# An Attempt to Improve Generalization Performance in Reinforcement Learning with Deterministic World Models and WGANs

Tianshuai Yu[1,a) ]  Yoshimasa Tsuruoka[2]

**Abstract:** Significant progress has been made in the field of Reinforcement Learning (RL) in recent years. Using artificial neural networks, researchers are able to train agents that can play video games as well as or even better than human experts. However, it is common that the same environments are used in both training phases and testing phases, which results in agents' failure to generalize to other environments. In this work, we propose a method in which environment models and generative models are used to generate virtual game levels so as to improve the generalization performance of RL agents. We conducted experiments using a fully-observable deterministic discrete maze game in order to test the proposed method. However, the proposed method failed to converge during training because our environmnet model was not able to predict the future of unseen levels accurately.

## 1. Introduction

Deep Reinforcement Learning (RL) has been proved powerful not only in board games like Go [1] but also in video games like Atari [2]. However, generalization in RL is still a serious problem. When trained agents encounter an new level in a video game, they may fail catastrophically even the new level is similiar to the training levels. In short, it is common that RL agents overfit to the environments they experienced during the training phase.

Justesen et al. used hard-coded procedural level generators in GVGAI framework to create game levels for agent training [3]. However, there is no research using learned genrators and learned simulators to improve the genelization performance of RL agents.

In this work, we propose a new method to improve the generalization performance of RL agents. Generative models are used to generate new initial frames of the game and environment models are used to simulate the game frame by frame recursively from the generated initial frames.

The rest of this paper consists of five sections. Section 2 introduces related work. Section 3 describes the proposed method in detail. The conducted experiments are described in section 4. Results and discussions are shown in section 5. The last section is a conlclusion of this work.

[1] Department of Information and Communication Engineering, The University of Tokyo
[2] Department of Information and Communication Engineering, The School of Information Science and Technology, The University of Tokyo
[a)] tianshuai@logos.t.u-tokyo.ac.jp

## 2. Related work

### 2.1 Overfitting and Genelization in RL

Zhang et al. studied the overfitting problem in deep RL using a maze environment [4]. They found that the generalization performances of RL agents are highly dependent on the complexity of mazes and training set size. When trained and tested in mazes of the same complexity, agents trained with a larger set of training levels can generalize to unseen levels better.

Cobbe et al. studied the generalization in RL using a 2D platform game called CoinRun, which is designed by them as a benchmark for generalization in RL [5]. Their experimental results are similiar to Zhang et al.'s, which showed that more traning levels lead to better generalization performance. Cobbe et al. also tested several regularization techniques that are widely used in supervised learning. They found that dropout, L2 regularization, data augmentation, batch normalization, introducing stochasticity are also useful to the reduction of generalization gap in RL.

### 2.2 Model-Based RL

Our proposed method is inspired by model-based RL, especially by Kaiser et al.'s work in which they used model-based RL to train agents that can play Atari 2600 games [6]. They used convolution networks and deconvolution networks with input action embeddings to build their environment model. The environment model takes four stacked frames as well as the action selected by the agent as input and the model is able to predict the next frame and the reward. They collected observations from real environments to train the environment model and then trained the agent in the environment model. By using environment models, their agents were able to achieve the same level of gameplay as model-free algorithms with much fewer samples from real environments. One exciting result of the experiment is that in some games like `Pong`, `Freeway`, `Breakout`, their environment models were able to predict the future pixel-perfectly for up to 50 time-steps.

### 2.3 Generative Adversarial Networks (GANs)

Generative Adversarial Networks (GANs) were first introduced by Goodfellow et al. [7]. The training process of GANs can be regarded as a two-player mini-max game. The first player is the generator whose input is a noise vector and output is a generated image. The second player is the discriminator which takes an image as input and outputs the probability that the input image is in the training set. The discriminator is trained to minimize the probability of misjudgement while the generator is trained to maximize this probability. An ideal training process ends with the generator generating realistic images and the discriminator being not able to judge the input image more accurately than random guesses.

Wasserstein GANs (WGANs) are one variation of GAN introduced by Arjovsky et al. [8]. WGANs minimize the Wasserstein distance so that images generated by the generator have a distribution which is close to the real distribution. WGANs are reported to have more stable training process than standard GANs. WGANs also do not suffer from model collapse, which is an occasionally encountered problem when training original GANs.

WGANs can be used to generate not only pixel based images, but also data with more abstract structure. Volz et al. [9] used a one-hot expression encoded according to the type of tile at each position to express stages of *Super Mario Bros* and trained WGANs with this expression. Their results showed that the trained WGANs were able to generate playable *Super Mario Bros* stages. It is also amazing that they only used 173 pieces training data to train the WGANs, which showed that WGANs are capable of learning from a relatively small data set.

### 2.4 Deep Q-Network (DQN)

In RL, the value of taking action $a$ at state $s$ under certain policy $\pi$ is given by the following equation:

$$Q_\pi(s,a) = \mathbb{E}[R_1 + \gamma R_2 + ...|S_0 = s, A_0 = a, \pi]$$

, where $\gamma$ is the discount factor for delayed rewards. The optimal Q-value is given by

$$Q_*(s,a) = \max_\pi Q_\pi(s,a)$$

. The optimal policy can be obtained easily by taking the action with the highest Q-value at each time-step.

The optimal action value function can be approximated by Q-Learning [10]. The Q-function is usually approximated by a function with parameters $\boldsymbol{\theta}$. When the Q-funtion is denotated by $Q(s,a;\boldsymbol{\theta_t})$ and Stochastic Gradient Descent (SGD) is used for parameter updates, the parameters are updated as the following equation:

$$\boldsymbol{\theta_{t+1}} = \boldsymbol{\theta_t} + \alpha(Y_t^Q - Q(S_t, A_t; \boldsymbol{\theta_t}))\nabla_{\boldsymbol{\theta_t}} Q(S_t, A_t; \boldsymbol{\theta_t})$$

, where $\alpha$ is the step size of parameter updates and the target $Y_t^Q$ is defined by the following equation:

$$Y_t^Q = R_{t+1} + \gamma \max_a Q(S_{t+1}, a; \boldsymbol{\theta_t})$$

.

Deep Q-Network (DQN) is introduced by Mnih et al. [2]. DQNs take states as input and output the value of each action. Experience replay and target networks are the two most characteristic techniques of DQNs. Experience replay means that the transitions experienced by the agent are stored in a replay buffer, and they are sampled from the replay buffer to be used to update the parameters of the network during the training phase. Target networks are the networks only used to calculate the targets for parameter updates. In contrast, the networks used for agent's exploration are called online networks. When parameter updates are performed, only the parameters $\boldsymbol{\theta}$ of online networks are updated. The parameters of target networks are copied from the online networks every $\tau$ time-steps. When the parameters of target networks are denoted by $\boldsymbol{\theta^-}$, the target of online networks' parameter updates is defined by the following equation.

$$Y_t^{DQN} = R_{t+1} + \gamma \max_a Q(S_{t+1}, a; \boldsymbol{\theta_t^-})$$

Mnih et al.'s work showed that DQNs are able to play certain Atari 2600 games at human-level [2]. In our work, DQNs are used as the method of agent's policy learning.

## 3. Proposed Method

According to Zhang et al.'s and Cobbe et al.'s work, agents trained with a larger set of training levels can generalize to unseen levels better [4] [5]. Based on this fact, we propose a new method to improve the genelization performance of RL agents.

Our method aims to create virtual levels using environment models and generative models and then train the agent in both real levels and created levels. Ideally, the agent can experience more levels in the training phase so that higher generalization performance is expected.

There are generally four steps in our method:

(1) Use the given training level set to train an environment model that can predict the next frame of the game, the reward and whether the episode is done after this frame.
(2) Extract initial frames from given training levels and train a generator with these initial frames.
(3) Generate new initial frames of the game and use these generated frames as the input of the environment model, so that we can create vitural levels by predicting upcoming frames, rewards recursively.
(4) Train the agent using both real levels and created virtual levels.

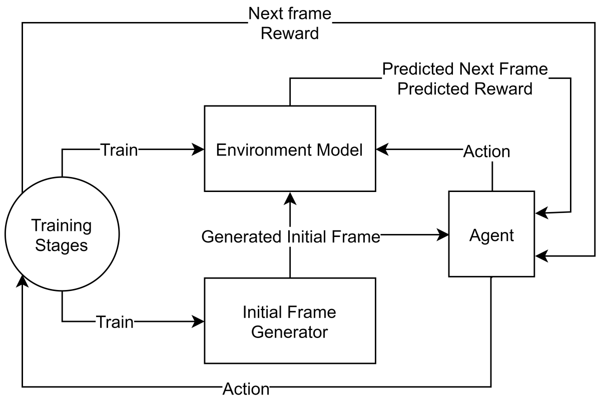


Fig. 1: Proposed method

The procedure of our method is shown in Figure 1.

However there are some constraints to this method. Firstly, the size of training level set cannot be too small. It is difficult to train a useful generator using an extremely small data set. Secondly, this method can only be used in certain games. The game need to be fully observable; otherwise it may be difficult for the environment model to predict the next frame only using previous frames and input actions. Finally, the game should have short episodes. Because the virtual levels are recursively simulated by the environment model, errors of the environment model will accumulate as the episode becomes longer. To avoid catastrophically wrong predictions, each episode of the game need to be short.

## 4. Experiments

### 4.1 Environment

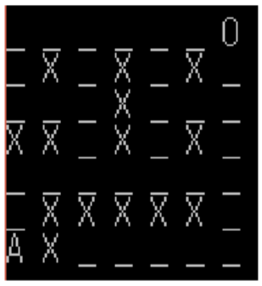
We used a fully-observable deterministic discrete maze game as the environment of our experiments. The mazes are square-sized and they are generated using Kruskal's algorithm [11]. The environment can generate different mazes according to given random seeds. The agent is always spawned at the left-bottom corner and the goal is always at the right-top corner of the maze. There are four available actions, up, down, left and right for the agent. The agent can move one block toward the selected direction at each time-step. However, if there is a wall block at the destination block, the agent won't be moved.

An observation of the environment is encoded as an image consists of 4 channels. Each channel is the binary expression of the corresponding type of blocks. For example, if one block is a wall block, there will be a 1 at that position in wall blocks' channel and there will be zeros at that position in other channels. An example of observations from a 7 × 7 maze is shown in Figure 2.

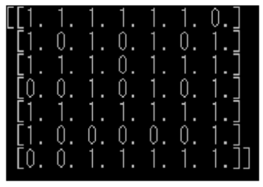
The agent receives a reward of −1 at each time-step except the step when the agent reaches the goal; a reward of 0 is given to the agent instead. In our experiments, we used 15 × 15 mazes and set the time limit to 200 time-steps.

### 4.2 DQN

We used DQN for agents' policy learning. We followed Mnih et al.'s network structure consisting of convolution layers and fully-connected layers [2]. Detailed network structure is illustrated in Figure 3.

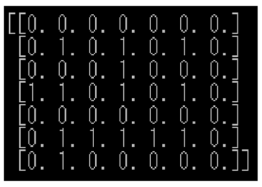


(a) Screenshot



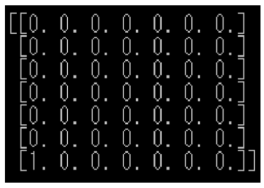
```
[[1. 1. 1. 1. 1. 1. 0.]
 [1. 0. 1. 0. 1. 0. 1.]
 [1. 1. 1. 0. 1. 1. 1.]
 [0. 0. 1. 0. 1. 0. 1.]
 [1. 1. 1. 1. 1. 1. 1.]
 [1. 0. 0. 0. 0. 0. 1.]
 [0. 0. 1. 1. 1. 1. 1.]]
```

(b) Empty blocks' channel



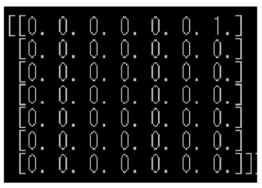
```
[[0. 0. 0. 0. 0. 0. 0.]
 [0. 1. 0. 1. 0. 1. 0.]
 [0. 0. 0. 1. 0. 0. 0.]
 [1. 1. 0. 1. 0. 1. 0.]
 [0. 0. 0. 0. 0. 0. 0.]
 [0. 1. 1. 1. 1. 1. 0.]
 [0. 1. 0. 0. 0. 0. 0.]]
```

(c) Wall blocks' channel



```
[[0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0.]
 [1. 0. 0. 0. 0. 0. 0.]]
```

(d) Agent block's channel



```
[[0. 0. 0. 0. 0. 0. 1.]
 [0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0.]]]
```

(e) Goal block's channel

Fig. 2: An example of observations: wall blocks are notated as X, empty blocks are notated as _, agent block is notated as A, goal block is notated as 0 in the screenshot

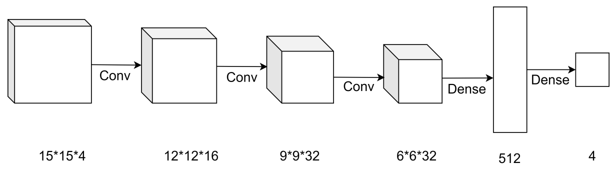


Fig. 3: DQNs' network structure

### 4.3 Environment Model

We followed Kaiser et al.'s deterministic world model to build our environment model [6]. Our environment model is mainly constructed with convolution layers and deconvolution layers. The input actions are embedded with fully-connected layers and multiplied to the input of each deconvolution layer. There are also skip connections between convolution layers and deconvolution layers. Batch normalization is applied between every two adjacent layers. The network structure of our environment model is illustrated in Figure 4.

Our environment model consists of three modules: a next frame predictor, a reward predictor and a done predictor. They share the same network structure except the output part. However, the parameters of each module are not shared. The inputs of these three predictors are the current game frame and the action selected by the agent. The

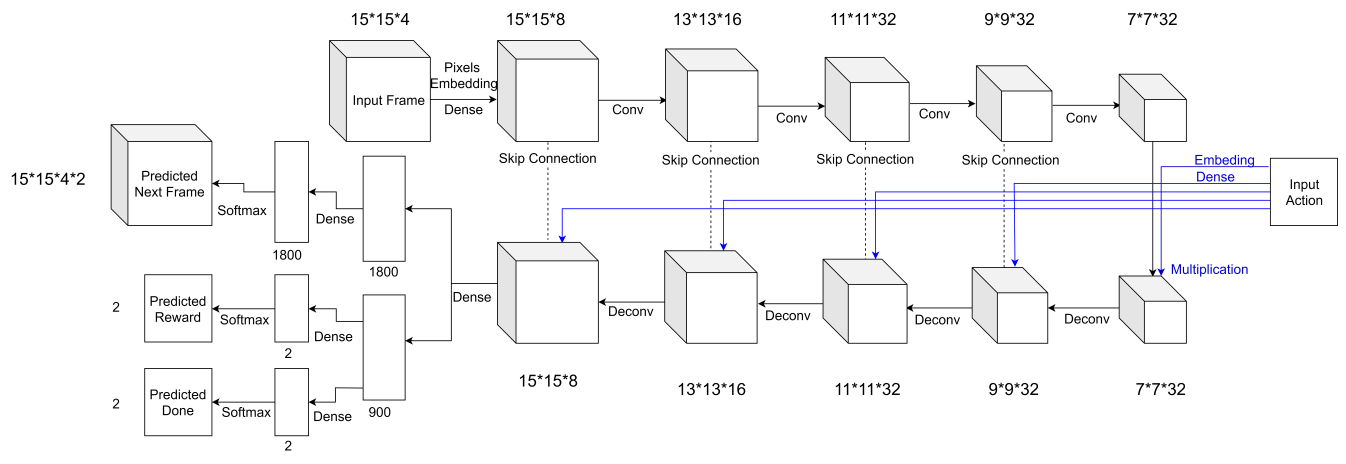


Fig. 4: Environment model's network structure

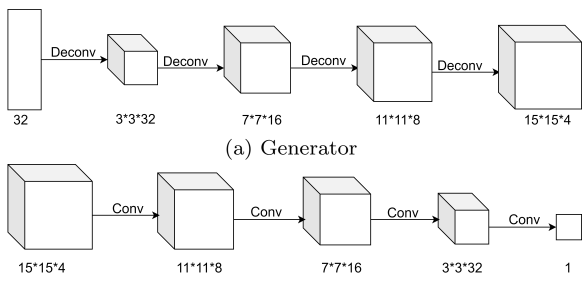


Fig. 5: WGAN's network structure

next frame predictor outputs the probabilities of each block in each channel being 1 or 0. The reward predictor predicts the probabilities of the reward being $-1$ or 0. The done predictor predicted the probability that the episode ends at this time-step.

However, it is impossible to predict the end of an episode caused by out of time. Our done predictor only handle the end of episodes caused by the agent reaching the goal. When training the agent with the environment model, the number of past time-step is counted and the episode is ended manually when it reaches the time limit.

### 4.4 Generator

We used WGANs to train our initial frame generators. The generator and the critic were implemented with deep deconvolution networks and deep convolution networks. The generator used ReLu activation for each layer and Tanh for the output. The critic used Leaky ReLU activation in all layers. Batch normalization is applied after each layer in both the generator and the critic. The network structure of our WGAN is illustrated in Figure 5.

### 4.5 Training

The environment model was trained by experience replay which is similar to the training of DQNs in 100 given levels. The parameters of the environment model were updated every four time-steps and it was trained totally for 3M time-steps with a learning rate of 0.001. In order to make sure that small objects such as agent and goal can also be properly predicted, each value in the predicted frame was clipped before the differentiation was performed. In our case, if the confidence that a value was correctly predicted is over 97%, the value would be clipped and the gradient from this value would not be used in parameter updates.

Table 1: Results of environment model

| | block-level accuracy | frame-level accuracy |
|---|---|---|
| next frame (train) | 100% | 100% |
| next frame (test) | 89.57% | 0% |
| reward (train) | — | 100% |
| reward (test) | — | 100% |
| done (train) | — | 100% |
| done (test) | — | 100% |

The WGAN was trained with 100 given initial frames for 30K epochs. The learning rate was set to 0.00005 and the clip values of the critic's parameters were set to ± 0.01. The parameters of the critic network were updated five times for each update of the generator.

The baseline DQN was trained in 100 levels for 20M time-steps. A level was selected randomly at the beginning of an episode. The parameters were updated every four time-steps. $\epsilon$-greedy was used for exploration. $\epsilon$ decreased from 1 to 0.1 linearly in 10M time-steps. The size of replay buffer was set to 50K and the learning rate was set to 0.00025.

The proposed method was trained with 100 real levels and 100 unique generated vitural levels. At the beginning of an episode one level is randomly chosen from all the 200 levels. All other settings, including network structure and hyperparameters, are the same with the baseline DQN.

## 5. Results & Discussions

### 5.1 Environment Model

The environment model was able to predict next frame, reward and whether the episode would be done perfectly in training levels. However, it failed to predict the future in unseen testing levels. The detailed results are shown in Table 1.

### 5.2 WGAN

The genrator of WGAN was able to generate patterns of

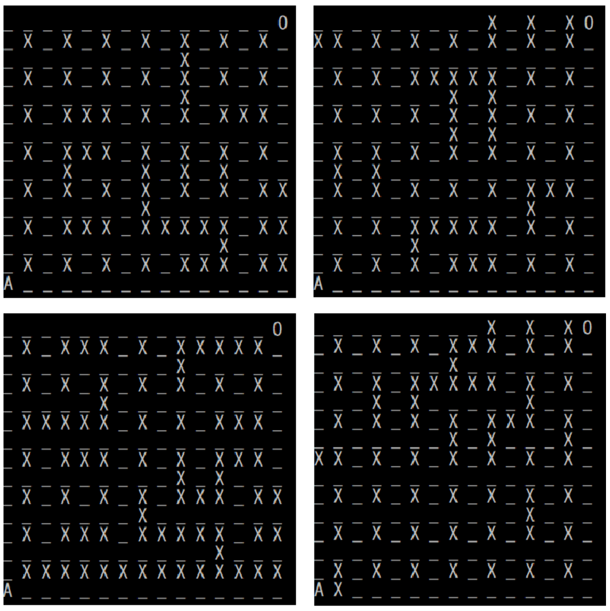


Fig. 6: Samples of generated initial frames

Table 2: Percentage of levels solved by DQN and proposed method

| | train | test |
|---|---|---|
| DQN (baseline) | 100% | 48.69% |
| proposed method | 0% | 0% |

wall blocks and empty blocks which share the same characteristics with real mazes. However, the generator always failed to correctly generate the channel of agent and the channel of goal. So that we only used generated blocks' channel and emptys' channel, the channel of agent and the channel of goal were set manually.

Some initial frames generated by the generator of WGAN are shown in Figure 6 (with manual replacement of agent's channel and goal's channel).

### 5.3 Proposed Method

The proposed method failed to converge during training.

We tested the agents trained by the baseline DQN and the proposed method in both training level set and testing level set which consists of 10,000 mazes generated with unseen random seeds. The solved rates are shown in table 2.

### 5.4 Discussion

The main reason for the failure of the proposed method is the low accuracy of the environment model in unseen levels. Because the environment couldn't predict the future accurately, the replay data gathered from vitural levels was likely to be meaningless. As about 50% of the replay buffer was filled with meaningless data, the stability of policy learning was hurt, which made the policy fail to converge.

As environment models are trained by supervised learning, some widely used techniques to prevent overfitting in supervised learning may be helpful when aiming to train an environment model that can generalize to unseen levels. If a environment model that can generalize well to new levels is available, the proposed method should work much better.

## 6. Conclusion & Future Work

We proposed a method to improve the genelization performance of RL agents. Environment models and generative models are used to generate virtual game levels in the proposed method. However, the proposed method failed because our environment model was not able to predict the future of unseen levels.

In the future, we will try to build an environment model that can predict the next frame with a high accuracy and test the proposed method again with the new environment model.

### References

[1] Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M. et al.: Mastering the game of Go with deep neural networks and tree search, *nature*, Vol. 529, No. 7587, p. 484 (2016).

[2] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G. et al.: Human-level control through deep reinforcement learning, *Nature*, Vol. 518, No. 7540, p. 529 (2015).

[3] Justesen, N., Torrado, R. R., Bontrager, P., Khalifa, A., Togelius, J. and Risi, S.: Illuminating generalization in deep reinforcement learning through procedural level generation, *arXiv preprint arXiv:1806.10729* (2018).

[4] Zhang, C., Vinyals, O., Munos, R. and Bengio, S.: A study on overfitting in deep reinforcement learning, *arXiv preprint arXiv:1804.06893* (2018).

[5] Cobbe, K., Klimov, O., Hesse, C., Kim, T. and Schulman, J.: Quantifying Generalization in Reinforcement Learning, *International Conference on Machine Learning*, pp. 1282–1289 (2019).

[6] Kaiser, L., Babaeizadeh, M., Milos, P., Osinski, B., Campbell, R. H., Czechowski, K., Erhan, D., Finn, C., Kozakowski, P., Levine, S. et al.: Model-based reinforcement learning for Atari, *arXiv preprint arXiv:1903.00374* (2019).

[7] Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A. and Bengio, Y.: Generative adversarial nets, *Advances in neural information processing systems*, pp. 2672–2680 (2014).

[8] Arjovsky, M., Chintala, S. and Bottou, L.: Wasserstein gan, *arXiv preprint arXiv:1701.07875* (2017).

[9] Volz, V., Schrum, J., Liu, J., Lucas, S. M., Smith, A. and Risi, S.: Evolving mario levels in the latent space of a deep convolutional generative adversarial network, *Proceedings of the Genetic and Evolutionary Computation Conference*, ACM, pp. 221–228 (2018).

[10] Watkins, C. J. C. H.: Learning from delayed rewards, PhD Thesis, King's College, Cambridge (1989).

[11] Kruskal, J. B.: On the shortest spanning subtree of a graph and the traveling salesman problem, *Proceedings of the American Mathematical society*, Vol. 7, No. 1, pp. 48–50 (1956).