Kuhn, 1955) are the representative tools for maintaining the doubly stochastic property. Here, the elements in $\mathbb{V}_1$ and $\mathbb{V}_2$ do not communicate with each other before the affinity matrix calculation, and there are no trainable parameters after that. Therefore, such models request their feature extractor to project elements in linearly-assignable locations without any communications between $\mathbb{V}_1$ and $\mathbb{V}_2$, which is almost impossible. Our study aims to develop a differentiable solver that finds a better assignment through communications instead of calculations with affinity matrix.

## 3 WeaveNet

### 3.1 Differentiable non-linear assignment solvers

We define a differentiable non-linear assignment solver as a trainable function $f : \mathsf{P}, \mathsf{Q} \rightarrow \boldsymbol{M}$, where inputs $\mathsf{P}$ and $\mathsf{Q}$ are tensors of size $N_1 \times N_2 \times D_{input}$ and $N_2 \times N_1 \times D_{input}$. The output $\boldsymbol{M}$ is a $N_1 \times N_2$ matrix. $\mathsf{P}$'s $vw$-th element $\mathsf{P}_{v,w}$ represents the $D_{input}$-dimensional attribute of the directed edge from $v \in \mathbb{V}_1$ to $w \in \mathbb{V}_2$, and $\mathsf{Q}_{w,v}$ from $w$ to $v$. We can obtain $\mathbb{M}$, the discrete assignment solution, by binarizing $\boldsymbol{M}$.

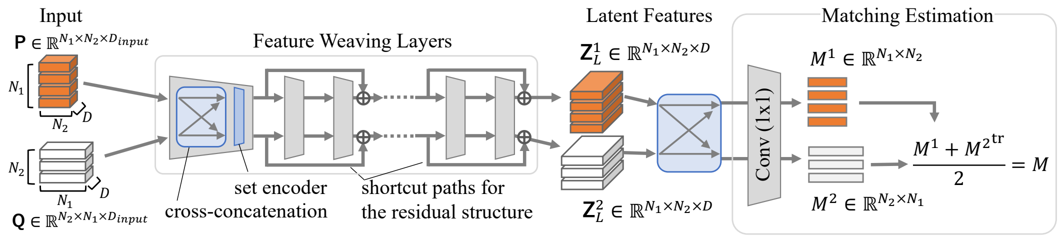
Input
$\mathbf{P} \in \mathbb{R}^{N_1 \times N_2 \times D_{input}}$
$\mathbf{Q} \in \mathbb{R}^{N_2 \times N_1 \times D_{input}}$
Feature Weaving Layers
set encoder
cross-concatenation
shortcut paths for the residual structure
Latent Features
$\mathbf{Z}_L^1 \in \mathbb{R}^{N_1 \times N_2 \times D}$
$\mathbf{Z}_L^2 \in \mathbb{R}^{N_1 \times N_2 \times D}$
Matching Estimation
Conv (1x1)
$M^1 \in \mathbb{R}^{N_1 \times N_2}$
$M^2 \in \mathbb{R}^{N_2 \times N_1}$
$\frac{M^1 + M^{2\,\mathrm{tr}}}{2} = M$

Figure 1: Two stream architecture of WeaveNet. It embeds inputs $\mathbf{P}$ and $\mathbf{Q}$ into two bands of latent features $\mathbf{Z}^1$ and $\mathbf{Z}^2$ through $L$ FW layers with the residual structure. The two streams communicate through the cross-concatenation in every FW layer. $\mathbf{Z}^1$ and $\mathbf{Z}^2$ are further processed through a convolution layer with $1 \times 1$ kernel size, which yields the output $M$.

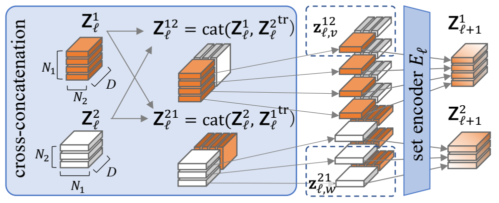
cross-concatenation
$\mathbf{Z}_\ell^1$
$\mathbf{Z}_\ell^2$
$\mathbf{Z}_\ell^{12} = \mathrm{cat}(\mathbf{Z}_\ell^1, \mathbf{Z}_\ell^{2\,\mathrm{tr}})$
$\mathbf{Z}_\ell^{21} = \mathrm{cat}(\mathbf{Z}_\ell^2, \mathbf{Z}_\ell^{1\,\mathrm{tr}})$
$\mathbf{z}_{\ell,v}^{12}$
$\mathbf{z}_{\ell,w}^{21}$
set encoder $E_\ell$
$\mathbf{Z}_{\ell+1}^1$
$\mathbf{Z}_{\ell+1}^2$

Figure 2: Architecture of feature weaving layer. It orthogonally concatenates the weft-wise and warpwize components ($\mathbf{Z}_\ell^1$ and $\mathbf{Z}_\ell^2$) in a symmetric way (cross-concatenation). Then, the concatenated tensors $\mathbf{Z}_\ell^{12}$ ($\mathbf{Z}_\ell^{21}$) is sliced into $\mathbf{z}_{\ell,v}^{12}$ ($\mathbf{z}_{\ell,w}^{21}$), which represents a set of directed edges from $v \in \mathbb{V}_1$ ($w \in \mathbb{V}_2$) to $\mathbb{V}_2$ ($\mathbb{V}_1$), and woven by $E_\ell$ one-by-one.

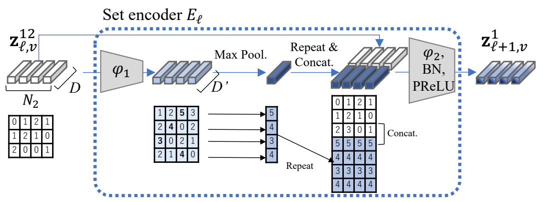

Figure 3: Architecutre of set encoder $E_\ell$. It first embed $\mathbf{z}_{\ell,v}^{12}$ (white) into $D'$ dimensional features (pale blue) by $\varphi_1$, then max-pooled to obtain the group characteristics of the corresponding (directed) edges (blue). It is combined to $\mathbf{z}_{\ell,v}^{12}$ and further processed by $\varphi_2$, batch normalization, and an activation to yield the output $\mathbf{z}_{\ell+1,v}^1$.

To model this calculation, previous model, DBA (Gibbons et al., 2019), firstly concatenates $P_{v,w}$ and $Q_{w,v}$ for each $\{v, w\}(\in \mathbb{A})$ and obtain an input tensor with the size of $N_1 \times N_2 \times 2D_{input}$, then encodes its row and column alternately through multiple layers. DBA realizes two important properties, size-independence and permutantion-equivalence for $\mathbb{V}_1$ and $\mathbb{V}_2$ by appropriately design its layer-level encoders. On the other hand, a swap of $\mathbf{P}$ and $\mathbf{Q}$ yields different solutions because each encoder is trained with each direction. This is not desirable because many problems are symmetric. Even with asymmetric problems, the structure is sub-optimal since the encoder shares the common functionality, selecting a partner of an element.

Furthermore, DBA's encoder is not efficient for assignment problems. They proposed two options for the encoder: self-attention and max-pool-concat. Self-attention works under the assumption that we can solve the problem without seeing entire elements, which is not the case for many complex assignment problems. The max-pool-concat sees entire elements but it unnecessarily restricts the function shape (see the next subsection).

## 3.2 The architecture of WeaveNet

WeaveNet overcomes above two problems by *cross-concatenation* and *set-encoder*. Fig. 1 illustrates the entire model. It consists of $L$ feature weaving (FW) layers, which have two streams that models the process for $\mathbb{V}_1$ to select $\mathbb{V}_2$ ($\mathbf{P} \rightarrow M^1$) and vice versa ($\mathbf{Q} \rightarrow M^2$).

Figure 2 illustrates the feature weaving layer and the operation of cross-concatenation. Let $\mathbf{Z}_\ell^1$ and $\mathbf{Z}_\ell^2$ be inputs for the first and second stream at $\ell$-th layer ($0 \leq \ell < L$), where $\mathbf{Z}_0^1 = \mathbf{P}$ and $\mathbf{Z}_0^2 = \mathbf{Q}$. Cross-concatenation is an operation to yields $\mathbf{Z}_\ell^{12} = cat(\mathbf{Z}_\ell^1, \mathbf{Z}_\ell^{2\,\mathrm{tr}})$ and $\mathbf{Z}_\ell^{21} = cat(\mathbf{Z}_\ell^2, \mathbf{Z}_\ell^{1\,\mathrm{tr}})$, where $\mathrm{tr}$ is an operation to transpose the first and second axes, and $cat$ is the concatenation operation on the third axis. Here, $\mathbf{Z}_\ell^{12}$ and $\mathbf{Z}_\ell^{21}$ are organized by $2D$ dimensional features. Swapping the first and last $D$ dimensions of $\mathbf{Z}_\ell^{12}$'s $vw$-th feature corresponds to its opposite, $\mathbf{Z}_\ell^{21}$'s $wv$-th feature. Since the first and last half components are both outputs of the same encoder $E_{\ell-1}$, features in $\mathbf{Z}_\ell^{12}$ and $\mathbf{Z}_\ell^{21}$ are always in the same latent space. Therefore, the single encoder $E_\ell$ can process them identically.

More formally, let $\mathbf{z}_{\ell,v}^{12} \in \mathbb{R}^{N_2 \times D}$ be a row of $\mathbf{Z}_\ell^{12}$, which is a set of $N_2$ feature vectors corresponding to edges from $v \in \mathbb{V}_1$ to the $N_2$ elements in $\mathbb{V}_2$, and $\mathbf{z}_{\ell,w}^{21} \in \mathbb{R}^{N_1 \times D}$ is a row of $\mathbf{Z}_\ell^{21}$ ($w$ to $\mathbb{V}_1$). Then, the layer-level encoder is applied for each row as $\mathbf{z}_{\ell+1,v}^{1} = E_\ell(\mathbf{z}_{\ell,v}^{12})$ and $\mathbf{z}_{\ell+1,w}^{2} = E_\ell(\mathbf{z}_{\ell,w}^{21})$. From this formulation, we can see that a $L$-layered WeaveNet has the same encoding frequency with a $2L$-layered DBA and thus is twice parameter-efficient.

On the layer-level encoder, the DBA's max-pool-concat outputs $cat(\varphi(\mathbf{z}), \max_{\mathbf{z}' \in \mathbf{z}_v} \varphi(\mathbf{z}'))$ for each $\mathbf{z} \in \mathbf{z}_v$, where $\varphi$ is a trainable function for the $D$-dimensional feature $\mathbf{z}$. This calculation unnecessarily requests $\varphi(\mathbf{z})$ and its max-pooled vector to be informative simultaneously. We decompose them with two trainable functions $\varphi_1$ and $\varphi_2$ inspired by DeepSet (Zaheer et al., 2017) and PointNet (Qi et al., 2017). Figure 3 shows our layer-level encoder, *set encoder*, which we can formulate as

$$E(\mathbf{z}_v) = (\varphi_2(cat(\mathbf{z}, \max_{\mathbf{z}' \in \mathbf{z}_v} \varphi_1(\mathbf{z}')))|\mathbf{z} \in \mathbf{z}_v), \tag{1}$$

where $\varphi_1$ is a linear operation and $\varphi_2$ consists of another linear operation followed by batch-normalization and PReLU activation.

The output of the stacked FW layers $\mathbf{Z}_L^1$ and $\mathbf{Z}_L^2$ are further converted into $\boldsymbol{M}^1 \in \mathbb{R}^{N_1 \times N_2 \times 1}$ and $\boldsymbol{M}^2 \in \mathbb{R}^{N_2 \times N_1 \times 1}$. Then, they are averaged into a single matrix $\boldsymbol{M}$, which is the final output of the WeaveNet.

### 3.3 Asymmetric variant with split batch normalization

Owing to the above cross-concatenation strategy, WeaveNet is fully symmetric for inputs $\mathbf{P}$ and $\mathbf{Q}$ in the sense it satisfies the equation $f(\mathbf{P}, \mathbf{Q}) = f(\mathbf{Q}, \mathbf{P})^{\mathrm{tr}}$. This condition ensures that the model architecture cannot distinguish the two sides $\mathbb{V}_1$ and $\mathbb{V}_2$ innately. This property is beneficial when we need to treat $\mathbb{V}_1$ and $\mathbb{V}_2$ completely fair. However, when the distributions of $\mathbf{P}$ and $\mathbf{Q}$ are differently biased, or biased output is desirable (e.g., an asymmetric objective function purposing affirmative action), this symmetric property may be undesirable. To deal with such biased situations without losing the parameter-efficiency, we further propose to apply batch normalization independently for each stream (*split batch normalization*), and adding a side-identifiable code (i.e., concatenate $\mathbf{1}^{N_1 \times N_2}$ to $\mathbf{P}$ and $\mathbf{0}^{N_2 \times N_1}$ to $\mathbf{Q}$). We call this model the *asymmetric variant*.

## 4 Experiments

To evaluate differentiable solvers, we have conducted two experiments: a systematic test with the stable matching problem and a more practical test with 3D point cloud matching.

### 4.1 Test protocol with stable matching

As most vision-based applications suffer from the limitation in dataset size, it is hard to analyze the solver's ability from multiple points of view with such real applications. To evaluate solvers systematically, we propose a test protocol with stable matching.

Stable matching, also known as the stable marriage problem, is one of the most popular non-linear assignment problems that models the two-sided market. Let $\boldsymbol{P} \in \mathbb{Z}_+^{N_1 \times N_2}$ be a matrix representing a set of preference list. Namely, $P_{v,w}$ is the preference rank of $v$-th agent[1] in $\mathbb{V}_1$ for $w$-th agent in $\mathbb{V}_2$ ($P_{v,w} = 1$ represents the highest preference and $P_{v,w} = N_2$ the lowest). $Q_{w,v}$ is the preference rank in the opposite direction. In a standard definition, $\boldsymbol{P}_{v,:}$ ($v$-th row in $\boldsymbol{P}$) includes no ties ($P_{v,w_1} = P_{v,w_2}$ only if $w_1 = w_2$) and $N_1 = N_2$ is assumed without loss of generality[2]. We say that an unmatched pair $\{v, w\} \in \mathbb{A} \backslash \mathbb{M}$ blocks $\mathbb{M}$ when $v$ prefers $w$ more than $v$'s partner in $\mathbb{M}$ and $w$ prefers $v$ more than $w$'s partner in $\mathbb{M}$. Matching $\mathbb{M}$ is stable when there is no blocking pair in $\mathbb{A} \backslash \mathbb{M}$.

[1]An element in $\mathbb{V}_1$ and $\mathbb{V}_2$ is called *agent* in the context of stable matching.

[2]We can add $(N_1 - N_2)$ dummy agents to $\mathbb{V}_2$ without disturbing original solutions.

Note that the Gale-Shapley (GS) algorithm can find it in $O(N^2)$ (Gale & Shapley, 1962). However, the GS algorithm has a biased nature, where one side is prioritized and the other side only gets the least preferable result among all the possibilities of stable matching. To compensate for the unfairness, we can introduce diverse objectives to maintain a stable matching fair. Among them, the following two fairness objectives make the stable matching problem *strongly NP-hard*. The first one is **Sex equality cost** ($SEq$) (Gusfield & Irving, 1989). It focuses on the unfairness brought by the gap between the two sides' satisfaction and defined by

$$SEq(\mathbb{M};\boldsymbol{P},\boldsymbol{Q}) = |\rho(\mathbb{M};\boldsymbol{P}) - \rho(\mathbb{M};\boldsymbol{Q})|, \quad \rho(\mathbb{M};\boldsymbol{P}) = \sum_{\{v,w\}\in\mathbb{M}} P_{v,w}, \quad \rho(\mathbb{M};\boldsymbol{Q}) = \sum_{\{v,w\}\in\mathbb{M}} Q_{w,v}. \tag{2}$$

The other is **Balance cost** ($Bal$) (Feder, 1995; Gupta et al., 2021), which is a compromise between side-equality and overall satisfaction. It is defined by

$$Bal(\mathbb{M};\boldsymbol{P},\boldsymbol{Q}) = \max(\rho(\mathbb{M};\boldsymbol{P}), \rho(\mathbb{M};\boldsymbol{Q})) \tag{3}$$

We optimize differentiable solvers based on these objective with the no-blocking-pair constraint, based on the continuous relaxation originally proposed in (Li, 2019), which requires no supervision from inaccessible ground truth. In that optimization process, we converts $\boldsymbol{P}$ and $\boldsymbol{Q}$ into $\mathsf{P} \in \mathbb{R}^{N\times N\times 1}$ and $\mathsf{Q} \in \mathbb{R}^{N\times N\times 1}$, respectively, while binarizing $\boldsymbol{M}$ into $\mathbb{M}$ by $\mathrm{argmax}$ operations. See A.1 in the appendix for more details, including the continuously relaxed loss functions.

### 4.1.1 Dataset

To evaluate the robustness of the model in diverse input distributions, we generated the following five datasets by the protocol developed in Tziavelis et al. (2019).

**Uniform (U)** Each agent's preference towards any matching candidate is totally random, defined by a uniform distribution $\mathcal{U}(0,1)$ (larger value means prior in the preference list).

**Discrete (D)** Each agent has a preference of $\mathcal{U}(0.5,1)$ towards a certain group of $\lfloor 0.4N \rfloor$ popular candidates, while $\mathcal{U}(0,0.5)$ towards the rest.

**Gauss (G)** Each agent's preference towards $i$-th candidate is defined by a Gaussian distribution $\mathcal{N}(i/N, 0.4)$.

**Biased (UD)** $\boldsymbol{P}$ and $\boldsymbol{Q}$ are generated by the distribution for U and D, respectively.

**LibimSeTi (Lib)** Simulate real rating activity on the online dating service LibimSeTi (Brožovský & Petříček, 2007) based on the 2D distribution of frequency of each rating pair ($P_{vw},\ Q_{wv}$).

### 4.1.2 Advantages of the proposed test protocol

The test protocol with stable matching provides us the following advantages.

**Scalability** We can easily generate a large-size dataset while changing $N$. Although this paper test the model with small $N$, we can continuously enhance the technology with this test protocol.

**Generalizability check** On stable matching, the total number of problem instances is $N!^{2N-2}$ (e.g., $4.3\times10^{16}$ at $N=5$, $2.7\times10^{44}$ at $N=7$, and $3.73\times10^{1880}$ at $N=30$), while a small change in an instance results in totally different output. Thus, an overfit model cannot work well.

**Gold standards** The code for strong baselines of hand-crafted algorithms are implemented by Tziavelis et al. (2019)[3]. We can use them as a gold standard instead of inaccessible optimal solutions of the strongly NP-hard settings.

**Input distribution shift** We can evaluate the model with a shifted test set. For example, changing $N$ or the type of distribution at training and test is a good simulation of real situations.

### 4.1.3 Training and evaluation protocols

We trained learning-based models 50k total iterations at $N \le 10$, 200k at $N \le 30$, and 300k at $N = 100$, with a batch size of 8. Thus, 400k, 1.6M, and 2.4M randomly generated samples are involved in training, which dominates only a tiny percentage of the input space when $N \ge 5$, but

[3]They only provides the code of algorithms. Thus, we newly publish the code for data generation.

can involve all possible problem instances at $N = 3$. We used Adam optimizer with an initial learning rate of 0.0001 unless otherwise stated.

We have sampled 1,000 validation and 1,000 test samples for each distribution and each $N$. Note that the training set can overlap with them, but its chance rate is negligible when $N \geq 5$. We will publish these validation/test sets to ensure reproducibility.

A solution that violates the no-blocking-pair constraint can achieve a fairness cost ($SEq$ or $Bal$) even lower than the ideal value. Hence, a direct comparison in $SEq$ ($Bal$) is unfair. To enable a fair comparison with hand-crafted algorithms, we propose *pseudo fairness costs*, $pSEq$ and $pBal$, which is calculated by replacing a fairness cost of violation cases with the highest one[4], which maximally penalizes the violation.

We also propose *average error per agent*, which is defined as $(pSEq - SEq_{ideal})/N$ where $SEq_{ideal}$ is the optimal solution. This error indicates how much additional cost (against the optimal solution) each agent must pay. Note that we can calculate this score only when the optimal solution is accessible by a brute-force search ($N \leq 10$).

### 4.1.4 COMPARISON WITH DIFFERENTIABLE SOLVERS ($N = 3,\ 5,\ 7,\ 9$)

In this experiment, we show results obtained by following baselines and WeaveNet variants. **MLP** is the model proposed in Li (2019). **GIN** is the state-of-the-art GCN model proposed in Xu et al. (2019), which will, however, suffer from over-smoothing problem (Li et al., 2018; Oono & Suzuki, 2020) when applied to distinguish agent's characteristics. **DBA-P/DBA-A** are DBA (Gibbons et al., 2019) with max-pool-concat and self-attention, respectively. They are re-implemented by us due to the absence of the author's original code. Since the training process of these models tends to corrupt after a convergence, we have always applied early stopping, but only for these methods. **SSWN** is the single-stream WeaveNet, which is equivalent to a DBA adopting WeaveNet' set-encoder. **WN** is the WeaveNet. Note that we adjusted the number of parameters of all the models for a fair comparison. See A.4 in the appendix for the further details.

This experiment restarted the training process five times with different random seeds. To accelerate each training, we have used RAdam optimizer (Liu et al., 2020) with an initial learning rate of 0.0001.

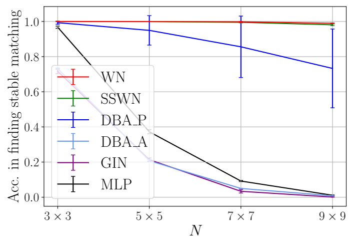

Figure 4: Accuracy in finding stable matching (↑) according to $N$

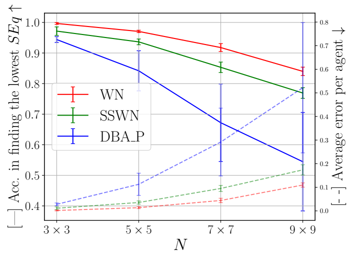

Figure 5: Accuracy in finding lowest $SEq$ matching (solid) and errors (dashed)

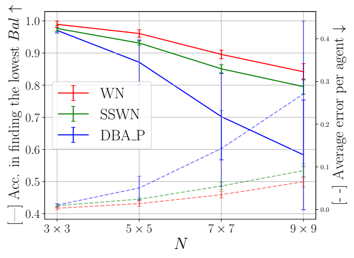

Figure 6: Accuracy in finding lowest $Bal$ matching (solid) and errors (dashed)

Fig. 4 shows the accuracy in finding a stable matching, where we trained models to minimize losses other than those related to fairness objectives (see A.1 in the appendix for details). Since MLP and GIN have size-dependency, we trained the models independently for $N = 3,\ 5,\ 7,\ 9$. The other models were trained with $N = 10$ and tested on $N = 3,\ 5,\ 7,\ 9$. As a result, MLP, GIN, and DBA_A could hardly find stable matching when $N \geq 5$. DBA_P performs better but is quite unstable. WN and SSWN could stably approximate the problem with a good performance.

Figs. 5 and 6 show the accuracy in finding the optimal solution with the two fairness objectives $SEq$ and $Bal$, respectively, with the average error per agent. We omitted MLP, GIN, and DBA_A since they cannot solve these advanced problems without finding any stable matching. WN, SSWN, and DBA_P are trained to maximize fairness objectives $SEq$ (for Fig. 5) and $Bal$ (for Fig. 6). In the

[4]We obtained the highest cost by the GS algorithm (prioritizing each side once and adopting the *worse* one).

results, both SSWN and WN largely outperformed DBA_P, which proved the advantage of the set-encoder. The performance gain of WN from SSWN shows the effect of the two-stream architecture, which is small in the range of $N < 10$.

### 4.1.5 Comparison with hand-crafted algorithms ($N = 20,\ 30$)

We prepared three hand-crafted algorithm baselines as gold standards. **GS** is the *better* result of applying the GS algorithm to prioritize each side once, which runs in $O(N^2)$. **DACC** by Dworczak (2016) is an approximate algorithm that runs in $O(N^4)$. **PowerBalance** by Tziavelis et al. (2019) is the state-of-the-art method that runs in $O(N^2)$.

In addition, we added **WN-asym**, the asymmetric variant of WeaveNet, which is applied to non-symmetric distributions. We implemented all the differentiable solvers with $L = 60$ layers. This time, we used the identical $N$ at training and test.

Table 1: Average fairness scores (↓) and the success rate of stable matching (↑) at $N = 20$. Bold and underlined scores shows the **best** and <u>second best</u> ones, respectively. The success rates in stable matching are colored in red if it is less than 95%.

| | *SEq* (*pSEq*) | | | | | *Bal* (*pBal*) | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | U | D | G | UD | Lib | U | D | G | UD | Lib |
| GS | 41.89 | 18.81 | 19.52 | **70.97** | 19.66 | 89.14 | 146.16 | 108.36 | **140.53** | 68.62 |
| DACC | 24.34 | 20.13 | 23.07 | 101.75 | 20.40 | 78.49 | 146.71 | 110.06 | 151.34 | 68.75 |
| Power Balance | 16.28 | 8.93 | <u>17.07</u> | <u>71.09</u> | 15.40 | <u>73.28</u> | 140.12 | <u>106.92</u> | <u>140.55</u> | 65.89 |
| WN | **12.16** | **6.53** | **15.56** | 82.48 | <u>14.59</u> | **72.33** | **138.75** | **106.65** | 142.37 | **65.82** |
| Stably Matched (%) | 99.10 | 99.40 | 99.40 | 96.00 | 99.50 | 98.00 | 99.10 | 98.60 | 97.30 | 98.90 |
| WN-asym | - | - | - | 71.34 | **14.53** | - | - | - | 140.79 | <u>65.84</u> |
| Stably Matched (%) | - | - | - | 99.50 | 99.80 | - | - | - | 99.80 | 99.10 |
| SSWN | <u>15.06</u> | <u>7.74</u> | 17.65 | 74.50 | 16.02 | 79.06 | <u>139.59</u> | 107.90 | 141.48 | 67.22 |
| Stably Matched (%) | 95.80 | 99.20 | 96.50 | 92.40 | 97.90 | 92.60 | 98.50 | 98.40 | 99.60 | 98.60 |
| DBA_P | 19.34 | 8.62 | 17.98 | 75.00 | 18.36 | 76.39 | 140.78 | 108.57 | 142.06 | 67.18 |
| Stably Matched (%) | 94.20 | 98.50 | 96.50 | 95.70 | 95.00 | 93.50 | 99.30 | 94.50 | 99.10 | 97.30 |

Table 2: Average fairness scores (↓) and success rate of stable matching (↑) at $N = 30$.

| | *SEq* (*pSEq*) | | | | | *Bal* (*pBal*) | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | U | D | G | UD | Lib | U | D | G | UD | Lib |
| GS | 94.03 | 43.46 | 36.56 | **163.77** | 39.78 | 184.05 | 322.05 | 225.49 | **312.12** | 137.59 |
| DACC | 40.87 | 34.35 | 40.59 | 240.48 | 33.88 | 150.71 | 316.18 | 227.52 | 337.43 | 133.59 |
| Power Balance | <u>18.45</u> | <u>11.05</u> | **27.22** | <u>163.90</u> | **21.57** | **138.04** | <u>302.30</u> | **220.26** | **312.12** | **126.96** |
| WN | **18.30** | **10.52** | <u>27.39</u> | 210.06 | <u>22.14</u> | <u>140.40</u> | **301.59** | <u>223.02</u> | 322.98 | <u>127.79</u> |
| Stably Matched (%) | 98.10 | 99.00 | 98.00 | 81.50 | 98.80 | 97.90 | 98.60 | 93.70 | 80.30 | 98.10 |
| WN-asym | - | - | - | 170.35 | 22.17 | - | - | - | 313.59 | 127.93 |
| Stably Matched (%) | - | - | - | 93.90 | 98.6 | - | - | - | 98.80 | 98.00 |
| SSWN | 34.34 | 14.57 | 33.91 | 197.04 | 27.25 | 169.22 | 304.52 | 225.06 | 317.31 | 132.19 |
| Stably Matched (%) | 90.00 | 97.50 | 89.00 | 69.20 | 92.30 | 79.80 | 96.70 | 93.90 | 94.00 | 92.80 |
| DBA_P | 52.93 | 16.21 | 37.27 | 189.07 | 36.63 | 167.26 | 308.45 | 227.45 | 320.45 | 133.82 |
| Stably Matched (%) | 84.60 | 96.00 | 85.20 | 82.70 | 79.20 | 74.10 | 92.20 | 82.80 | 89.30 | 87.20 |

Tables 1 and 2 show the results for $N = 20$ and 30, respectively. At $N = 20$, WN or WN-asym constantly performed better than any hand-crafted algorithms for both $SEq$ and $Bal$ except for UD. When $N = 30$, they are comparative with PowerBalance, but still outperform DACC. SSWN has a performance drop in the stable matching rate at $N = 30$, which shows the importance of parameter efficiency for larger $N$. For UD, GS performed even better than PowerBalance, which indicates that the ideal solution constantly prioritizes one side, and the input distribution is highly biased. In this situation, the symmetric WN suffers from finding good solutions. In contrast, WN-asym achieved similar performance to GS and PowerBalance, which proves the effectiveness of the split batch normalization and side-identifiable code for a biased situation.

### 4.1.6 Robustness against input distribution shift

We performed two additional experiments to prove the robustness of WeaveNet against the input distribution shift in $N$ and the distribution type.

First, Table 3 shows fairness scores obtained by WN trained with $N = 20$ and 30 samples, which we refer to WN(20) and WN(30), respectively. From the result, we confirmed that WN(30) works comparatively with WN(20) at $N = 20$, but WN(20) is slightly worse than WN(30) at $N = 30$. Even though, it is remarkable that WN(20) outperformed the DACC's scores (shown in Tables 1 and 2), that proves WeaveNet's robustness against the shift in $N$.

Table 3: Comparison of the models trained with different size of instances.

| | $20 \times 20$ | | | | | | $30 \times 30$ | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | $pSEq$ | | | $pBal$ | | | $pSEq$ | | | $pBal$ | | |
| | U | D | G | U | D | G | U | D | G | U | D | G |
| WN(20) | 12.23 | **6.37** | **15.50** | **71.89** | 138.79 | **106.20** | 25.21 | 11.38 | 29.36 | 141.49 | 302.73 | **221.92** |
| WN(30) | **12.16** | 6.53 | 15.56 | 72.33 | **138.75** | 106.65 | **18.30** | **10.52** | **27.39** | **140.40** | **301.59** | 223.02 |

Second, we evaluated the performance of models against the shift in distribution type. We prepared three models, WN(U), WN(D), and WN(G), that are trained with U, D, and G, respectively. Table 4 shows the evaluation in combination of distribution types. Interestingly,WN(D) could hardly find stable matching in U and G, which resulted in poor $pSEq$ scores. In contrast, WN(G) achieved satisfying $pSEq$ scores on U and D. Since G always generates preference lists based on a common reference score ($i/N$ for $i$-th agent) with random noise, preference lists in the training set are always similar and hard to assign appropriately. Hence, these results revealed that WeaveNet works well even under the shift in distribution type but only when trained with hard samples.

Table 4: Generalizability of WeaveNet against difference in distribution type ($N = 30$).

| | | U | D | G | Avg. |
|---|---|---|---|---|---|
| WN(U) | $pSEq$ | 18.30 | 25.81 | 29.09 | <u>21.10</u> |
| | Stably Matched (%) | 98.10 | 94.90 | 93.60 | 95.53 |
| WN(D) | $pSEq$ | 171.27 | 10.52 | 77.36 | 86.38 |
| | Stably Matched (%) | 2.80 | 99.00 | 0.10 | 33.97 |
| WN(G) | $pSEq$ | 21.38 | 12.85 | 27.39 | **20.54** |
| | Stably Matched (%) | 97.30 | 98.10 | 98.00 | 97.80 |

Table 5: Demonstration at $N = 100$.

| $100 \times 100$, U | $SEq$ | $Bal$ |
|---|---|---|
| GS | 1259.39 | 1709.53 |
| DACC | <u>194.65</u> | <u>988.02</u> |
| Power Balance | **49.41** | **909.73** |
| WN+Hungarian | | |
| Fairness costs | 68.36 | 919.75 |
| Pseudo fairness costs | 257.99 | 1145.36 |
| Stably Matched (%) | 89.40 | 80.80 |

### 4.1.7 Demonstration with $N = 100$

We further demonstrate the capability of WeaveNet under a larger size of problem instances, $N = 100$. We set $L = 80$ for this experiment to fully deepen the network as long as the GPU memory allows. In this case, we found that WeaveNet fails even in one-to-one matching for 13.4% and 19.8%, respectively. To cover these failures, we applied the Hungarian algorithm (Kuhn, 1955) to ensure $\mathbb{M}$ to be one-to-one matching (see B in the appendix for an additional analysis on this option). Table 5 shows that WeaveNet works even worse than DACC in this setting. Seeing $SEq$ and $Bal$ scores, the poor $pSEq$ and $pBal$ scores mostly derive from the poor stable matching rate (and large penalties for each violation due to the larger $N$). It may fix this problem to sample such violating data actively at training, for example, but such further explorations are beyond the scope of this paper.

## 4.2 Evaluation with 3D point cloud matching

To demonstrate the versatility of a differentiable solver, we applied WeaveNet and DBAs to 3D point cloud matching. As a SoTA base model, we adopted CorrNet3D (Zeng et al., 2021). We have replaced the affinity matrix calculation to WeaveNet (**CN3D+WN**). We prepared **CN3D+DBA_P** and **CN3D+DBA_A** in the same manner.

We have conducted the experiments for rigid shape correspondence with Surreal dataset (Groueix et al., 2018) with an unsupervised setting[5]. Due to the large memory consumption in constructing a full-connected bipartite graph for two point clouds, we sub-sampled 64 among 1024 points after the feature embedding module and evaluated their accuracy.

Figure 7 shows the result, where the accuracy (the vertical axis) is defined as the ratio of the samples whose error is in the error margin (the horizontal axis). In addition to the original model (**CN3D**), we prepared **CN3D-Deep**, a model with an additional feature embedding layer for a fair comparison (See C in the appendix for more details).

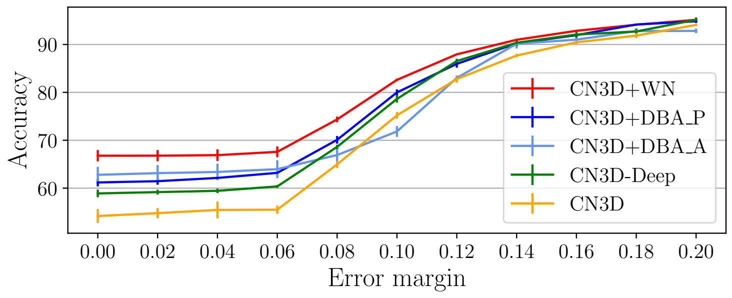

Figure 7: Comparison in 3D point cloud matching.

The results show that WeaveNet boosted the performance by 12.6 percentage points from the original CorrNet3D at zero error margin. It has also achieved a gain of 7.88 percentage points against the deeper model. These advantages were significant at a small error margin ($< 0.06$), indicating that WeaveNet could refine the last-one-mile errors of the assignment that are hard to solve only by training the feature extractor.

DBA_P has also boosted the performance, but its gain was smaller than WeaveNet. The accuracy of DBA_A was better than DBA_P at a small error margin but performed unstable at a large error margin, which indicates some negative effect by self-attention. Notably, the difference in performance does not contradict the test on stable matching. In other words, these results support the validity of the test protocol proposed in the previous subsection. At the same time, this experiment revealed a limitation by its GPU memory consumption. We expect that edge pruning can enable a full-size assignment, for example. Exploring the best practice for it is beyond the scope of this paper.

## 5 Conclusion

This paper proposed a novel differentiable assignment solver, *WeaveNet*, with an evaluation protocol on stable matching and its *strongly NP-hard* variants. In the experiments, we demonstrated the advantage of *set encoder* and the two-stream architecture by *cross-concatenation* against the other possible differentiable solvers. These techniques also achieved a better performance than the gold standard of the state-of-the-art hand-crafted algorithm at $N = 20$ and a comparative performance at $N = 30$. Furthermore, the asymmetric variants, *split batch normalization* with the *side-identifiable code*, enabled WeaveNet to work even with the strongly biased dataset of UD despite the architecture's symmetric nature.

We have also confirmed that our method can collaborate with a contemporary method for computer vision, CorrNet3D. The result shows that WeaveNet can significantly boost the performance of 3D point cloud matching through a joint optimization with the feature extractor. We hope this study opens a new vista for various machine learning applications that potentially involve non-linear assignment problems.

[5]We could not bring out supervised setting and non-rigid cases because the provided code does not support them.

## ETHICS STATEMENT

This paper uses two existing datasets; LibimSeTi dataset (Brožovský & Petříček, 2007) and Surreal dataset (Groueix et al., 2018). The former is obtained by hiring participants for dataset creation, which does not contain any private information and has no ethical problems. The latter is a synthetic human dataset, which also has no ethical issues.

As discussed in 3.3, the proposed method mathematically ensures completely fair treatment in a non-linear assignment, which is a highly ethical tool contributing to human well-being. On the other hand, we can intentionally (or accidentally) train its asymmetric variant (or any other baseline differential solvers in this paper) to yield an unfair assignment. Hence, we must carefully check the training protocol to avoid undesirable bias when using asymmetric differential solvers in any social application.

## REPRODUCIBILITY STATEMENT

To ensure the reproducibility, we attached the code for all the experiments as the supplementary material, which will be published with the paper. There is also general description to reproduce the results in 4.1.3, A.2, A.4, and C. Furthermore, we restart the training five times to obtain the results with error bars in Figures 4, 5, 6, and 7 in the main text and Figures 8, 9, 10, 11 in the appendix. These data prove the low variance of the WeaveNet performance. We generated some datasets in this paper. Due to the size limitation, we could not include the datasets in the supplementary material. Instead, we included random seeds for generating the same validation and test sets in the attached code. We will make both the code and the generated datasets publicly available.

## REFERENCES

Lukáš Brožovský and Václav Petříček. Recommender system for online dating service. In *Proceedings of Znalosti 2007*, pp. 29–40, 2007.

Piotr Dworczak. Deferred acceptance with compensation chains. In *Proceedings of the ACM Conference on Economics and Computation*, pp. 65–66, 2016.

Mohamed El Banani, Luya Gao, and Justin Johnson. UnsupervisedR&R: Unsupervised pointcloud registration via differentiable rendering. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 7129–7139, 2021.

Tomás Feder. *Stable networks and product graphs*. American Mathematical Society, 1995.

Kexue Fu, Shaolei Liu, Xiaoyuan Luo, and Manning Wang. Robust point cloud registration framework based on deep graph matching. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 8893–8902, 2021.

David Gale and Lloyd S Shapley. College admissions and the stability of marriage. *The American Mathematical Monthly*, 69(1):9–15, 1962.

Quankai Gao, Fudong Wang, Nan Xue, Jin-Gang Yu, and Gui-Song Xia. Deep graph matching under quadratic constraint. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 5069–5078, 2021.

Vijay Kumar Garg and Changyong Hu. Improved paths to stability for the stable marriage problem. *CoRR*, abs/2007.07121, 2020. URL `https://arxiv.org/abs/2007.07121`.

Daniel Gibbons, Cheng-Chew Lim, and Peng Shi. Deep learning for bipartite assignment problems. In *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics*, pp. 2318–2325, 2019.

Steven Gold and Anand Rangarajan. A graduated assignment algorithm for graph matching. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 18(4):377–388, 1996.

Thibault Groueix, Matthew Fisher, Vladimir G. Kim, Bryan Russell, and Mathieu Aubry. 3D-CODED : 3D correspondences by deep deformation. In *Proceedings of the European Conference on Computer Vision*, pp. 230–246, 2018.

Sushmita Gupta, Sanjukta Roy, Saket Saurabh, and Meirav Zehavi. Balanced stable marriage: How close is close enough? *Theoretical Computer Science*, 883:19–43, 2021.

Dan Gusfield and Robert W Irving. *The stable marriage problem: structure and algorithms*. MIT press, 1989.

Jiawei He, Zehao Huang, Naiyan Wang, and Zhaoxiang Zhang. Learnable graph matching: Incorporating graph partitioning with deep feature learning for multiple object tracking. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 5299–5309, 2021.

Akiko Kato. Complexity of the sex-equal stable marriage problem. *Japan Journal of Industrial and Applied Mathematics*, 10(1):1, 1993.

Harold W Kuhn. The hungarian method for the assignment problem. *Naval research logistics quarterly*, 2(1-2):83–97, 1955.

Qimai Li, Zhichao Han, and Xiao-Ming Wu. Deeper insights into graph convolutional networks for semi-supervised learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pp. 3538–3545, 2018.

Shira Li. *Deep Learning for Two-Sided Matching Markets*. Bachelor's thesis, Harvard University, 2019.

Liyuan Liu, Haoming Jiang, Pengcheng He, Weizhu Chen, Xiaodong Liu, Jianfeng Gao, and Jiawei Han. On the variance of the adaptive learning rate and beyond. In *Proceedings of the International Conference on Learning Representations*, 2020. URL `https://openreview.net/forum?id=rkgz2aEKDr`.

Miguel Angel Lozano and Francisco Escolano. Graph matching and clustering using kernel attributes. *Neurocomputing*, 113:177–194, 2013.

Eric McDermid and Robert W Irving. Sex-equal stable matchings: Complexity and exact algorithms. *Algorithmica*, 68(3):545–570, 2014.

Gonzalo Mena, David Belanger, Scott Linderman, and Jasper Snoek. Learning latent permutations with gumbel-sinkhorn networks. In *Proceedings of the International Conference on Learning Representations*, 2018. URL `https://openreview.net/forum?id=Byt3oJ-0W`.

Kenta Oono and Taiji Suzuki. Graph neural networks exponentially lose expressive power for node classification. In *Proceedings of the International Conference on Learning Representations*, 2020. URL `https://openreview.net/forum?id=S1ldO2EFPr`.

Charles R. Qi, Hao Su, Kaichun Mo, and Leonidas J. Guibas. PointNet: Deep learning on point sets for 3D classification and segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 652–660, 2017.

Alvin E Roth, Tayfun Sönmez, and M Utku Ünver. Efficient kidney exchange: Coincidence of wants in markets with compatibility-based preferences. *American Economic Review*, 97(3):828–851, 2007.

Daniel Selsam, Matthew Lamm, Benedikt Bünz, Percy Liang, Leonardo de Moura, and David L. Dill. Learning a SAT solver from single-bit supervision. In *Proceedings of the International Conference on Learning Representations*, 2019. URL `https://openreview.net/forum?id=HJMC_iA5tm`.

Nikolaos Tziavelis, Ioannis Giannakopoulos, Katerina Doka, Nectarios Koziris, and Panagiotis Karras. Equitable stable matchings in quadratic time. In *Proceedings of the Advances in Neural Information Processing Systems*, pp. 457–467, 2019.

Po-Wei Wang, Priya Donti, Bryan Wilder, and Zico Kolter. SATNet: Bridging deep learning and logical reasoning using a differentiable satisfiability solver. In *Proceedings of the International Conference on Machine Learning*, pp. 6545–6554, 2019.

Olivia Wiles, Sebastien Ehrhardt, and Andrew Zisserman. Co-attention for conditioned image matching. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 15920–15929, 2021.

Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *Proceedings of the International Conference on Learning Representations*, 2019. URL `https://openreview.net/forum?id=ryGs6iA5Km`.

Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Russ R Salakhutdinov, and Alexander J Smola. Deep sets. In *Proceedings of the Advances in Neural Information Processing Systems*, pp. 3394–3404, 2017.

Yiming Zeng, Yue Qian, Zhiyu Zhu, Junhui Hou, Hui Yuan, and Ying He. CorrNet3D: Unsupervised end-to-end learning of dense correspondence for 3D point clouds. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 6052–6061, 2021.

# WEAVENET: A DIFFERENTIABLE SOLVER FOR NON-LINEAR ASSIGNMENT PROBLEMS: APPENDIX

## A ADDITIONAL DETAILS FOR THE TEST PROTOCOL WITH STABLE MATCHING

This appendix section explains further details of the test protocol to enhance reproducibility and show how we designed the experiments fairly.

### A.1 UNSUPERVISED LEARNING FOR STABLE MATCHING

The preference list $\boldsymbol{P}$ and $\boldsymbol{Q}$ ranges $[1, N]$, depending on the size of the problem instance. Such inputs are not preferable since any machine learning method can poorly perform with an unseen value range and harm robustness against the shift in $N$. Hence, we normalize them to have $(0, 1]$ regardless of $N$ by composing $\mathbf{P}, \mathbf{Q} \in \mathbb{R}^{N\times N\times 1}$ as

$$\begin{aligned}\mathbf{P}_{v,w,:} &= ((1 - C_{min})(N - P_{v,w})/N + C_{min})\\ \mathbf{Q}_{w,v,:} &= ((1 - C_{min})(N - Q_{w,v})/N + C_{min}),\end{aligned} \tag{4}$$

where $C_{min}$ is a constant value in range $(0, 1)$ and is set to 0.1 in our experiments. The above normalization inverts the rank into a preference score (higher is preferred more). This inversion is introduced to follow the loss formulation in Li (2019).

The constraint of stable matching is originally defined as a non-differentiable discrete function. To optimize the model fully differentiable without accessing ground truth, we relax such discrete functions into continuous ones. For example, we can realize the loss function for a stable matching solver that minimizes $SEq$ as a linear sum of the following three losses.

$\mathcal{L}_m$ conditions the binarization of $\boldsymbol{M}$ to be doubly stochastic.

$\mathcal{L}_s$ conditions the matching to be stable.

$\mathcal{L}_f$ minimizing the fairness cost $SEq$ of the matching

Considering the symmetricity of the bipartite graph, we merge the three losses as

$$\mathcal{L}_{\mathrm{fsm}}(\boldsymbol{M}) = \lambda_m\mathcal{L}_m(\boldsymbol{M}) + \frac{1}{2}\sum_{\hat{\boldsymbol{M}}\in\{\sigma(\boldsymbol{M}),\sigma(\boldsymbol{M}^{\mathrm{tr}})\}} \big(\lambda_s\mathcal{L}_s(\hat{\boldsymbol{M}}) + \lambda_f\mathcal{L}_f(\hat{\boldsymbol{M}})\big), \tag{5}$$

where $\sigma$ is a function that applies the softmax operation in the horizontal direction (row by row), and $\lambda_*$ is the weight for each loss function.

An essential advantage of a differentiable solver is its flexibility; we can optimize a model for different problems only by modifying the loss functions. For example, removing $\mathcal{L}_f$ in Eq.(5) leads to the standard stable matching, or replacing $\mathcal{L}_f$ with $\mathcal{L}_b$ leads to the balanced stable matching, which we can formulate as

$$\mathcal{L}_{\mathrm{sm}}(\boldsymbol{M}) = \lambda_m\mathcal{L}_m(\boldsymbol{M}) + \frac{1}{2}\sum_{\hat{\boldsymbol{M}}\in\{\sigma(\boldsymbol{M}),\sigma(\boldsymbol{M}^{\mathrm{tr}})\}} \lambda_s\mathcal{L}_s(\hat{\boldsymbol{M}}), \tag{6}$$

$$\mathcal{L}_{\mathrm{bsm}}(\boldsymbol{M}) = \lambda_m\mathcal{L}_m(\boldsymbol{M}) + \frac{1}{2}\sum_{\hat{\boldsymbol{M}}\in\{\sigma(\boldsymbol{M}),\sigma(\boldsymbol{M}^{\mathrm{tr}})\}} \big(\lambda_s\mathcal{L}_s(\hat{\boldsymbol{M}}) + \lambda_b\mathcal{L}_b(\hat{\boldsymbol{M}})\big). \tag{7}$$

**One-to-one matching constraint** We can safely binarize $\boldsymbol{M}$ by column-wise or row-wise $\mathrm{argmax}$ operation when $\boldsymbol{M}$ is a doubly stochastic matrix. To reach such a solution, Li (2019) proposed to constrain $\boldsymbol{M}$ to be doubly stochastic by minimizing divergence between by mean absolute error (MAE) between $\sigma(\boldsymbol{M})$ and $\sigma(\boldsymbol{M}^{\mathrm{tr}})$. We follow this strategy but with a small modification; we apply cosine distance to $\exp(\boldsymbol{M})$ to allow $\boldsymbol{M}$ smoothly rotate to search for a better solution while keeping $\mathcal{L}_m = 0$ during training. This modification achieved slightly better performance than MAE in our preliminary experiment with an MLP model.

Overall, $\mathcal{L}_m$ is formulated as

$$\mathcal{L}_m(\boldsymbol{M}) = 1 - \frac{1}{N}\sum_{v=0}^{N}\sum_{w=0}^{N} \frac{\exp(M_{v,w})}{\|\exp(M_{v,:})\|_2} \cdot \frac{\exp(M_{v,w})}{\|\exp(M_{:,w})\|_2}, \tag{8}$$

where $M_{v,:}$ and $M_{:,w}$ means the $v$-th row and $w$-th column of $\boldsymbol{M}$. Thus, the second term on the right side calculates an average of $\exp(\boldsymbol{M})$'s row-wise and column-wise cosine similarity for the exponential of $\boldsymbol{M}$.

$\mathcal{L}_m(\boldsymbol{M})$ is zero typically for a uniform matrix, a permutation matrix, or linear sum of these two types of matrice (but not limited to them). By suppressing blocking pairs, it goes apart from the uniform matrix and thus reaches a permutation matrix, which results in a one-to-one matching.

**Blocking pair suppression** As for $L_s$, we used the function proposed in Li (2019) as it is. The function is defined as

$$\begin{aligned}\mathcal{L}_s(\boldsymbol{M};\mathsf{P},\mathsf{Q}) =& \sum_{(v,w)\in\mathbb{A}} g(v;w,\boldsymbol{M})g(w;v,\boldsymbol{M})\\ g(v;w,\boldsymbol{M}) =& \sum_{w'\neq w} M_{v,w'}\cdot\max(\mathsf{P}_{v,w}-\mathsf{P}_{v,w'},0)\\ g(w;v,\boldsymbol{M}) =& \sum_{v'\neq v} M_{v',w}\cdot\max(\mathsf{Q}_{w,v}-\mathsf{Q}_{w,v'},0),\end{aligned} \tag{9}$$

where $g(v;w,\boldsymbol{M})$ is a criterion known as ex-ante justified envy, which has a positive value if and only if there is any $w'$ that satisfies $\boldsymbol{M}_{v,w'} > 0$ and is preferred more than $w$. This is the same for $g(w;v,\boldsymbol{M})$. Therefore, we can penalize a (soft) blocking pair $\{v,w\}$ that makes both $g(v;w,\boldsymbol{M})$ and $g(w;v,\boldsymbol{M})$ more than zero.

**Fairness measurements** $\mathcal{L}_f$ and $\mathcal{L}_b$ minimize $SEq(\boldsymbol{M};\boldsymbol{P},\boldsymbol{Q})$ and $Bal(\boldsymbol{M};\boldsymbol{P},\boldsymbol{Q})$, respectively, and are defined as

$$\begin{aligned}\mathcal{L}_f(\boldsymbol{M};\boldsymbol{P},\boldsymbol{Q}) &= \frac{1}{N}|\rho(\boldsymbol{M};\boldsymbol{P}) - \rho(\boldsymbol{M};\boldsymbol{Q})|\\ \mathcal{L}_b(\boldsymbol{M};\boldsymbol{P},\boldsymbol{Q}) &= -\frac{1}{N}\min(\rho(\boldsymbol{M};\boldsymbol{P}),\rho(\boldsymbol{M};\boldsymbol{Q})),\end{aligned} \tag{10}$$

where

$$\rho(\boldsymbol{M};\boldsymbol{P}) = \sum_{v=1}^{N}\sum_{w=1}^{N} M_{v,w}\mathsf{P}_{v,w}, \quad \rho(\boldsymbol{M};\mathsf{Q}) = \sum_{v=1}^{N}\sum_{w=1}^{N} M_{v,w}\mathsf{Q}_{w,v}. \tag{11}$$

For the output at inference, we binarize $\boldsymbol{M}$ row by row. There is another option to use the Hungarian algorithms or Paths-to-stability [6] algorithms (Garg & Hu, 2020), which may perform better. However, we avoid using such error corrective options in default since it will cover the mistakes of a model and hide any flaws of the model, which contradicts the purpose of the test protocol to evaluate the model performance for general assignment problems. Furthermore, a paths-to-stability algorithm must access complete input information, which is unavailable with latent vector inputs or online assignment problems).

## A.2 Loss weights

Through the experiments, we set the loss weights $\lambda_m = 1.0$, $\lambda_s = 0.7$, and $\lambda_f = \lambda_b = 0.01$ to train any differentiable solvers. We adjusted these parameters with the validation sets in the following process.

As a preliminary model, we prepared WN-15 (WN with $L = 15$) with a set encoder with $D = 64$ and $D' = 256$. In this investigation, we used a balanced distribution U and the

[6] *Paths to stability* is a problem to find any editing path to reach stable matching from a non-stable one-to-one matching.

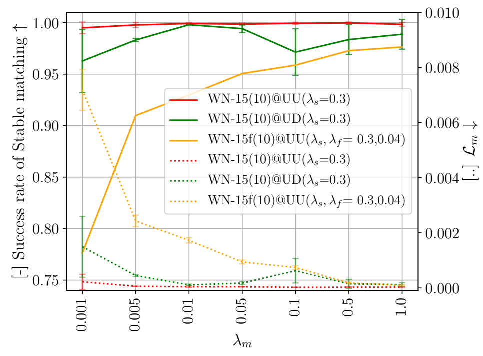

Figure 8: Sensitivity against $\lambda_m$

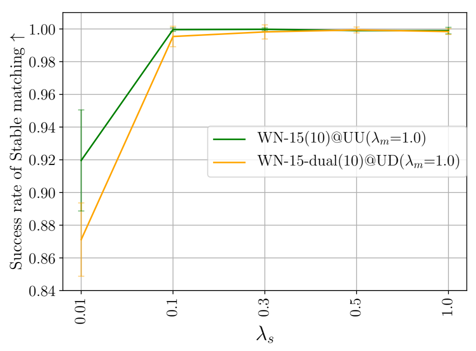

Figure 9: Sensitivity against $\lambda_s$

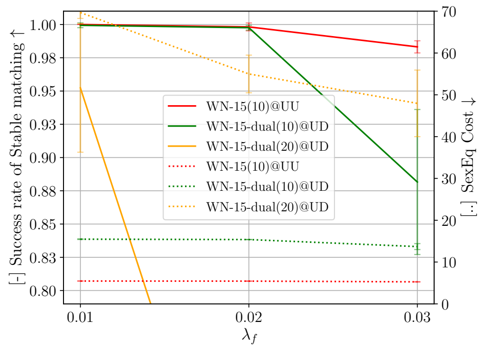

Figure 10: Sensitivity against $\lambda_f$

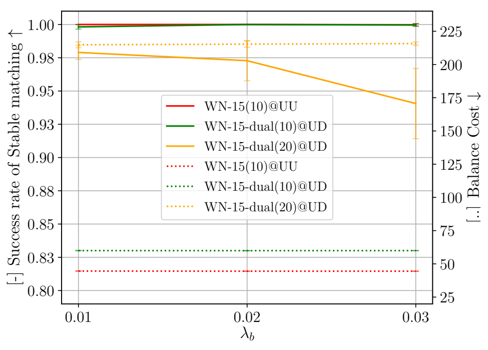

Figure 11: Sensitivity against $\lambda_b$

most biased distribution UD. With this model, we first tried to fix $\mathcal{L}_m$ because we experimentally found the tendency that the model hardly outputs a stably matched solution without minimizing $\mathcal{L}_m$. Fig. 8 shows the success rate of stable matching with different $\mathcal{L}_m$ in the range $\{0.001,\ 0.005,\ 0.010,\ 0.050,\ 0.100,\ 0.500,\ 1.000\}$, where WN-15($n$) is the model trained and validated with the samples of $N = n$. From this result, we decided to set $\lambda_m = 1.0$ (with the initial learning rate of 0.0001) and use it as the maximum weight among the loss weights.

Next, fixing $\lambda_m = 1.0$, we observed the trend in success rate of stable matching against $\lambda_s$. Fig. 9 shows our investigation of $\mathcal{L}_s$ in the range $\{0.01, 0.10, 0.30, 0.50, 1.00\}$. Here, WN-dual is a prototype of WeaveNet's asymmetric variant, in which each stream has an independent set-encoder at each layer. As a result, we found that $\lambda_s$ should simply be large enough (ca. $\lambda_s \geq 0.30$) and decided to use 0.7 with a safety margin against the constraint violation.

To investigate the sensitivity of the model against $\lambda_f$ and $\lambda_b$, we tentatively removed the safety margin and set $\lambda_s = 0.3$, which is the minimum satisfiable value. Figures 10 and 11 show the results. From these figures, we found that too-strong weights for fairness may disrupt stability. Thus, we decided to use $\lambda_f = \lambda_b = 0.01$, which achieved the highest success rate of stable matching in the search range of $\{0.01, 0.02, 0.03\}$.

Table 6: Comparison in parameter efficiency among different shape architectures. Deep achieved the best success rate in stable matching despite its smallest architecture.

| Name | $L$ | $D$ | $D'$ | # of params. | Stably Matched (%) |
|---|---|---|---|---|---|
| Deep | 30 | 22 | 44 | 117k | 95.7% |
| Wide1 | 15 | 32 | 64 | 119k | 76.0% |
| Wide2 | 15 | 24 | 98 | 120k | 73.1% |

## A.3 Analysis on the shape of the WeaveNet architecture

Table 6 shows the success rate of stable matching after 100k iterations of training. Here, we drew samples from the U distribution with $N = 30$ for both training and validation. The result shows that the deepest model performs the best despite its smallest number of parameters. Hence, we decided to use the set encoder that has $D \leq 32$ and $D' = 2D$.

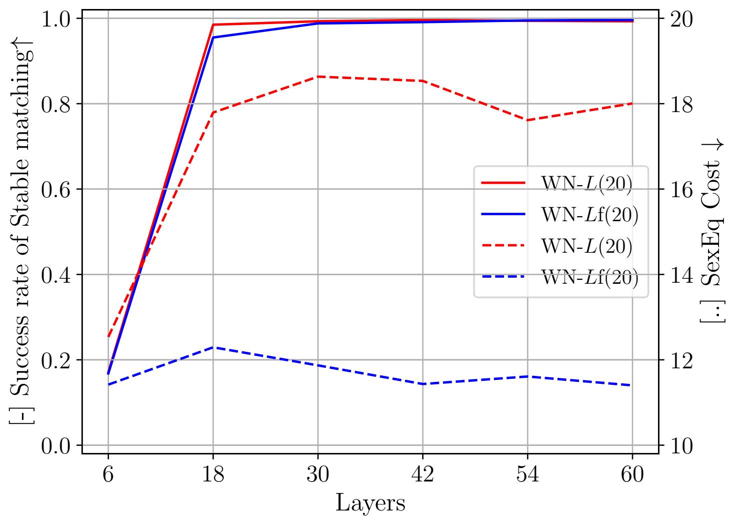

Figure 12: Success rate of stable matching (solid, ↑) and $SEq$ (dashed, ↓) according to $L$.

We further investigated the impact of network depth on the problem. To observe how the strongly NP-hard target increases the difficulty in optimization, we plotted both results by WN-$L$ and WN-$L$f with $L \in \{6, 18, 30, 42, 54, 60\}$. Fig. 12 shows the trend of success rate against different $L$. Here, the models were trained and validated with the U distribution at $N = 20$. We can see from the result that $L = 6$ is not enough to stably match samples of $N = 20$, but $L = 18$ is enough if only for that purpose. Besides, we observed that a deeper stack of layers tends to improve $SEq$ slightly.

## A.4 Network architecture for a fair comparison

To avoid a complicated architecture search for all the baseline models, we used a common shape for all $L$ layer-level encoders regardless of the method. With this setting, we have at most three hyper-parameters to decide the architecture. $L$ is the number of layers, and $D$ is the number of output channels at each layer. WN and SSWN have an additional parameter $D'$, the number of output channels of $\phi_1$ in the set-encoder. Similarly, DBA_A also has an additional parameter that corresponds to the channels of key and query. We also refer it to $D'$. Note that we repeat the shortcut path for the residual structure regularly for every two layers.

Table 7 summarizes the hyper-parameters for the differentiable solvers including baselines. **MLP** could not converge when more than three hidden layers are stacked. Hence, we set the number of layers to be three, as used in Li (2019). For **GIN** (Xu et al., 2019), we put two graph convolutional layers followed by a single linear layer, which output $N^2$ channels of $\boldsymbol{M}$. The number of GCN layers is decided based on the original paper, which reported it performs best for a node classification task. Some other papers also pointed out the over-smoothing problem of GCNs (Li et al., 2018; Oono & Suzuki, 2020), which is not a problem for graph classification, but seriously damages the results in node classification. Our preliminary experiments also support that we can not boost the performance by deepening GCN.

Table 7: Architecture of each model. $D'$ represents the output channels of $\phi_1$ for the set-encoder and the length of key and query features for self-attention. Since MLP and GIN parameters differ for $N$, we show here cases with $N = 5$.

| Model | Encoder | $D$ | $D'$ | Residual Structure | # of params. |
|---|---|---|---|---|---|
| MLP | dense layer | 200 | - | w/o | 97k |
| GIN | graph conv. | 70 | - | w/o | 79k |
| DBA_P($L = 10$) | max pooling | 56 | - | w/ | 77k |
| DBA_A($L = 10$) | self-attention | 56 | 48 | w/ | 80k |
| SSWN($L = 10$) | set encoder | 64 | 32 | w/ | 78k |
| WN($L = 10$) | set encoder | 32 | 64 | w/ | 77k |
| DBA_P($L = 60$) | max pooling | 64 | - | w/ | 725k |
| SSWN($L = 60$) | set encoder | 64 | 64 | w/ | 740k |
| WN($L = 60$) | set encoder | 32 | 64 | w/ | 493k |
| WN($L = 80$) | set encoder | 32 | 64 | w/ | 659k |

Table 8: Numbers of blocking pairs in the estimated matching with U at $N = 100$. Fail counts outputs that are not a one-to-one matching.

| #Block. Pairs | WN-80f | +Hung. | WN-80b | +Hung. |
|---|---|---|---|---|
| 0 (stable) | 84.4% | 89.4% | 73.2% | 80.8% |
| 1 | 2.2% | 4.6% | 6.7% | 10.9% |
| 2 | 0.0% | 0.4% | 0.3% | 1.2% |
| $\geq 3$ | 0.0% | 5.6% | 0.0% | 7.1% |
| Fail | 13.4% | - | 19.8% | - |

## B EFFECT OF THE HUNGARIAN ALGORITHM AT $N = 100$

To obtain further insight, we prepared Table 8, which summarizes the difference w/ and w/o the Hungarian algorithm at $N = 100$. In this setting, we have more agents with which WN fails even to obtain one-to-one matching by the $\operatorname{argmax}$ binarization. Binarization by the Hungarian algorithm can force one-to-one output even such failure cases; however, for $SEq$, we obtained only 5.0% gain of success cases from the 13.4% fail cases. Similarly, for $Bal$, we obtained 7.5% success gain from 19.8% fail cases. These results confirmed that the Hungarian algorithm can improve solution quality but only for a limited volume of hard samples.

## C TRAINING PROTOCOL FOR 3D POINT CLOUD MATCHING

In the experiment in 4.2, we set a batch size of 50. We trained models 30 epochs and confirmed that all the models reached convergence. We preserved any other training conditions as in the original code of CorrNet3D. For a fair comparison, we adjusted the models to have a similar number of parameters (see Table 9.)

Table 9: Architecture of each model. $D'$ represents the output channels of $\phi_1$ for the set-encoder and the length of the key and query vectors for self-attention.

| Model | $L$ | $D$ | $D'$ | # of params. |
|---|---|---|---|---|
| CN3D | - | - | - | 3.1M |
| CN3D-Deep | - | - | - | 3.8M |
| CN3D-DBA_P | 6 | 16 | - | 3.1M |
| CN3D-DBA_A | 6 | 16 | 16 | 3.1M |
| CN3D-WN | 6 | 16 | 16 | 3.2M |