

Learning Non-Parametric Invariances from Data with Permanent Random Connectomes: Supplementary Material

Dipan K. Pal
dipanp@andrew.cmu.edu

Akshay Chawla
akshaych@andrew.cmu.edu

Marios Savvides
marioos@andrew.cmu.edu

Dept. Electrical and Computer Engg.
Carnegie Mellon University
Pittsburgh, PA, USA

Abstract

In this supplementary material, we provide a proof for Lemma 2.1 in the main paper, a more complete table of results including different parameter comparisons of NTPN baselines, timing results of our more efficient CUDA implementation and the pseudocode for PRCN-NPTNs. Further, we present results on the dataset ETH-80 consisting of 3D viewpoint variations of objects in comparison with previous work. Finally, we present results on CIFAR 10 with vanilla DenseNets and PRCN applied to DenseNets to form DensePRC-NPTNs.

1 Appendix

Prior Art using Alternate Architectures. Several works have explored alternate deep layer architectures. A few of the main developments were the application of the skip connection [8], depthwise separable convolutions [10] and group convolutions [9]. Randomly initialized channel shuffling is an operation that is central to the application of permanent random connectomes. However, *deterministic* non-randomized channel shuffling was explored while optimizing networks for computation efficiency [8]. Nonetheless, none of these methods explored *permanent* and *random* connectomes from the perspective of explicitly learning invariances from data itself.

Invariances in a PRC-NPTN layer. Recent work introducing NPTNs [8] had highlighted the Transformation Network (TN) framework in which invariance is generated during the forward pass by pooling over dot-products with transformed filter outputs. A vanilla convolution layer with a single input and output channel (therefore a single convolution filter) followed by a $k \times k$ *spatial* pooling layer can be seen as a single TN node enforcing translation invariance with the number of filter outputs being pooled over to be $k \times k$. It has been shown that $k \times k$ spatial pooling over the convolution output of a single filter is an approximation to channel pooling across the outputs of $k \times k$ translated filters [8]. The output

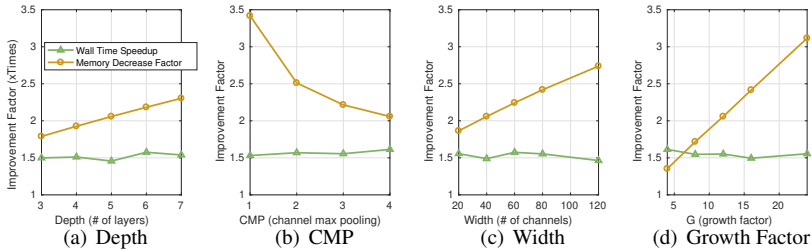


Figure 1: Computational efficiency improvements of our CUDA kernel implementations.

$\Upsilon(x)$ of such an operation with an input patch x can be expressed as

$$\Upsilon(x) = \max_{g \in \mathcal{G}} \langle x, gw \rangle \quad (1)$$

where \mathcal{G} is the set of filters whose outputs are being pooled over. Thus, \mathcal{G} defines the set of transformations and thus the invariance that the TN node enforces. In a vanilla convolution layer, this is the translation group (enforced by the convolution operation followed by *spatial* pooling). An NPTN removes any constraints on \mathcal{G} allowing it to approximately model arbitrarily complex transformations. A vanilla convolution layer would have *one* filter whose convolution is pooled over spatially (for translation invariance). In contrast, an NPTN node has $|\mathcal{G}|$ *independent* filters whose convolution outputs are pooled across *channel* wise leading to general invariance.

A PRC-NPTN layer inherits the property from NPTNs to learn arbitrary transformations and thereby arbitrary invariances using \mathcal{G} . Individual channel max pooling (CMP) nodes act as NPTN nodes sharing a *common* filter bank as opposed to independent and disjoint filter banks for vanilla NPTNs. This allows for greater activation sharing, where transformations learned from data through one subset of filters can be used for invoking similar invariances in a parallel computation path. This sharing and reuse of activation maps allows for higher parameter and sample efficiency. As we find in our experiments, randomization plays a critical role here, allowing for a simple and quick approximation to obtaining high performing invariances. A high activation map can activate multiple CMP nodes, winning over multiple sub-sets of low activations. Gradients flow back to these winning activations updating the filters to further model the features observed during that particular batch. Note that, CMP nodes in the same layer can pool over disjoint subsets to invoke a variety of invariances, leading to a more versatile network and also better modelling of a particular kind of invariance as we find in our experiments. Further, the primary source of invoking invariances in NPTN was understood to be the symmetry of the unitary group action space [8]. General invariances were assumed to be only approximately forming a group. Lemma 1.1 shows that group symmetry is not necessary to reduce variance of the quantity $\max \Upsilon(x)$ due to the action of the set elements g on some test input patch x . Though, the result makes a strong assumption regarding the distribution of $\Upsilon(x)$, it to the best of our knowledge the first result of its kind to show increased invariance without a group symmetric action.

Efficacy on CIFAR10 Image Classification. MNIST was a good candidate for the previous experiment where the addition of nuisance transformations such as translation and rotation did not introduce any artifacts. However, in order to validate permanent random connectomes on more realistic data, we utilize the CIFAR10 dataset and AutoAugmentation [2] as the nuisance transformation. Note that, from the perspective of previous works in network invariance, it is unclear how to hand craft architectures to handle invariances due

to the variety of transformations that AutoAugment invokes. Here is where the general invariance learning capability of PRC-NPTNs would help, without the need of expertise in such hand-crafting.

We replace vanilla convolution layers with kernel size 3 in DenseNets with PRC-NPTNs without the 2-layered pooling networks. There was another modification for this experiment. For each input channel of a layer, a total of $|G| = 12$ filters were learnt. However only a few of them were pooled over (channel max pool or CMP). We pool with CMP = 1, 2, 3 or 4 channels randomly keeping $|G| = 12$ fixed always. Note that in contrast with the MNIST experiment, pooling was always done over $|G|$ number of channels (CMP= $|G|$). This provides a different setting under which PRC-NPTN can be utilized. All models in this experiment were trained with AutoAugment and were tested on both a) the original testing images and also on b) the test set transformed by AutoAugment. Similarly to the previous experiment, a model would have learn invariance towards these auto-augment transformations in order to perform well. All DenseNet models have 12 layers with the PRC-NPTN variant having the same number of parameters to enable us to perform multiple runs in a reasonable amount of time. The lower accuracy compared to other studies can be accounted by this. We train 5 models for each setting and report the mean and standard deviation of the errors. Training 5 runs for each of the hyperparameter combination to account for the randomization is yet another reason which tended to result in unreasonably large experiment times. Importantly, the goal of this experiment is not to push the state-of-the-art, but rather to investigate the behavior of DensePRC-NPTNs within the limits of computational resources available for this study while executing 5 runs for each network.

Discussion. Table. 1 presents the results of this experiment. We find PRC-NPTN provides clear benefits even with architectures employing heavy use of skip connections such as DenseNets with the same number of parameters. Performance seems to increase as channel max pooling increased. Further, randomization seems to be important to the overall architecture even when given the complex nature of real image transformations. PRC-NPTN helps DenseNets account for nuisance transformations better even for those as extreme as auto-augment with its 16 transformation types ShearX/Y, TranslateX/Y, Rotate, AutoContrast, Invert, Equalize, Solarize, Posterize, Contrast, Color, Brightness, Sharpness, Cutout, Sample Pairing to various degrees. With these evidence, it is interesting to find that random connectomes can be motivated from the perspective of learning heterogeneous invariances from data without any change in architectures. We find that they provide a promising alternate dimension in future network design in contrast to the ubiquitous use of highly structured and ordered connectomes.

Evaluation on the ETH-80 dataset The ETH-80 dataset was introduced in [8] as a benchmark to test models against 3D pose variation of various objects. The dataset contains 80 objects belonging to 8 different classes. Each object has images from different viewpoints on a hemisphere for a total of 41 images per object. The images were resized to 50×50 following [8]. This dataset is perfectly poised to test how efficiently a model can learn invariance to 3D viewpoint variation.

Protocol: For this experiment, we devise a new protocol in which we train on one half of the horizontal views and test on the other half. We randomly split the vertical views between these train and test sets. We test with the same set of architectures as in the main paper with the same experimental settings. This protocol is harder since the other side of the object is not seen along with the objects not being symmetrical. Results are shown in Tab. 2. The results follow a similar trend to that of the original protocol described in the main paper. We find PRC-NPTN outperforms all other methods using a almost 3X less parameters and

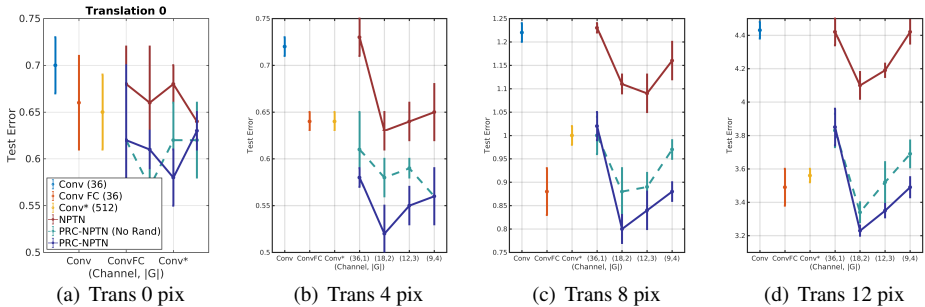


Figure 2: **Individual Transformation Results:** Test error statistics with mean and standard deviation on MNIST with progressively extreme transformations with **random pixel shifts**. For PRC-NPTN and NPTN the brackets indicate the number of channels in the layer 1 and G . ConvNet FC denotes the addition of a 2-layered pooling 1×1 pooling network after every layer. Note that for this experiment, $CMP=|G|$. Permanent Random Connectomes help with achieving better generalization despite increased nuisance transformations.

Method	CIFAR10	(w/o Random)	CIFAR 10++	(w/o Random)
DenseNet-Conv	11.47 ± 0.19	-	21.37 ± 0.29	-
DensePRC-NPTN (CMP=1)	11.82 ± 0.20	13.33 ± 0.23	22.03 ± 0.08	23.88 ± 0.38
DensePRC-NPTN (CMP=2)	10.78 ± 0.31	11.67 ± 0.36	20.71 ± 0.23	21.90 ± 0.33
DensePRC-NPTN (CMP=3)	10.95 ± 0.12	11.59 ± 0.23	20.95 ± 0.20	21.80 ± 0.42
DensePRC-NPTN (CMP=4)	10.61 ± 0.11	11.41 ± 0.12	20.80 ± 0.12	21.47 ± 0.16

Table 1: **Efficacy on CIFAR10:** Test error statistics on CIFAR10 with mean and standard deviation. ++ indicates AutoAugment testing. Each DenseNet and its corresponding PRC-NPTN variant has the same number of parameters. $|G| = 12$ for PRC-NPTN and growth rate was kept at 12 for DenseNet-Conv. (w/o Random) indicates no randomization in the connectomes constructed (as an ablation study). The speed and memory improvements are multiplicative improvement factors of our CUDA kernel implementation compared to baseline optimized PyTorch code.



Figure 3: Sample images from the ETH-80 database. The dataset contains 80 objects belonging to 8 different classes. Each object has images from different viewpoints on a hemisphere resulting in 3D pose and viewpoint variation for each object.

Method (Protocol 2)	Accuracy (%)	#parameters	Reduction	#filters (3×3)	Reduction
ConvNet [9]	84.49	1.4M		230	-
ConvNet B	95.58	110K	$1 \times$	3780	$1 \times$
ConvNet B (1×1)	94.33	115K	$0.95 \times$	3780	$1 \times$
NPTN-large B (3)	94.58	189K	$0.58 \times$	11268	$0.33 \times$
NPTN-small B (3)	94.70	120K	$0.91 \times$	4356	$0.86 \times$
PRC-NPTN B (8, 2)	94.34	97K	$1.13 \times$	708	$5.33 \times$
ConvNet C	95.44	116K	$1 \times$	12740	$1 \times$
ConvNet C (1×1)	94.40	138K	$0.84 \times$	12740	$1 \times$
PRC-NPTN C (8, 2)	94.68	64K	$1.81 \times$	1220	$10.44 \times$
PRC-NPTN C (8, 4)	95.63	39K	$2.97 \times$	1220	$10.44 \times$

Table 2: Test accuracy on ETH-80 Protocol 2. For NPTN is number in the bracket denotes $|G|$, for PRC-NPTN the numbers denote $|G|$ and CMP respectively.

almost 10X fewer 3×3 filters. These results show that PRC-NPTN are more effective in learning even out of the plane invariances from data itself without any change in architecture.

1.1 Proof of Lemma 2.1

Lemma 1.1. (*Invariance Property*) Assume a novel test input x and a filter w both fixed vectors $\in \mathbb{R}^d$. Further, let g denote a random variable representing unitary operators with some distribution. Finally, let $Y(x) = \langle x, gw \rangle$, with $Y(x) \sim U(a, b)$ i.e. a Uniform distribution between a and b . Then, we have

$$\text{Var}(\max Y(x)) \leq \text{Var}(Y(x)) = \text{Var}(\langle g^{-1}x, w \rangle)$$

Proof. Let X be the random variable representing the randomness in $\langle x, gw \rangle$ for fixed x, w and random g . We assume that $X \sim U(0, 1)$.

Considering a sample set $X_1, X_2, \dots, X_n \sim U(0, 1)$, then $X_{(n)} = \max_{1 \leq i \leq n} X_i$. Now,

$$P(X_{(n)} \leq x) = P(X_i \leq x, \forall i) \tag{2}$$

$$= P(X_i \leq x)^n \tag{3}$$

$$= x^n \tag{4}$$

Let the density of $X_{(n)}$ be denoted by $f_{X_{(n)}}(x)$, then

$$f_{X_{(n)}}(x) = \begin{cases} 0 & x \leq 0 \\ nx^{n-1} & 0 \leq x \leq 1 \\ 1 & x \geq 1 \end{cases}$$

Now,

$$\mathbb{E}[X_{(n)}] = \int_0^1 xnx^{n-1}dx = \frac{x^{n+1}}{n+1}n \Big|_0^1 = \frac{n}{n+1}$$

$$\mathbb{E}[X_{(n)}^2] = \int_0^1 x^2nx^{n-1}dx = \frac{x^{n+2}}{n+2}n \Big|_0^1 = \frac{n}{n+2}$$

Therefore,

$$\text{Var}(X_{(n)}) = \frac{n}{n+1} - \left(\frac{n}{n+1} \right)^2 = \frac{n}{(n+2)(n+1)^2}$$

```

1: class PRCN-NPTN:
2: def init(self, inch, outch, G, CMP, kernelSize, padding, stride):
3:     self.G = G
4:     self.maxpoolSize = CMP
5:     self.avgpoolSize = int((inch*self.G)/(self.maxpoolSize*outch))
6:     self.expansion = self.G*inch
7:     self.conv1 = nn.Conv2d(inch, self.G*inch, kernelSize=kernelSize, groups=inch, padding=padding,
        bias=False)
8:     self.transpool1 = nn.MaxPool3d((self.maxpoolSize, 1, 1))
9:     self.transpool2 = nn.AvgPool3d((self.avgpoolSize, 1, 1))
10:    self.index = torch.LongTensor(self.expansion).cuda()
11:    self.randomlist = list(range(self.expansion))
12:    random.shuffle(self.randomlist)
13:    for ii in range(self.expansion):
14:        self.index[ii] = self.randomlist[iii]
15:
16: def forward(self, x):
17:     out = self.conv1(x) #inch  $\rightarrow$  G*inch
18:     out = out[:,self.index,:,:] # randomization
19:     out = self.transpool1(out) # G*inch  $\rightarrow$  inch*G/maxpool
20:     out = self.transpool2(out) # inch*G/(maxpool*meanpool)  $\rightarrow$  outch
21:     return out

```

Figure 4: PRC-NPTN pseudo-code.

Since the variance of $U(0, 1)$ is $\frac{1}{12}$ i.e. $\text{Var}(X_i) = \frac{1}{12}$, and $\text{Var}(X_{(n)})$ is a decreasing function in n , along with the fact that $\text{Var}(X_{(n)})$ for $n = 1$ is $\frac{1}{12}$, we have

$$\text{Var}(X_{(n)}) \leq \text{Var}(X_i)$$

For general $U(a, b)$, it follows shortly after considering $Y_i = \frac{X_i - a}{b - a}$ and that $\text{Var}(Y_i) = \text{Var}(X_i)$. Finally, due to unitary g , $\langle x, gw \rangle = \langle g^{-1}x, w \rangle$. \square

References

- [1] François Chollet. Xception: Deep learning with depthwise separable convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1251–1258, 2017.
- [2] Ekin D Cubuk, Barret Zoph, Dandelion Mane, Vijay Vasudevan, and Quoc V Le. Autoaugment: Learning augmentation policies from data. *arXiv preprint arXiv:1805.09501*, 2018.
- [3] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.
- [4] Renata Khasanova and Pascal Frossard. Graph-based isometry invariant representation learning. In *International Conference on Machine Learning*, pages 1847–1856, 2017.

Rotation	0°	***	30°	***	60°	***
ConvNet (36)	0.70 \pm 0.03	-	0.92 \pm 0.03	-	1.32 \pm 0.07	-
ConvNet (36) FC	0.66 \pm 0.05	-	0.80 \pm 0.03	-	1.08 \pm 0.02	-
ConvNet (512)	0.65 \pm 0.04	-	0.80 \pm 0.02	-	1.14 \pm 0.03	-
NPTN (36,1)	0.68 \pm 0.04	-	0.93 \pm 0.01	-	1.35 \pm 0.05	-
NPTN (18,2)	0.66 \pm 0.02	-	0.87 \pm 0.04	-	1.18 \pm 0.02	-
NPTN (12,3)	0.68 \pm 0.06	-	0.84 \pm 0.02	-	1.19 \pm 0.01	-
NPTN (9,4)	0.64 \pm 0.01	-	0.88 \pm 0.05	-	1.21 \pm 0.03	-
PRCN (36,1)	0.62 \pm 0.08	0.62 \pm 0.06	0.84 \pm 0.01	0.83 \pm 0.03	1.17 \pm 0.05	1.19 \pm 0.02
PRCN (18,2)	0.61 \pm 0.02	0.57 \pm 0.02	0.68\pm0.02	0.73 \pm 0.02	0.93\pm0.04	0.99 \pm 0.04
PRCN (12,3)	0.58\pm0.03	0.62 \pm 0.04	0.72 \pm 0.02	0.74 \pm 0.02	0.95 \pm 0.01	1.04 \pm 0.04
PRCN (9,4)	0.63 \pm 0.02	0.62 \pm 0.04	0.75 \pm 0.03	0.77 \pm 0.02	0.99 \pm 0.03	1.05 \pm 0.03
Translations	0 pixels	***	4 pixels	***	8 pixels	***
ConvNet (36)	0.69 \pm 0.04	-	0.72 \pm 0.01	-	1.22 \pm 0.02	-
ConvNet (36) FC	0.60 \pm 0.02	-	0.64 \pm 0.01	-	0.88 \pm 0.05	-
ConvNet (512)	0.63 \pm 0.02	-	0.64 \pm 0.01	-	1.00 \pm 0.02	-
NPTN (36,1)	0.68 \pm 0.04	-	0.73 \pm 0.02	-	1.23 \pm 0.01	-
NPTN (18,2)	0.61 \pm 0.04	-	0.63 \pm 0.02	-	1.11 \pm 0.02	-
NPTN (12,3)	0.66 \pm 0.02	-	0.64 \pm 0.02	-	1.09 \pm 0.04	-
NPTN (9,4)	0.65 \pm 0.05	-	0.65 \pm 0.03	-	1.16 \pm 0.04	-
PRC-NPTN (36,1)	0.65 \pm 0.02	0.65 \pm 0.05	0.58 \pm 0.01	0.61 \pm 0.04	1.02 \pm 0.03	1.00 \pm 0.04
PRC-NPTN (18,2)	0.59\pm0.07	0.59 \pm 0.03	0.52\pm0.03	0.58 \pm 0.02	0.80\pm0.03	0.88 \pm 0.05
PRC-NPTN (12,3)	0.63 \pm 0.02	0.66 \pm 0.08	0.55 \pm 0.02	0.59 \pm 0.01	0.84 \pm 0.04	0.89 \pm 0.03
PRC-NPTN (9,4)	0.65 \pm 0.02	0.69 \pm 0.03	0.56 \pm 0.03	0.56 \pm 0.03	0.88 \pm 0.02	0.97 \pm 0.02

Table 3: **Individual Transformation Results:** Test errors on MNIST with progressively extreme transformations with a) **random rotations** and b) **random pixel shifts**. *** indicates ablation runs without any randomization *i.e.* without any random connectomes (applicable only to PRC-NPTNs). For PRC-NPTN and NPTN the brackets indicate the number of channels in the layer 1 and G. ConvNet FC denotes the addition of a 2-layered pooling 1×1 pooling network after every layer. Note that for this experiment, $CMP=|G|$. Permanent Random Connectomes help with achieving better generalization despite increased nuisance transformations.

- [5] Bastian Leibe and Bernt Schiele. Analyzing appearance and contour based methods for object categorization. In *Computer Vision and Pattern Recognition, 2003. Proceedings. 2003 IEEE Computer Society Conference on*, volume 2, pages II–409. IEEE, 2003.
- [6] Dipan K Pal and Marios Savvides. Non-parametric transformation networks for learning general invariances from data. *AAAI*, 2019.
- [7] Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1492–1500, 2017.
- [8] Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. Shufflenet: An extremely efficient convolutional neural network for mobile devices. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6848–6856, 2018.

Rot/Trans	0° 0	15° 2	30° 4	45° 6	60° 8	75° 10
ConvNet (36)	0.68 \pm 0.03	0.72 \pm 0.02	1.31 \pm 0.02	2.32 \pm 0.04	5.06 \pm 0.04	10.90 \pm 0.08
ConvNet (36) FC	0.64 \pm 0.03	0.66 \pm 0.01	0.95 \pm 0.04	1.50 \pm 0.02	3.42 \pm 0.03	8.14 \pm 0.11
ConvNet (512)	0.66 \pm 0.05	0.65 \pm 0.02	0.97 \pm 0.02	1.60 \pm 0.04	3.50 \pm 0.04	7.90 \pm 0.06
NPTN (36,1)	0.71 \pm 0.04	0.78 \pm 0.02	1.27 \pm 0.02	2.35 \pm 0.03	5.02 \pm 0.14	11.08 \pm 0.09
NPTN (18,2)	0.65 \pm 0.02	0.68 \pm 0.02	1.09 \pm 0.02	1.94 \pm 0.04	4.17 \pm 0.06	9.59 \pm 0.10
NPTN (12,3)	0.66 \pm 0.02	0.69 \pm 0.03	1.07 \pm 0.03	1.85 \pm 0.02	4.24 \pm 0.11	9.58 \pm 0.06
NPTN (9,4)	0.64 \pm 0.01	0.71 \pm 0.02	1.09 \pm 0.04	1.98 \pm 0.04	4.41 \pm 0.09	9.78 \pm 0.16
PRC-NPTN (36,1)	0.61 \pm 0.03	0.70 \pm 0.01	1.09 \pm 0.04	1.80 \pm 0.02	3.93 \pm 0.02	9.09 \pm 0.11
PRC-NPTN (18,2)	0.57\pm0.02	0.58\pm0.01	0.77\pm0.02	1.21\pm0.07	2.74\pm0.04	6.78\pm0.12
PRC-NPTN (12,3)	0.59 \pm 0.03	0.58 \pm 0.01	0.78 \pm 0.02	1.26 \pm 0.02	2.91 \pm 0.05	7.13 \pm 0.09
PRC-NPTN (9,4)	0.63 \pm 0.04	0.59 \pm 0.02	0.81 \pm 0.02	1.35 \pm 0.02	3.12 \pm 0.02	7.26 \pm 0.02

Table 4: **Simultaneous Transformation Results:** Test errors on MNIST with progressively extreme transformations with **random rotations** and **random pixel shifts simultaneously**. For PRC-NPTN and NPTN the brackets indicate the number of channels in the layer 1 and G . Note that for this experiment, $\text{CMP}=|G|$.