



Optimal Terminal Interconnection Reconstruction along with Terminal Transition in Randomly Divided Planes

Jiwon Youn and Byungyeon Hwang* , *Member, KIICE*

School of Computer Science and Information Engineering, The Catholic University of Korea, Bucheon 14662, Korea

Abstract

This paper proposes an efficient method of reconstructing interconnections when the terminals of each plane change in real-time situations where randomly divided planes are interconnected. To connect all terminals when the terminals of each plane are changed, we usually reconstruct the interconnections between all terminals. This ensures a minimum connection length, but it takes considerable time to reconstruct the interconnection for the entire terminal. This paper proposes a solution to obtain an optimal tree close to the minimum spanning tree (MST) in a short time. The construction of interconnections has been used in various design-related areas, from networks to architecture. One of these areas is an ad hoc network that only consists of mobile hosts and communicates with each other without a fixed wired network. Each host of an ad hoc network may appear or disappear frequently. Therefore, the heuristic proposed in this paper may expect various cost savings through faster interconnection reconstruction using the given information in situations where the connection target is changing.

Index Terms: Ad-hoc network, Minimum spanning Tree, Tinkered Tree, Terminal interconnection problem

I. INTRODUCTION

The construction of a maximum interconnection of elements distributed within a given space is a problem abstracted in various industrial fields, from networks to architecture [1-3]. In this interconnection construction, connecting all terminals using the minimum length can be obtained using the minimum cost spanning tree (MST) algorithm [4-5]. However, using the MST algorithm requires considerable time [6-7]. This paper defines a tinkered tree as an optimal tree that connects all terminals faster than the MST algorithm.

As Internet has developed, some networks have undergone dynamic changes. A network with connected terminals can be inserted or deleted frequently, such as an ad hoc network. Ad-hoc networks are aggregates of wireless mobile nodes without an existing network infrastructure or centralized management. Building an MST whenever terminals are

changed takes a lot of time. Therefore, a heuristic that responds to changes in terminals must apply a tinkered tree to such a dynamic situation [8].

This paper considers a situation where the terminal status of divided planes changes in real-time when each plane forms a tinkered tree. In this situation, a heuristic was proposed to connect all terminals faster than the MST algorithm. This heuristic can be used in cases responding quickly to frequent network changes when networks operated regionally on a large area in the real world are connected. So, we can expect an efficient CDS (connected dominating set) composition of an Ad-hoc wireless network consisting of only mobile hosts without constructing a fixed wired network [9-12].

In the following section, we briefly explain the outline of the tinkered tree. In Section 3, we explain the heuristic proposed in this paper, and in Section 4, we prove the useful


Received 20 March 2022, Revised 13 May 2022, Accepted 20 May 2022

*Corresponding Author Byungyeon Hwang (E-mail: byhwang@catholic.ac.kr, Tel: +82-2-2164-4363)

School of Computer Science and Information Engineering, The Catholic University of Korea, 43 Bucheon 14662, Korea

Open Access <https://doi.org/10.56977/jicce.2022.20.3.160>

print ISSN: 2234-8255 online ISSN: 2234-8883

 This is an Open Access article distributed under the terms of the Creative Commons Attribution Non-Commercial License (<http://creativecommons.org/licenses/by-nc/3.0/>) which permits unrestricted non-commercial use, distribution, and reproduction in any medium, provided the original work is properly cited.

Copyright © The Korea Institute of Information and Communication Engineering

ness of the proposed algorithm through experiments. In Section 5, we present the conclusions of this paper.

II. TINKERED TREE

Suppose that an interconnection expansion occurs between planes and that the divided plane has a set of terminals connected by MST. By applying the MST algorithm to all terminals on a plane, all planes can be connected using minimum lengths. However, this process requires a considerable amount of time. A tinkered tree is an optimal tree faster than the MST algorithm and has a similar shape to the MST in this situation. The tinkered tree uses the portal concept to build an optimal tree similar to MST without creating an MST for all terminals. We use a portal to find the closest pair to connect the partition to an adjacent partition. After calculating the distance between all partitions, they were connected using the distance.

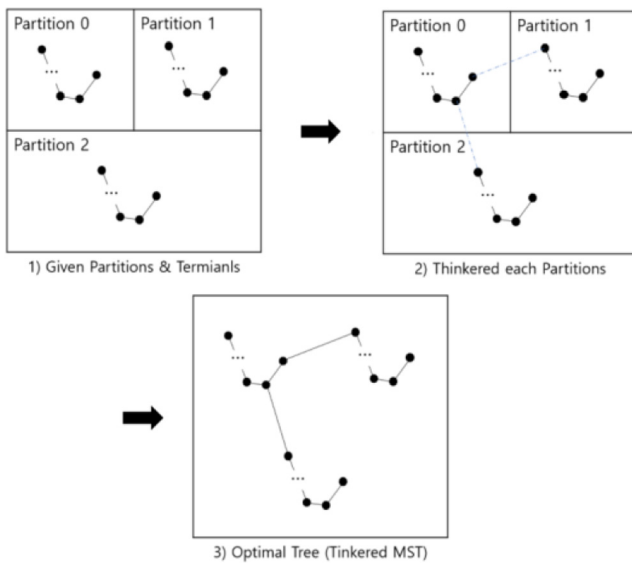


Fig. 1. Tinkered Tree Heuristic.

Figure 1 shows the process of tinkered tree heuristic with three partitions. By comparing the distance of closet pairs and creating a tinkered tree that connects the three partitions, the interconnection is effectively constructed, maintaining existing terminal subsets.

In this section, the tinkered tree is applied to a situation involving the interconnection of static terminal sets. This paper further expands the concept of an existing tinkered tree in response to a situation in which partitions are connected, and the terminals in the partitions are changed.

III. HEURISTIC OF THIS PROBLEM

A. Improvement of Tinkered Tree

When terminals change frequently, it takes considerable time to reconstruct a new interconnection for all terminals whenever a change occurs. Therefore, to quickly respond to changes in terminals, it is necessary to rebuild the interconnection by considering only the terminals related to the change while maintaining the current interconnection status as much as possible. To achieve this, the current interconnection information must be stored.

1) Information on Adjacent Terminals

Each terminal in the partition is connected to one or more other terminals. When there is a change in the terminal in the plane, if we want to consider only the changed terminal, we need information on the terminals connected with this changed terminal. When terminals are inserted or deleted, this information makes quick operation possible because only the connection status of the terminals related to the changed terminal is updated. Figure 2 shows a class diagram of the added terminal information.

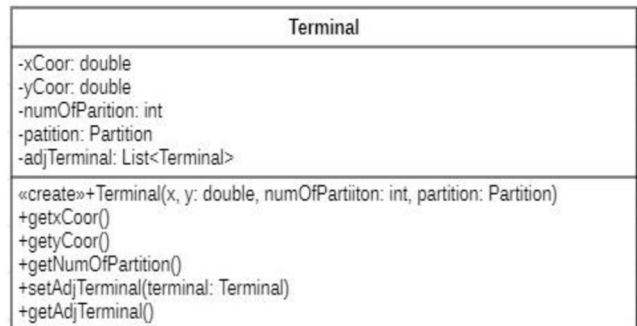


Fig. 2. Class Diagram of Terminal.

2) Information on tinkered tree

The critical information in the interconnected tinkered tree is the terminal responsible for connecting adjacent partitions. In this paper, we refer to this terminal as a tinkered terminal. Each partition stores the information of the Tinkered terminal, the terminals paired with the Tinkered terminal, and the portal used for that connection. If the tinkered terminal is changed in the insertion and deletion operations, we use this information to replace the tinkered terminal to make it close to the MST. In this paper, this information is called TinkeredInfo. Fig. 3 shows the class diagram of TinkeredInfo.

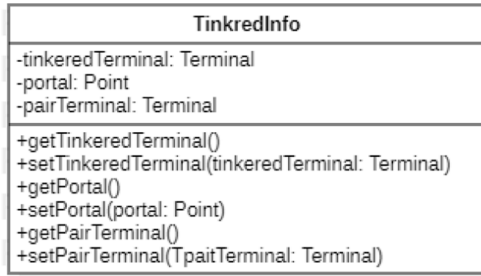


Fig. 3. Class Diagram of TinkredInfo.

B. Operation Algorithm

1) Insertion Operation

When there is an interconnected tinkered tree, and a new terminal is inserted, the algorithm in this paper works according to the logic shown in Fig. 4.

When a new terminal is inserted, the distance between the tinkered terminal and portal with the distance between the tinkered terminal and portal is compared. If the inserted terminal is closer to the portal than the tinkered terminal, the tinkered terminal is replaced by the inserted terminal. This is necessary to keep the tree as close as possible to the MST, even if the operation continues. In this manner, we compared the tinkered terminals with all adjacent partitions. The inserted terminal is then connected to the nearest terminal in the partition so that it can be connected to all terminals. When a terminal is inserted, the change occurs only in the partition whose terminal is inserted following this logic.

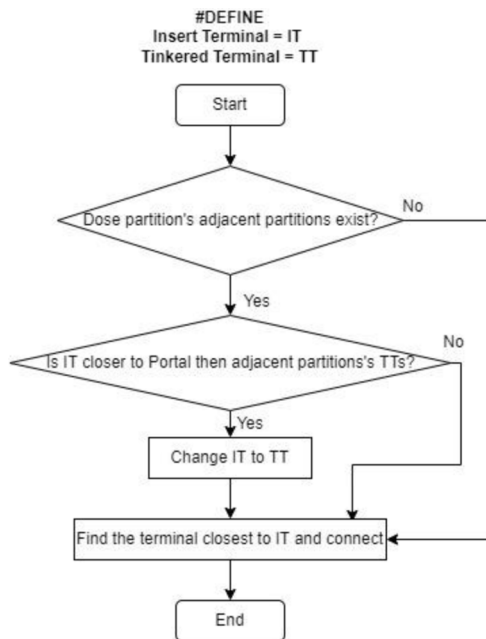


Fig. 4. Insertion Operation Algorithm.

Therefore, this can save considerable time compared with reconstructing the MST for the entire terminal.

2) Deletion Operation

When there is an interconnected tinkered tree, and in the case of an existing terminal being deleted, the algorithm in this paper works with the logic shown in Fig. 5.

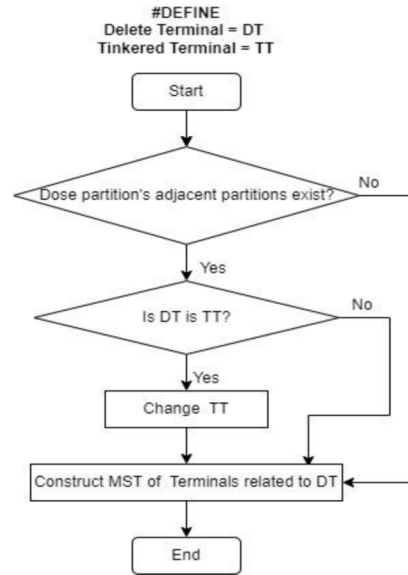


Fig. 5. Deletion Operation Algorithm.

The connection to the adjacent partition is lost if one terminal is deleted from the tinkered tree and the terminal is tinkered. Therefore, it is necessary to replace the tinkered terminal with another terminal when it is deleted. At this time, the terminal to be replaced is the second terminal, a nearby portal used for the connection. After updating the tinkered terminal, the connection between the deleted and connected terminals proceeds. As shown in Fig. 6, if the terminal is at the end of the connection, the connection is maintained even though the terminal is deleted.

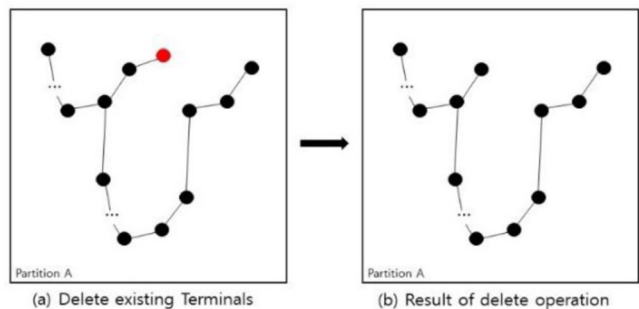


Fig. 6. Deletion of End Terminal.

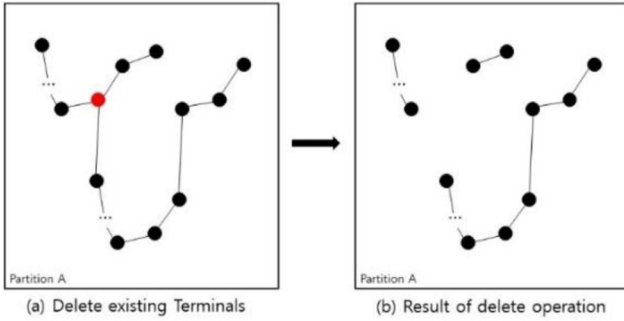


Fig. 7. Deletion Middle Terminal.

However, as shown in Fig. 7, if the terminal is in the middle of the connection, the tinkered tree is disconnected if that terminal is deleted. Therefore, when the middle terminal is deleted, it is necessary to build an MST for the terminals connected to the deleted terminal to maintain the connection.

When an existing terminal is deleted, the change occurs only in the partition whose terminal is deleted following this logic. Following this logic, we can save considerable time constructing an MST for the entire terminal after changing.

IV. EXPERIMENTS

A. Benchmark Model

We require a benchmark model to evaluate the performance of the proposed heuristic. For cases of interconnection where the number of components to be interconnected is fixed, and the cost of the length should be minimized, it is natural to apply an MST algorithm to obtain the optimal solution. Among the algorithms for interconnection problems, the Prim algorithm for MSTs is the most suitable benchmark because it is directly applicable to our proposed problem. Therefore, in this paper, we select the Prime algorithm for our benchmark model and compare our heuristic with the benchmark by constructing the MST whenever the terminals change. The comparison figures are the percentage of runtime and length, as shown in Eqs. (1) and (2).

$$\text{Result of Time} = \frac{\text{Time of Tinkered Tree}}{\text{Time of Benchmark model}} * 100 \quad (1)$$

$$\text{Result of Length} = \frac{\text{Length of Tinkered Tree}}{\text{Length of Benchmark model}} * 100 \quad (2)$$

B. Create Instance

We created an instance that is a set of operations required in this paper. The set of operations was randomly selected from the three operations: insertion, deletion, and movement.

Each operation has the following constraints.

Constraint 1. The operation involves the insertion of a new terminal, not an existing one. This implies that an existing terminal cannot be inserted.

Constraint 2. The deletion operation targets existing terminals. This is because a terminal that does not exist cannot be removed.

Constraint 3. Movement operations target existing terminals. Terminals that do not exist cannot be moved.

C. Environment for Experiments

The experiment environment was as follows: Windows 11 Pro 64-bit OS, 8.00 GB RAM, and CPU Intel(R) Core (TM) i-51135G7 2.40 GHz. The algorithms designed herein were implemented using Java and the Eclipse IDE.

The difference in the heuristic performance is related to the number of terminals and operations. Therefore, we needed to find the optimal values of these variables to be applied in the final experiment. In the experiment, for each variable, the results are the ratio of the time and length compared to the benchmark model. We then select the optimal values among them.

D. Choosing Optimal Variable

1) Terminal Experiment

To determine the optimal number of terminals for the final experiment, we experimented while changing the number of terminals from 1,000 to 100,000. The experimental results were the averages of the time and length using 10 different input files for each case. Fig. 8 shows a graph of time and length according to the number of terminals of the heuristic proposed in this paper. Each value is compared with the values presented in Equations 1 and 2 of Section IV.

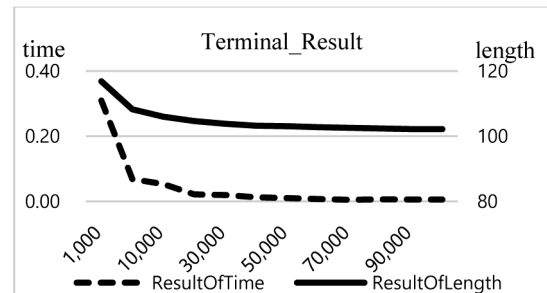


Fig. 8. Results according to the number of Terminals.

As shown in Fig. 8, as the number of terminals increases, the time and length decrease. From more than 50,000 terminals, the gap of decrease in the time and length value is smaller, and the value of the time and length gradually converges. Therefore, we fixed the number of terminals at

50,000 in the final experiment.

2) Operation Experiment

To determine the optimal number of operations for the final experiment, we experimented while changing the number from 50 to 5,000. Similar to the experiment for determining the number of terminals, the experimental results were the average of the time and length using 10 different input files for each case. Figure 9 shows a graph of time and length according to the number of operations of the heuristic proposed in this paper. Each value is compared using the equation in section IV.

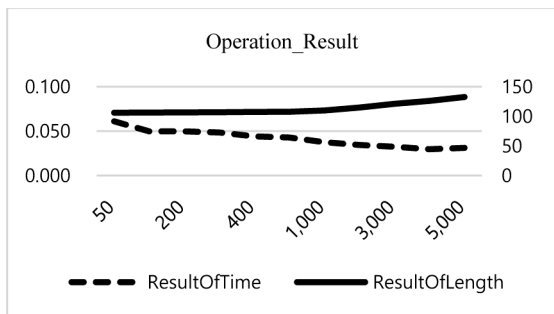


Fig. 9. Results according to the number of Operations.

As shown in Fig. 9, the time required decreased as the number of operations increased. However, the length decreased to 500 and then gradually increased. As the number of operations increases, the length increases because the shape of the tinkered tree gradually deviates from that of the MST. Therefore, we fixed the number of operations at 500 in the final experiment.

E. Performance Comparison

The optimal values for the number of terminals and operations were determined in the previous experiment. In the final experiment, we applied these optimal values and evaluated the performance of the heuristic compared to that of the benchmark model. The experiment used 50,000 terminals, 110 partitions, 50 portals, and 500 operations. The experi-

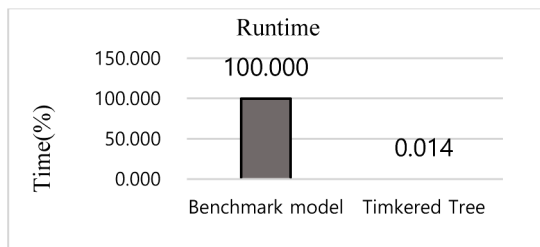


Fig. 10. Comparison of Runtime.

mental results are the averages of the time and length from ten different input files. Each value is compared using the equation in section IV. Figure 10 shows a bar graph of the runtime comparing the heuristic model with the benchmark model.

As shown in Fig. 10, the heuristic proposed in this paper shows 99.986% better runtime performance than the benchmark model. Following Table is the average value runtime of the heuristic and benchmark models.

Table 1. Standard runtime of Fig. 10

	Benchmark Model	Tinkered Tree
Runtime (sec)	8108.492	1.115

Figure 11 shows a bar graph of the length used to compare the heuristic with the benchmark model.

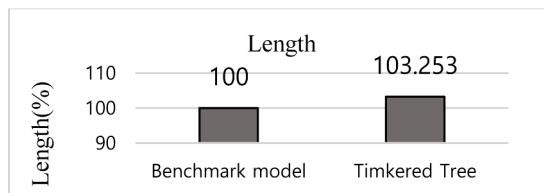


Fig. 11. Comparison of Length.

As shown in Fig. 11, when connecting all terminals, the heuristic proposed in this paper uses a length of approximately 3.3% more than that of the benchmark model. The following Table shows the average values of the used length of the heuristic and benchmark models.

Table 2. Standard length of Fig. 11

	Benchmark Model	Tinkered Tree
Length	13426039.546	13861466.665

The heuristic proposed in this paper improved the runtime by 99.986%, although it used a length of 3.3% more than that of the benchmark model. This proves that the heuristic proposed in this paper works effectively in terms of the runtime.

V. CONCLUSIONS

Given the MSTs of the randomly divided partition on a two-dimensional plane, we defined the tinkered tree, a heuristic that quickly constructs the interconnection using the given information without reconstructing the interconnection for the entire terminal. In addition, we expanded the concept

of the tinkered tree in a way that responds to the situation where terminals change in real-time on the interconnected plane. By comparing the tinkered tree with the benchmark model, we demonstrate that it can achieve more efficient performance. Although the tinkered tree does not ensure the minimum length, as in an MST algorithm, we can expect an effect in situations where a quick interconnection for fluid or flexibility networks is needed by constructing the interconnections within a short time.

Our future research intends to propose a plan to expand the problems in various fields, such as situations in which terminals in a plane have a specific distribution and situations in which the divided plane itself changes so that they can be applied to various fields.

ACKNOWLEDGMENTS

This work was supported by the Catholic University of Korea Research Fund (2021).

REFERENCES

- [1] K. Zhou and J. Chen, "Simulation DNA algorithm of set covering problem," *Applied Mathematics & Information Sciences*, vol. 8, no. 1, pp. 139-144, Jan. 2014. DOI: 10.12785/amis/080117.
- [2] P. K. Tripathy, R. K. Dash, and C. R. Tripathy, "A genetic algorithm based approach for topological optimization of interconnection networks," in *Procedia Technology*, Rourkela, India, vol. 6, pp. 196-205, 2012. DOI: 10.1016/j.protcy.2012.10.024.
- [3] J. Kim, J. Oh, M. Kim, Y. Kim, J. Lee, S. Han, and B. Hwang, "Maximum node interconnection by a given sum of euclidean edge lengths," *Journal of Information and Communication Convergence Engineering*, vol. 17, no. 4, pp. 246-254, 2019. DOI: 10.6109/jicce.2019.17.4.246
- [4] D. Hu, P. Dai, K. Zhou, and S. Ge, "Improved particle swarm optimization for minimum spanning tree of length constraint problem," in *8th International Conference on Intelligent Computation Technology and Automation*, Nanchang, China, pp. 474-477, 2015. DOI: 10.1109/icicta.2015.124.
- [5] G. Hong-mei, X. Chao, and Y. Ben-cheng, "Design and analysis of minimum spanning tree in euclidean plane," in *International Conference on Computational and Information Sciences*, Shiyang, China, pp. 976-979, 2013. DOI: 10.1109/iccis.2013.262.
- [6] V. Geetha, S. Aithal, and K. C. Sekaran, "Effect of mobility over performance of the ad hoc networks," in *International Symposium on Ad Hoc and Ubiquitous Computing*, Mangalore, India, pp. 138-141, 2006. DOI: 10.1109/isahuc.2006.4290661.
- [7] P. Flocchini, T. M. Enriquez, L. Pagli, G. Prencipe, and N. Santoro, "Distributed minimum spanning tree maintenance for transient node failures," *IEEE Transactions on Computers*, vol. 61, no. 3, pp. 408-414, Nov. 2012. DOI: <https://doi.org/10.1109/tc.2010.228>.
- [8] R. Ghoshal and S. Sundar, "Two approaches for the min-degree constrained minimum spanning tree problem," *Applied Soft Computing*, vol. 111, 107715, Nov. 2021. DOI: 10.1016/j.asoc.2021.107715.
- [9] Perkins, "Ad hoc networking in the IETF," in *IEEE International Workshop on Broadband Convergence Networks*, Vancouver: BC, Canada, pp. 1-35, 2006. DOI: 10.1109/bcn.2006.1662294.
- [10] S. Ren, P. Yi, D. Hong, Y. Wu, and T. Zhu, "Distributed construction of connected dominating sets optimized by minimum-weight spanning tree in wireless ad-hoc sensor networks," in *IEEE 17th International Conference on Computational Science and Engineering*, Chengdu, China, pp. 901-908, 2014. DOI: 10.1109/cse.2014.183.
- [11] J. J. Kponyo, Y. Kuang, E. Zhang, and K. Domenic, "VANET cluster-on-demand minimum spanning tree (MST) prim clustering algorithm," in *International Conference on Computational Problem-Solving*, Jiuzhai, China, pp. 101-104, 2013. DOI: 10.1109/iccps.2013.6893585.
- [12] X. Zhang and X. Zhang, "A binary artificial bee colony algorithm for constructing spanning trees in vehicular ad hoc networks," *Ad Hoc Networks*, vol. 58, pp. 198-204, Apr. 2017. DOI: 10.1016/j.adhoc.2016.07.001.



Jiwon Youn

is an undergraduate student majoring in Computer Science and Information Engineering at the Catholic University of Korea since 2018. Her research interests include databases, algorithms, and artificial intelligence.



Byungyeon Hwang

received his B.S. degree in Computer Engineering from Seoul National University, Korea in 1986, and M.S. and Ph.D. degrees in Computer Science from Korea Advanced Institute of Science and Technology (KAIST) in 1989 and 1994, respectively. He is a professor in the School of Computer Science and Information Engineering at the Catholic University of Korea. His research interests include database, social network analysis, and approximation algorithms.