RESEARCH

KEPCO Journal on Electric Power and Energy
p-ISSN 2465-8111, e-ISSN 2466-0124
Volume 8, Number 2, December 2022, pp. 159-179
DOI 10.18770/KEPCO.2022.08.02.159

Analysis of Distributed Computational Loads in Large-scale AC/DC Power System using Real-Time EMT Simulation

대규모 AC/DC 전력 시스템 실시간 EMP 시뮬레이션의 부하 분산 연구

In Kwon Park, Yi Zhong Hu, Yi Zhang, Hyun Keun Ku, Yong Han Kwon
박인권, 이종후, 이장, 구현근, 권용한

Abstract

Often a network becomes complex, and multiple entities would get in charge of managing part of the whole network. An example is a utility grid. While the entire grid would go under a single utility company's responsibility, the network is often split into multiple subsections. Subsequently, each subsection would be given as the responsibility area to the corresponding sub-organization in the utility company. The issue of how to make subsystems of adequate size and minimum number of interconnections between subsystems becomes more critical, especially in real-time simulations. Because the computation capability limit of a single computation unit, regardless of whether it is a high-speed conventional CPU core or an FPGA computational engine, it comes with a maximum limit that can be completed within a given amount of execution time. The issue becomes worsened in real time simulation, in which the computation needs to be in precise synchronization with the real-world clock.

When the subject of the computation allows for a longer execution time, i.e., a larger time step size, a larger portion of the network can be put on a computation unit. This translates into a larger margin of the difference between the worst and the best. In other words, even though the worst (or the largest) computational burden is orders of magnitude larger than the best (or the smallest) computational burden, all the necessary computation can still be completed within the given amount of time. However, the requirement of real-time makes the margin much smaller. In other words, the difference between the worst and the best should be as small as possible in order to ensure the even distribution of the computational load. Besides, data exchange/communication is essential in parallel computation, affecting the overall performance. However, the exchange of data takes time. Therefore, the corresponding consideration needs to be with the computational load distribution among multiple calculation units. If it turns out in a satisfactory way, such distribution will raise the possibility of completing the necessary computation in a given amount of time, which might come down in the level of microsecond order. This paper presents an effective way to split a given electrical network, according to multiple criteria, for the purpose of distributing the entire computational load into a set of even (or close to even) sized computational loads. Based on the proposed system splitting method, heavy computation burdens of large-scale electrical networks can be distributed to multiple calculation units, such as an RTDS real time simulator, achieving either more efficient usage of the calculation units, a reduction of the necessary size of the simulation time step, or both.

Keywords: Large network, EMT, RTDS, METIS

I. INTRODUCTION

In order to ensure the successful completion of technologically complex projects such as a large scale HVDC project, many different levels of studies become necessary. One such level is EMT (Electro-Magnetic Transient) level, which allows more detailed view over many different aspects of the system. Meanwhile, the usual way of performing the necessary system study has been based on transient stability analysis (TSA). The study is based on two fundamental assumptions: the balanced three phase system in the study subject, and the main focus of the study to be on the mechanical dynamics of large-capacity three phase synchronous machines. The result of the second assumption determines the time step of the necessary simulation, in the range foretold by the system frequency. In the 60 Hz power system, usually the size of the time step is half of the period determined by the frequency, which is 8.33mS. The transient stability analysis (TSA) of system studies can be utilized in many different levels of studies and can offer valuable insights regarding the application of the special technology. However, those two fundamental assumptions begin to reveal their limitation once the study requirement goes over into a different ground. In particular, the second assumption loses its ground once higher switching frequency devices are employed as a vehicle of the special technology application in transmission or distribution systems. The usual time step size under the TSA type study indicates that the maximum frequency of dynamics is the fundamental frequency of the system,

Article Information

Manuscript Received June 30, 2022, Accepted September 16, 2022, Published online December 30, 2022

The authors are with KEPCO Research Institute, Korea Electric Power Corporation, 105 Munji-ro Yuseong-gu, Daejeon 34056, Republic of Korea.

Correspondence Author: *Jintae Cho (jintae.cho@kepco.co.kr)*

which is usually far less than the fastest dynamics expected from the higher switching frequency systems. Thus, it becomes more likely to see that vital part of the system transient response would be missed, as long as such low sampling frequency simulation as the traditional study is conducted. Therefore, these compelling reasons necessitate that the application of the proper level of study method, an EMT level study in such cases.

While the importance of the EMT level study is now widely recognized, the method comes with its own set of limitations. One such limitation is the amount of calculation. Once the scope of the simulation becomes large, such as an extensive coverage of a transmission level grid system, or the detail of the model begins to climb up, such as the phase domain frequency dependent transmission line models, the amount of computational burden begins to lay an obstacle on the path to the achieve the desired study results by using the technique. In the case of the larger extent of the network, partitioning the network into the set of smaller subnetworks by using the long enough transmission lines as the boundary has been known. Such system partition, then, would allow each subsystem to be assigned on its own computational unit such as a core of multicore CPU. The transmission line interface between them guarantees the integrity of the entire simulation by providing the necessary time delays between them. As long as the size of the system is manageable, such manual technique, which requires the manual identification of such boundary lines, would work. However, once the size of the system becomes more realistic, or goes beyond the possible extent of the manual approach, the necessary partitioning work becomes difficult. Furthermore, such manual system partitioning does not guarantee that the system is partitioned in an optimal manner. For instance, if a network is split into multiple subsystems, there is no guarantee that the computational load of each system would become well balanced. In the worst case, one subsystem, resulting from the manual system partitioning, might heavily outweigh the rest in terms of the computational burden. Consequently, the heaviest burden would drive the size of the execution time, compromising the purpose of the system partitioning.

The system partitioning is a long sought-after problem in graph theory. Many versatile algorithms have been proposed and utilized in various areas. One of them is an algorithm called METIS. The algorithm was first proposed, then the implementation was made available through the developer group's website. Later on, mode variants came into the collection of the implementation such as PyMETIS. The algorithm and its implementation offer intuitive ways to provide the necessary input data and interpret the outcome of the execution of the algorithm. This paper introduces the application of the METIS algorithms into the area of the electrical grid network partitioning. The algorithm and its implementation are explained in the next chapter of this paper. At the later portion of the chapter, how to check the proper installation of the implementation package would be presented. Then, the application of the algorithms to a widely used model electrical grid model, IEEE 39 bus system, will follow. Two different approaches are introduced there: the first one is to use a single weighting factor per each node (or a virtual bus) and the second one is using two different weighting factors. Subsequently, the aptitude of the algorithm towards the given application would be verified. Lastly, this paper finishes with a couple of concluding remarks.

II. METIS AND ITS INSTALLATION

A graph partitioning problem such as the one given in the aforementioned chapter is a common problem in many areas. As the computer software execution model moves from a single concentrated execution to multiple execution units such as multicore system, the partitioning problem comes under a new spotlight. The many cores approach to the computational problem means the problem needs to be partitioned in an adequate way to fit the nature of the given computational machine, many cores connected by a communication network. Therefore, the computational load assigning and balancing problem becomes the network partitioning problem, with a set of constraints such as minimizing the amount of communication between cores.

One software package has been developed to tackle the problem. The package started from a single partitioning algorithm named 'multi-level k-way partitioning', but it evolved to cover multiple partitioning algorithms and began to see more applications in various field. The package is 'METIS', developed by a computer science and engineering lab at University of Minnesota[1]. The algorithm has been progressing in terms of the development and diversification. Some of its siblings are ParMETIS (Parallel Graph Partitioning and Fill-reducing Matrix Ordering) and hMETIS (Hypergraph & Circuit Partitioning). A Python variant can be also utilized, 'PyMETIS', if one is interested in [2]. A website, 'http://glaros.dtc.umn.edu/gkhome/metis/metis', has been the place for the information regarding the development. The package can be downloaded from there as well. Once it is downloaded, it can be built in many different environments including Microsoft Windows and using Visual Studio.

The key idea of the seminal algorithm under the METIS package is succinctly described in the reference[3].

The k-way graph partitioning problem is defined as follows: Given a graph G = (V, E) with |V| = n, partition V into k subsets, V1, V2, ... , Vk such that Vi ∩ Vj = ∅ for i ≠ j, |Vi| = n/k, and Ui Vi = V, and the number of edges of E whose incident vertices belong to different subsets is minimized. A k-way partitioning of V is commonly represented by a partitioning vector P of length n, such that for every vertex v $\in$ V, P[v] is an integer between 1 and k, indicating the partition to which vertex v belongs. Given a partitioning P, the number of edges whose incident vertices belong to different partitions is called the edge-cut of the partitioning. The following figure, Fig. 1, shows the steps to run the partitioning algorithm.

The coarsening steps is to reduce the number of vertex and corresponding edges, resulting in a smaller set of them, i.e., network. Then, once the initial partition phase completes, each 'coarsened'

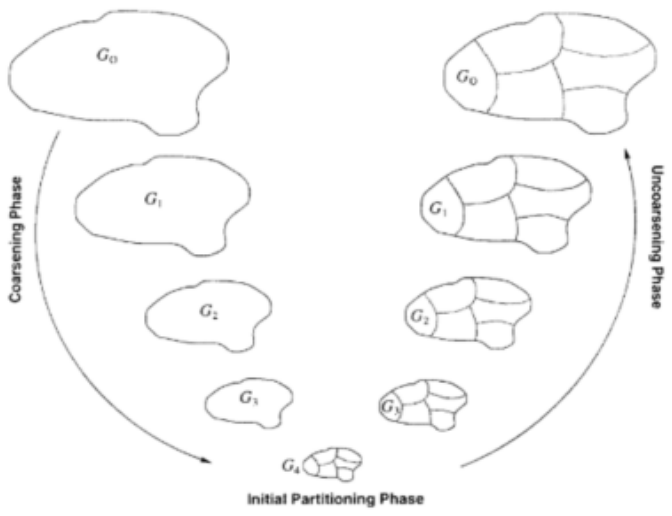

Fig. 1. Multilevel k-way partitioning scheme[3]

```
#
# /etc/pacman.conf
#
# See the pacman.conf(5) manpage for option and repository directives

#
# GENERAL OPTIONS
#
[options]
# The following paths are commented out with their default values listed.
# If you wish to use different paths, uncomment and update the paths.
#RootDir     = /
#DBPath      = /var/lib/pacman/
#CacheDir    = /var/cache/pacman/pkg/
#LogFile     = /var/log/pacman.log
#GPGDir      = /etc/pacman.d/gnupg/
HoldPkg     = pacman
#XferCommand = /usr/bin/curl -L -C - -f -o %o %u
#XferCommand = /usr/bin/wget --passive-ftp -c -O %o %u
#CleanMethod = KeepInstalled
Architecture = auto

# Pacman won't upgrade packages listed in IgnorePkg and members of IgnoreGroup
#IgnorePkg   =
#IgnoreGroup =

#NoUpgrade   =
#NoExtract   =

# Misc options
#UseSyslog
Color
#NoProgressBar
CheckSpace
#VerbosePkgLists
ParallelDownloads = 5

# By default, pacman accepts packages signed by keys that its local keyring
# trusts (see pacman-key and its man page), as well as unsigned packages.
SigLevel = Never
#SigLevel    = Required
LocalFileSigLevel = Optional
#RemoteFileSigLevel = Required
```

Fig. 2. pacman configuration modification

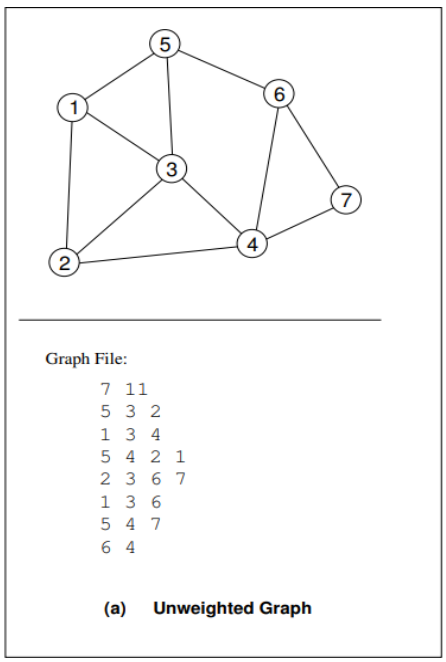

Fig. 3. A simple graph – given in METIS manual chapter4

```
pik140188B1JSI MSYS /d/parkik/MEng_Study/MSCI_7140_A01_QuantitiveAnalysisForManagement/Project/20191211_METIS/metis-5.1.0
$ gpmetis Fig2.txt 2
********************************************************************************
METIS 5.0 Copyright 1998-13, Regents of the University of Minnesota
 (HEAD: , Built on: Mar 27 2022, 23:09:35)
 size of idx_t: 32bits, real_t: 32bits, idx_t *: 64bits

Graph Information ---------------------------------------------------------
 Name: Fig2.txt, #Vertices: 7, #Edges: 11, #Parts: 2

Options -------------------------------------------------------------------
 ptype=kway, objtype=cut, ctype=shem, rtype=greedy, iptype=metisrb
 dbglvl=0, ufactor=1.030, no2hop=NO, minconn=NO, contig=NO, nooutput=NO
 seed=-1, niter=10, ncuts=1

Direct k-way Partitioning -------------------------------------------------
 - Edgecut: 4, communication volume: 5.

 - Balance:
     constraint #0:  1.143 out of 0.286

 - Most overweight partition:
     pid: 0, actual: 4, desired: 3, ratio: 1.14.

 - Subdomain connectivity: max: 1, min: 1, avg: 1.00

 - Each partition is contiguous.

Timing Information --------------------------------------------------------
  I/O:                          0.000 sec
  Partitioning:                 0.000 sec   (METIS time)
  Reporting:                    0.000 sec

Memory Information --------------------------------------------------------
  Max memory used:              0.050 MB
********************************************************************************
```

Fig. 4. METIS execution example

element of the result gets the refinement. The refinement expands the each 'coarsened' element of the previous steps.

What one would be more interested in is how to install the package and use it, rather going to the technical background of the package. Therefore, this chapter offers the explanation regarding the installation of the package. This chapter concludes with a simple example and its result obtained from the METIS, consequently the proper installation and operation of the package is verified. The very first step is downloading the package from the aforementioned website. It is in the form of the source code collection. Therefore, one needs to select the development environment where the source code

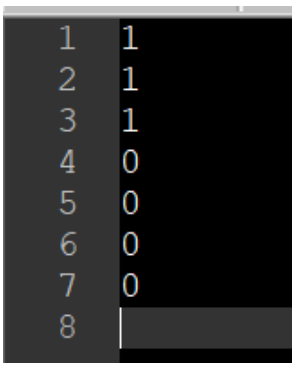

Fig. 5. Result of the simple graph partitioning

files of the package, downloaded from the website, can be compiled, linked and become a set of executables, which would be able to run on the selected environment. In the experiment reported in this document, a well-known unix-like environment, named MSYS2, was selected. One reason behind of the selection was that the original Visual Studio workspace included in the package (made by Visual Studio version 2010) did not work. An instruction file included in the package, 'Buid.txt', presents how to build the package by using gcc-tool chain in a unix-like environment such as the one chosen, MSYS2.

During the installation of the pre-requisite of the project, an error might occur. It might be related with an invalid security key, which would be related to a software package installation. There would be many different ways to overcome the issue, but in the process reported in this document, a recommendation (or instruction) given in this link(https://github.com/msys2/MSYS2-packages/issues/2343) was taken. It introduces a level of compromise in terms of the integrity of the software package installation by using the MSYS2 package manager ('pacman'), but it was taken due to its relative simplicity. The lines marked by a red box in the following figure, Fig. 2, shows the necessary modification for the purpose of disabling the security key ring checking of the 'pacman' package manager.

Unlike many people's conception regarding the MSYS2 environment, it does not come with the usual unix system compiler/linker toolchain, namely GCC. Therefore, the steps explained in this link (https://www.devdungeon.com/content/install-gcc-compiler-windows-msys2-cc) was followed to complete the installation of the compiler-linker toolchain in the given environment.

After the completion of the installation, a simple graph given in the fig. 2(a) of the manual chapter 4 was tested. The graph given in the figure is shown the figure below, Fig. 3. The figure also shows the input file which represents the graph in the style of the adjacent list. One needs to pay attention that the node (or vertices) index is implicit in this graph representation. In other words, the line number is used as the vertex index, instead of showing the index in an explicit way. For example, the second line of the file presents the connectivity status of the vertex 1. One can easily see it by comparing the line ('5 3 2') with the graphical representation of the graph in the upper portion of the figure.

The MSYS2 screen capture image shown below, Fig. 3, presents the output of the METIS execution, taking in the graph file presented in the lower portion of the figure, Fig. 2, as an input.

The input adjacent list file, presented in the bottom part of Fig. 3, was put as the input file, 'Fig2.txt'. Then the METIS executable was invoked with two command arguments. The first one is the name the input file ('Fig2.txt'). The second argument is the number of partitions. In the example presented in Fig. 4, '2' was give as the second argument, telling the METIS program to cut the input graph into two partitions. The result of the graph partition is given by a simple text file, 'Fig2.txt.part.2'. The first portion of this file name, 'Fig2.txt', is from the input file name and the rest of the name indicates that the graph was split into 2 parts, as given as a command line option when the program was executed. The following figure, Fig. 5, presents the result. Again, the vertex index is implicit. In order to clearly show the implicit index, the line number was also shown in the figure.

III. AN ELECTRIC GRID NETWORK PARTITIONING

The graph partitioning problem is to partition the vertices of a graph in p roughly equal partitions such that the number of edges connecting vertices in different partitions is minimized. An interesting application of the graph partitioning algorithm is the subsystem identification and splitting of an electrical utility grid network. The usual extent of the electrical grid can reach up to a wide coverage, often up to the entire territory of a country. While such extent is a given condition for any necessary work to be done, handling the entire grid at once and as a whole becomes a daunting task due to its size and complexity. Therefore, how to split a grid with an extensive size into a number of sub-units which would come under the manageable size limit has been an intriguing study topic, which drew attention from many researchers.

An exemplary system was selected and used in the study reported in this document. In other words, a realistic system was selected and posed as the problem, to be solved and reported in this

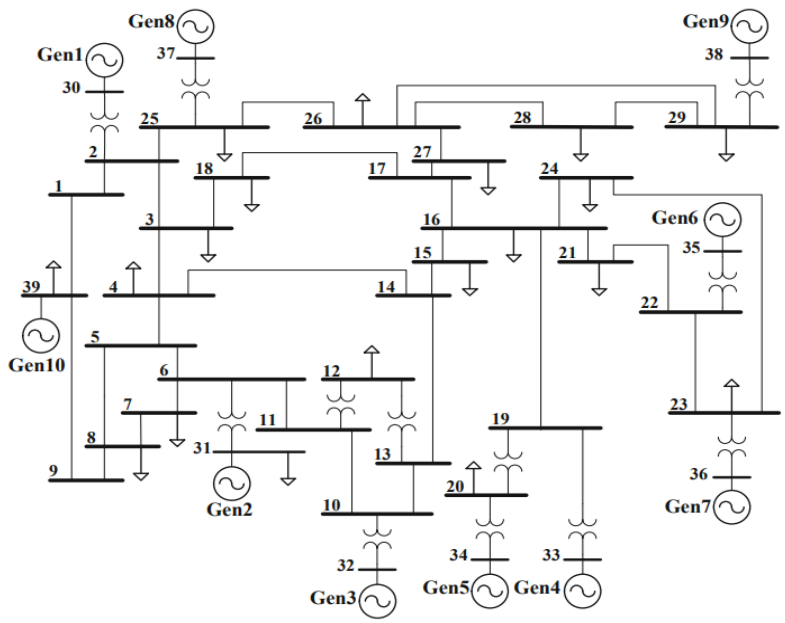

Fig. 6. Single-line diagram for the IEEE 39 bus power system[4]

TABLE 1
IEEE-39 bus system branch data[4]

Bus	Bus	Resistance (p.u./m)	Reactance (p.u./m)	Susceptance (p.u./m)	Transformer magn. (p.u.)	Transformer angle (deg.)
1	2	0.0035	0.0411	0.6987	0.000	0.0
1	39	0.0010	0.0250	0.7500	0.000	0.0
2	3	0.0013	0.0151	0.2572	0.000	0.0
2	25	0.0070	0.0086	0.1460	0.000	0.0
3	4	0.0013	0.0213	0.2214	0.000	0.0
3	18	0.0011	0.0133	0.2138	0.000	0.0
4	5	0.0008	0.0128	0.1342	0.000	0.0
4	14	0.0008	0.0129	0.1382	0.000	0.0
5	6	0.0002	0.0026	0.0434	0.000	0.0
5	8	0.0008	0.0112	0.1476	0.000	0.0
6	7	0.0006	0.0092	0.1130	0.000	0.0
6	11	0.0007	0.0082	0.1389	0.000	0.0
7	8	0.0004	0.0046	0.0780	0.000	0.0
8	9	0.0023	0.0363	0.3804	0.000	0.0
9	39	0.0010	0.0250	1.2000	0.000	0.0
10	11	0.0004	0.0043	0.0729	0.000	0.0
10	13	0.0004	0.0043	0.0729	0.000	0.0
13	14	0.0009	0.0101	0.1723	0.000	0.0
14	15	0.0018	0.0217	0.3660	0.000	0.0
15	16	0.0009	0.0094	0.1710	0.000	0.0
16	17	0.0007	0.0089	0.1342	0.000	0.0
16	19	0.0016	0.0195	0.3040	0.000	0.0
16	21	0.0008	0.0135	0.2548	0.000	0.0
16	24	0.0003	0.0059	0.0680	0.000	0.0
17	18	0.0007	0.0082	0.1319	0.000	0.0
17	27	0.0013	0.0173	0.3216	0.000	0.0
21	22	0.0008	0.0140	0.2565	0.000	0.0
22	23	0.0006	0.0096	0.1846	0.000	0.0
23	24	0.0022	0.0350	0.3610	0.000	0.0
25	26	0.0032	0.0323	0.5130	0.000	0.0
26	27	0.0014	0.0147	0.2396	0.000	0.0
26	28	0.0043	0.0474	0.7802	0.000	0.0
26	29	0.0057	0.0625	1.0290	0.000	0.0
28	29	0.0014	0.0151	0.2490	0.000	0.0
12	11	0.0016	0.0435	0.0000	1.006	0.0
12	13	0.0016	0.0435	0.0000	1.006	0.0
6	31	0.0000	0.0250	0.0000	1.070	0.0
10	32	0.0000	0.0200	0.0000	1.070	0.0
19	33	0.0007	0.0142	0.0000	1.070	0.0
20	34	0.0009	0.0180	0.0000	1.009	0.0
22	35	0.0000	0.0143	0.0000	1.025	0.0
23	36	0.0005	0.0272	0.0000	1.000	0.0
25	37	0.0006	0.0232	0.0000	1.025	0.0
2	30	0.0000	0.0181	0.0000	1.025	0.0
29	38	0.0008	0.0156	0.0000	1.025	0.0
19	20	0.0007	0.0138	0.0000	1.060	0.0

document. The system is usually called as IEEE 39 bus system, because the system has 39 buses (or nodes, following notation from graph theories). The system represents for part of the north-eastern electrical grid at the United States. Part of the grid network represents system equivalence, meaning that those parts are representing much larger portion of the network. Those system equivalences are the representations of the neighboring systems, connected to the network which is depicted in detail. The following figure, Fig. 6, presents the example system in a graphical manner

The partitioning of a grid network, such as the one presented in Fig. 6, would utilize the characteristics of the electrical network. In particular, the following two points determines the size of a partition and its composition. The first point is the number of buses (or nodes) in each partition. One possible objective of partitioning is to assign each group into the sub-divisions in a utility company. For example, a utility company can establish the subdivisions according to the

TABLE 2
Branch data screening

```
[      0]:      1 -      2 TRL
[      1]:      1 -     39 TRL
[      2]:      2 -      3 TRL
[      3]:      2 -     25 TRL
[      4]:      3 -      4 TRL
[      5]:      3 -     18 TRL
[      6]:      4 -      5 TRL
[      7]:      4 -     14 TRL
[      8]:      5 -      6 RL
[      9]:      5 -      8 TRL
[     10]:      6 -      7 TRL
[     11]:      6 -     11 TRL
[     12]:      7 -      8 TRL
[     13]:      8 -      9 TRL
[     14]:      9 -     39 TRL
[     15]:     10 -     11 RL
[     16]:     10 -     13 RL
[     17]:     13 -     14 TRL
[     18]:     14 -     15 TRL
[     19]:     15 -     16 TRL
[     20]:     16 -     17 TRL
[     21]:     16 -     19 TRL
[     22]:     16 -     21 TRL
[     23]:     16 -     24 TRL
[     24]:     17 -     18 TRL
[     25]:     17 -     27 TRL
[     26]:     21 -     22 TRL
[     27]:     22 -     23 TRL
[     28]:     23 -     24 TRL
[     29]:     25 -     26 TRL
[     30]:     26 -     27 TRL
[     31]:     26 -     28 TRL
[     32]:     26 -     29 TRL
[     33]:     28 -     29 TRL
[     34]:      2 -     30 UTRF
[     35]:      6 -     31 UTRF
[     36]:     10 -     32 UTRF
[     37]:     12 -     11 TRF
[     38]:     12 -     13 TRF
[     39]:     19 -     20 TRF
[     40]:     19 -     33 UTRF
[     41]:     20 -     34 UTRF
[     42]:     22 -     35 UTRF
[     43]:     23 -     36 UTRF
[     44]:     25 -     37 UTRF
[     45]:     29 -     38 UTRF
```

geographical boundary. However, each sub-division would become in charge of the maintenance of the electrical grid in its territory. Consequently, the partitioning of the network and its corresponding assignment of the responsibility needs to be balanced, in order to maintain a certain level of fairness in the task assignment to each sub-division. The second point is the electrical distance. Frequently, the electrical distance, usually measured as the magnitude of impedance, would corresponds to the geographical distance. Therefore, it would be a reasonable measure, offering the information of the configuration of the electrical grid network in geography. The information, usually called as line information, of the example grid is given by the following table, Table 1.

Once the necessary data regarding the topology of the network is ready, the next step is to screen the branch data to identify the candidates of the cut-sets. In other words, the electrical length of all the branches in the network need to be calculated, then each length needs to be compared to a given criterion(e.g., a electromagnetic travel time = sqrt(L*C)) in order to identify the branches which are long enough to be used as the border between partitions. First, a transformer branch cannot become the subject of split, or connecting edge between two partitions. Second, a shorter branch cannot become the subject of split either. 'Short' means the impedance of the branch is less than a certain value. The value is determined by the minimum travel time (or propagation delay in Electromagnetic terms). All of those branches in the given data, Table 1, were checked. The following table, Table 2, shows the result. Two tags, 'TRF' and 'UTRF', shows the branches which belong to the first category, transformer branch. Another tag, 'RL' designates the branches too short to be cut, belonging to the second category.

The following figure, Fig. 7, presents the information given in the Table 2 in a graphical manner. In the figure, Fig. 7, the green boxes

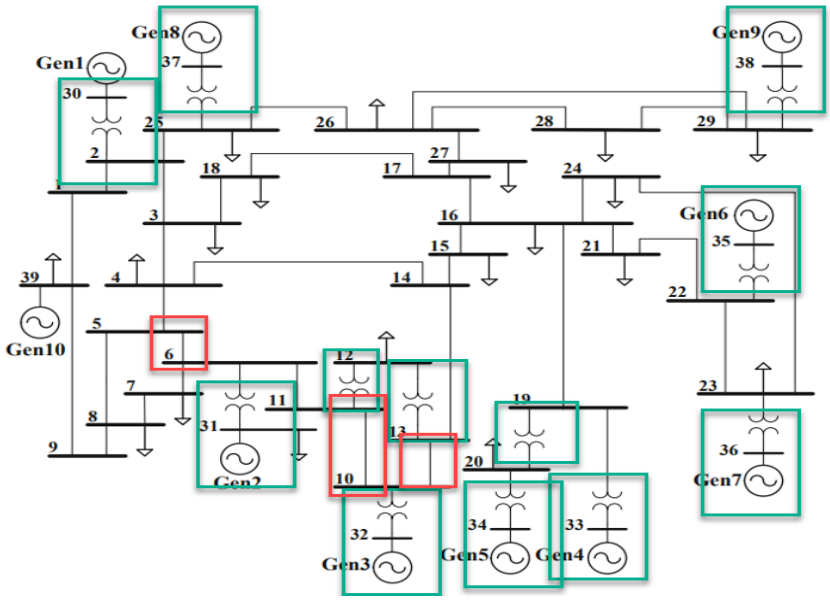

Fig. 7. Result of branch data screening

TABLE 3
IEEE-39 bus system METIS input file

```
25  31  011
1 2 1 25 1
1 1 1 3 1 20 1
1 2 1 4 1 14 1
1 3 1 5 1 10 1
2 4 1 7 1 6 1 9 1
1 5 1 7 1
1 5 1 6 1 8 1
1 7 1 25 1
4 5 1 10 1
1 4 1 9 1 11 1
1 10 1 12 1
1 11 1 13 1 15 1 16 1 19 1
1 12 1 14 1 22 1
1 3 1 13 1
2 12 1
1 12 1 17 1
1 16 1 18 1
1 17 1 19 1
1 12 1 18 1
1 2 1 21 1
1 20 1 22 1 23 1 24 1
1 13 1 21 1
1 21 1 24 1
1 21 1 23 1
1 1 1 8 1
```

show the location of the first condition, transformers, while the red boxes show the location of the second criterion, the short branches.

The following table, Table 3, shows the METIS input file, reflecting the information presented in the Table 2 and the Fig. 7.

The input file is saved as 'METIS_input2.txt'. Then, the number of the partition is decided. In the case of this example, the number was decided as 4. The next step is to run METIS program with the input file and the desired number of the partitions. The necessary command is 'gmetis -ptype=rb METIS_input2.txt 4'. This command, once given in the MSYS2 command, shell will execute the METIS, with the given number of partitions. The result is written in a text time, with the name of <input text tile>.part.#. '#' signifies the total number of partitions given as a part of the command. Here is a screen capture image, presenting an example of executing the METIS and the execution result.

The output text file is given in the following table, Table 4.

The node index is implicit, meaning that the file only shows the information of the partition where the vertex (i.e., node) belongs to. For instance, the very first line shows that vertex 1 (assuming the index starts from 1) belongs to partition 0 (assuming the index starts from 0). The output file can be re-interpreted to assign the partition number to the vertex or node index of the electrical network. The result of the re-interpretation is given in the following table, Table 5.

The partition result is displayed upon the graphical representtation of the electrical network, in the following figure, Fig. 9.

The next attempt is adding more weight to each vertex of the network and use it as additional criterion in the partitioning. When METIS runs the network and tries to partition, multiple goals can be given as the guideline of the partitioning. The most fundamental goals are these: the first was the balancing, meaning that the sum of weights of each partition needs to be balanced. The next is the minimization of the number of edge-cut. In other words, the algorithm tries to cut as few branches as possible. In this attempt, an extra goal was set. The goal was given as an additional weighting factor of the node. The extra weight is made up by these factors. A generator (an entity represented by a circle and a number within it, shown in Fig. 6) would impose a certain weight. A transformer would add its own weight and each terminal of the edge-cut would add its own. Probably handing the more involved condition and producing the necessary METIS input file, according to the given condition would be difficult, if one tries to accomplish it in a manual way. Therefore, a MATLAB M script was written to automate the necessary steps to read the data files and produce the METIS input file. The M script was presented in Appendix A. The necessary input data files for the execution of the M script file were presented in Appendix B. Those data files are representing the system information of the given example, IEEE 39 bus system.

The table above, Table 6, presents the METIS input file for the second attempt, with multiple weighting factors upon each node. The

```
pik14@O18BIJSI MSYS /d/parkik/MEng_Study/MSCI_7140_A01_QuantitiveAnalysisForManagement/Project/20191211_METIS/metis-5.1.0
$ gpmetis -ptype=rb METIS_input2.txt 4
********************************************************************************
METIS 5.0 Copyright 1998-13, Regents of the University of Minnesota
 (HEAD: , Built on: Mar 27 2022, 23:09:35)
 size of idx_t: 32bits, real_t: 32bits, idx_t *: 64bits

Graph Information ---------------------------------------------------------
 Name: METIS_input2.txt, #Vertices: 25, #Edges: 31, #Parts: 4

Options -------------------------------------------------------------------
 ptype=rb, objtype=cut, ctype=shem, rtype=fm, iptype=grow
 dbglvl=0, ufactor=1.001, no2hop=NO, minconn=NO, contig=NO, nooutput=NO
 seed=-1, niter=10, ncuts=1

Recursive Partitioning ----------------------------------------------------
 - Edgecut: 7, communication volume: 13.

 - Balance:
     constraint #0:  1.067 out of 0.533

 - Most overweight partition:
     pid: 1, actual: 8, desired: 7, ratio: 1.07.

 - Subdomain connectivity: max: 3, min: 1, avg: 2.00

 - Each partition is contiguous.

Timing Information --------------------------------------------------------
  I/O:                          0.015 sec
  Partitioning:                 0.000 sec   (METIS time)
  Reporting:                   -0.000 sec

Memory Information --------------------------------------------------------
  Max memory used:              0.050 MB
********************************************************************************
```

Fig. 8. METIS execution – IEEE 39 bus example

TABLE 4
IEEE-39 bus system system partitioning result

0
1
1
1
0
0
0
0
1
1
3
3
2
2
3
3
3
3
3
2
2
2
2
2
0

TABLE 5
Partitioning result – re-interpreted

1	1
2	2
3	2
4	2
5	1
6	1
7	1
8	1
9	1
10	2
11	2
12	2
13	2
14	2
15	4
16	4
17	3
18	3
19	4
20	4
21	4
22	4
23	4
24	4
25	3
26	3
27	3
28	3
29	3
39	1
30	2
31	1
32	2
33	4
34	4
35	4
36	4
37	3
38	3

file was generated from the MATLAB M script presented in the Appendix A and those input files presented in the Appendix B. The first line tells the information to the METIS program. The second last number of the first line, 011, tells METIS that the input file has the weight on both the node (i.e., vertex) and branch (i.e., edge). The last number of the first line, '2', tells to the METIS that the weight value given to each node is 2. The second line of the file shows an example. The node index 1 (assuming the starting index is 1, as the same as the previous attempt) has two weighting factors, 1 and 0. Then, the rest of the parameters on the line show the branch adjacent list with the weighting of each branch. In other words, '2 1' presents a branch from node 1 to node 2, with the weighting of 1.

In order to utilize the partitioning results from METIS, the result must be re-interpreted. The re-interpretation process must take care of the bus renaming (from internal bus indexing, through hidden bus indexing, to the final original indexing) and the allocation of the partition numbers for both merged buses and the generator buses,

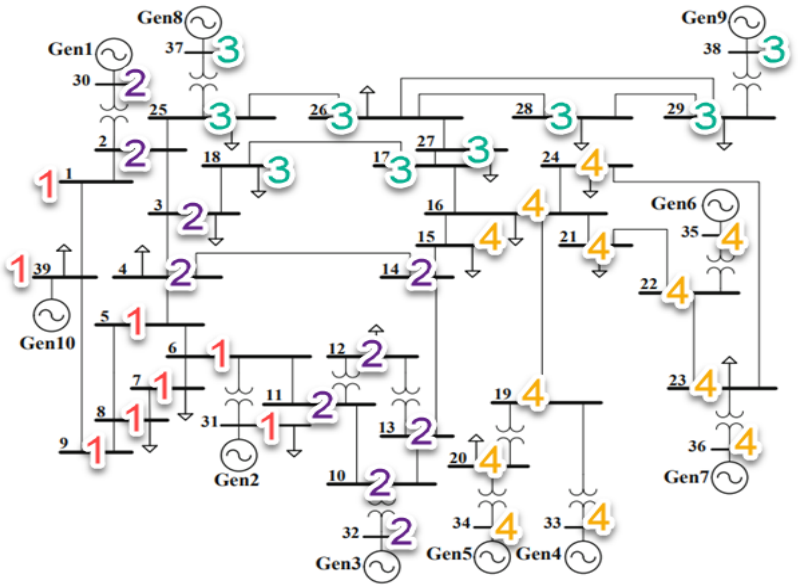

Fig. 9. IEEE 39 bus system partitioning result

TABLE 6
IEEE 39 bus system METIS input file-2

```
25  31  011 2
1 0 2 1 25 1
1 20 1 1 3 1 20 1
1 0 2 1 4 1 14 1
1 0 3 1 5 1 10 1
2 20 4 1 7 1 6 1 9 1
1 0 5 1 7 1
1 0 5 1 6 1 8 1
1 0 7 1 25 1
4 40 5 1 10 1
1 0 4 1 9 1 11 1
1 0 10 1 12 1
1 0 11 1 13 1 15 1 16 1 19 1
1 0 12 1 14 1 22 1
1 0 3 1 13 1
2 50 12 1
1 0 12 1 17 1
1 20 16 1 18 1
1 20 17 1 19 1
1 0 12 1 18 1
1 20 2 1 21 1
1 0 20 1 22 1 23 1 24 1
1 0 13 1 21 1
1 0 21 1 24 1
1 20 21 1 23 1
1 20 1 1 8 1
```

which became merged with its high voltage side bus across its unit transformer. Another MATLAB M script was written to automate the necessary steps. The code was presented in Appendix C. The following table, Table 7, presents the result of the re-interpretation.

The partition result is displayed upon the graphical representation of the electrical network, in the following figure, Fig. 10.

This paper presented a solution to an electrical grid network partitioning problem, using a general graph partitioning algorithm named METIS. The algorithm, METIS, was selected and examined to see if it could be applied for the solution of a given problem. The procedure to acquire the METIS package as well as the procedure for

TABLE 7
Re-interpreted partitioning result

1	4
2	4
3	2
4	3
5	3
6	3
7	4
8	4
9	4
10	3
11	3
12	3
13	3
14	3
15	2
16	2
17	2
18	2
19	2
20	2
21	1
22	1
23	1
24	1
25	4
26	1
27	2
28	1
29	1
39	4
30	4
31	3
32	3
33	2
34	2
35	1
36	1
37	4
38	1

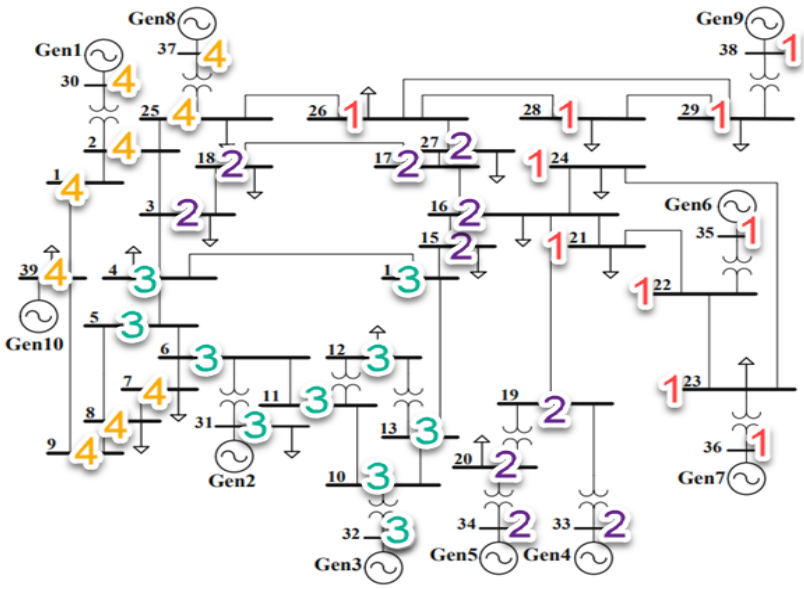

Fig. 10. IEEE 39 bus system partitioning result-2

the installation was described. The completeness of the installation was verified by using a simple test case, which was given as part of the METIS manual. Then the algorithm was applied to the given problem, partitioning a practical electrical utility grid with an adequate level of complexity. The tests successfully demonstrated that the METIS algorithm could solve the given problem. The test results presented in this paper show that the given graph (the electric utility grid network) can be split according to a pre-determined objective, such as the different weighting factor. The partitioning result would have many different applications. One such application is to use the partitioning result as a guideline to optimize the computation loads in each computational unit with minimum communications.

References

[1] Karypis Lab. "METIS - Serial Graph Partitioning and Fill-reducing Matrix Ordering." (accessed.)

[2] "PyMetis: A Python Wrapper for METIS." https://github.com/inducer/pymetis (accessed.)

[3] G. Karypis and V. Kumar, "Multilevelk-way Partitioning Scheme for Irregular Graphs," Journal of Parallel and Distributed Computing, vol. 48, no. 1, pp. 96-129, 1998/01/10/ 1998, doi: https://doi.org/10.1006/jpdc.1997.1404.

[4] V. Dinavahi and N. Lin, Parallel Dynamic and Transient Simulation of Large-Scale Power Systems: A High Performance Computing Solution. Cham: Springer International Publishing AG, 2021, https://doi.org/10.1007/978-3-030-86782-9

APPENDIX I

MATLAB M script to produce a METIS input file

```
clc;
clear all;

BUS_NUMBER = 30; % reduced
BRANCH_NUMBER = 37;

vertex = 1:BUS_NUMBER;
weight = ones(1,BUS_NUMBER);

branch = zeros(BRANCH_NUMBER,2);

% read in branch info

i=0;
j=0;
cross=0;
m=0;
n=0;

k=1;

real_hid = zeros(BUS_NUMBER,2);

% read the mapping between hid and real

FILE_NAME = '.¥InputFiles¥hid_real_mapping.txt';

fileID = fopen(FILE_NAME, 'r');
real_bus_number =[];
hid_bus_number =[];

while ~feof(fileID)

    tline = fgetl(fileID);
    tline = strtrim(tline);

    if (tline(1) == '#') || (strlength(tline) == 0)
        continue;
    end

    temp_A = sscanf(tline, '%d');
    real_hid(k, :) = [temp_A(2),  temp_A(1)];
    hid_bus_number(k) = temp_A(1);
    real_bus_number(k) = temp_A(2);
    k = k + 1;

end

fclose(fileID);

real2hid_Map = containers.Map(real_bus_number, hid_bus_number);
hid2real_Map = containers.Map(hid_bus_number, real_bus_number);


cross_count = 0;

FILE_NAME = '.¥InputFiles¥reduced_system_topology.txt';
fileID = fopen(FILE_NAME, 'r');

k=1;

while ~feof(fileID)

    tline = fgetl(fileID);
    tline = strtrim(tline);

    if (tline(1) == '#') || (strlength(tline) == 0)
        continue;
    end
```

```
    temp_A = sscanf(tline, '%d');
    i = real2hid_Map(temp_A(1));
    j = real2hid_Map(temp_A(2));
    cross = temp_A(3);

    % ensure i<j
    if (i>j)
        m=i;
        i=j;
        j=m;
    end

    % merge
    if (cross == 0)
        if(weight(i) == 0)
            i = vertex(i);
        end
        if(weight(j) == 0)
            j = vertex(j);
        end

        % already merged
        if(i==j)
            continue;
        end

        if (i~=vertex(i) || j ~=vertex(j) || weight(i) ==0 || weight(j)==0)
            disp('Bug 1');
            return;
        end

        % merge vertex j to i
        if(i < j)
            for (m = 1:BUS_NUMBER)
                if (vertex(m) == j)
                    vertex(m) = i;
                end
            end
            weight(i) = weight(i)+weight(j);
            weight(j) = 0;
        % merge vertex i to j
        else
            for (m = 1:BUS_NUMBER)
                if (vertex(m) == i)
                    vertex(m) = j;
                end
            end
            weight(j) = weight(i)+weight(j);
            weight(i) = 0;
        end
    else % cross line
        cross_count = cross_count+1;
        branch(cross_count, :) = [i  j];
    end

end

% weight(i): 본 모선에 머지된 모선 갯수 + 1, 피머지된 모선은 0
fclose(fileID);

% update the branches (cross lines) after vertex merge

branch2 = zeros(cross_count,2);
cross_count2 = 0;
for i = 1: cross_count
    m = vertex((branch(i,1)));
    n = vertex((branch(i,2)));
    if (m == n)
        continue; % the nodes on the sides of this crossline have merged
    end

    cross_count2 = cross_count2 +1;
```

```
    if(m<n)
        branch2(cross_count2,:)= [ m,  n];
    else
        branch2(cross_count2,:)= [ n,  m];
    end
end

% create the new vertex and the map between old and new
vertex_new_count = 0;
check=0;

for i = 1: BUS_NUMBER

    if (weight(i) ~= 0)
        check = check + weight(i);
        vertex_new_count = vertex_new_count+1;
        if (i ~= vertex(i))
            disp('Bug 3');
            return;
        end

        vertex_new(vertex_new_count) = i;
        vertex_new_weight(vertex_new_count) = weight(i);

        for (m = 1:BUS_NUMBER)
            if (vertex(m) == i)
                % map_old_to_new: 히든모선번호와 프로그램내 내부모선번호간 맵
                map_old_to_new(m) = vertex_new_count;
            end
        end

    end
end

if(check ~= BUS_NUMBER)
    disp('Bug 2');
    return;
end

% update branch2 to branch3, with vertex_new
branch3 = zeros(cross_count2,2);
cross_count3 = 0;
for i = 1: cross_count2
    m = map_old_to_new(branch2(i,1));
    n = map_old_to_new(branch2(i,2));

    if(m ==0 || n == 0)
        disp('Bug 4');
        return;
    end
    if (m == n)
        continue; % the nodes on the sides of this crossline have merged
    end

    cross_count3 = cross_count3 +1;

    if(m<n)
        branch3(cross_count3,:)= [m, n];
    else
        branch3(cross_count3,:)= [n, m];
    end
end

% for the easy use, sort the branch3 (Bubble)
temp_branch=[0, 0];
for i = cross_count3:-1:2
    for j = 1:i-1
        % compare branch3(j,1) and branch3(j+1,1)
        if(branch3(j,1)>branch3(j+1,1))
            temp_branch = branch3(j,:);
            branch3(j,:) = branch3(j+1,:);
            branch3(j+1,:) = temp_branch;
```

```
        end
    end
end

% 두모선간 복수의 브랜치가 있는지 확인

branch_new =zeros(cross_count3, 2);
branch_new_count = 0;
branch_new_weight = zeros(1,cross_count3);

branch_new_count =1;
branch_new(1,:)=branch3(1,:);
branch_new_weight(1)=1;

for i= 2:cross_count3
    % make sure branch3(i,:) is different with branches already in
    % branch_new
    flag=0;
    for (j=1:i-1)
        if(branch3(i,1) == branch_new(j,1) && branch3(i,2) == branch_new(j,2))
            flag = 1;
            branch_new_weight(j) = branch_new_weight(j)+1;
        end
    end

    if(flag ==0)
        branch_new_count = branch_new_count+1;
        branch_new(branch_new_count,:) = branch3(i,:);
        branch_new_weight(branch_new_count) = 1;
    end
end


check=0;

for i = 1:branch_new_count
    check = check+branch_new_weight(i);
end
if check~=cross_count3
    disp('Bug 5');
    return;
end

disp('reduced system topology process finished');

% The second weight for vertex_new (mimi systems) --> load unit
%  for each vertex_new (mini systems), we count
%       transformer
%       generator
%       t-line ?

vertex_new_weight2 = zeros(1, vertex_new_count);

TRANSFORMER_LOAD    = 10;
GENERATOR_LOAD      = 20;
TLINE_TERMINAL_LOAD = 5;

% read the transformers info
% real bus number in this file
FILE_NAME = '.¥InputFiles¥transformers.txt';
fileID = fopen(FILE_NAME, 'r');

while ~feof(fileID)

    tline = fgetl(fileID);
    tline = strtrim(tline);

    if (strlength(tline) == 0) || (tline(1) == '#')
        continue;
    end

    temp_A = sscanf(tline, '%d');
    m = real2hid_Map(temp_A(1));
```

```
    n = real2hid_Map(temp_A(2));

    i = map_old_to_new(m);
    j = map_old_to_new(n);

    if (i ~=j)
        disp('Bug 8');
        return;
    end

    vertex_new_weight2(i) = vertex_new_weight2(i) + TRANSFORMER_LOAD;

end

fclose(fileID);

% read the generators info
% real bus number in this file
FILE_NAME = '.¥InputFiles¥generators.txt';
fileID = fopen(FILE_NAME, 'r');
merged_generators =[];
merged_generator_count=0;

while ~feof(fileID)
    tline = fgetl(fileID);
    tline = strtrim(tline);

    if (strlength(tline) == 0) || (tline(1) == '#')
        continue;
    end

    temp_A = sscanf(tline, '%d');
    m = temp_A(1);
    n = temp_A(2);

    if (n~=0)
        merged_generator_count = merged_generator_count+1;      % these generators have terminal transformers and been
merged in reduce syste
        merged_generators(merged_generator_count,:) = [m,n];

        kk = real2hid_Map(n);
        i = map_old_to_new(kk);
        vertex_new_weight2(i) = vertex_new_weight2(i) + GENERATOR_LOAD;
    else  % n==0
        kk = real2hid_Map(m);
        i = map_old_to_new(kk);
        vertex_new_weight2(i) = vertex_new_weight2(i) + GENERATOR_LOAD;
    end

end

fclose(fileID);

% read the t-line info
FILE_NAME = '.¥InputFiles¥reduced_system_topology.txt';
fileID = fopen(FILE_NAME, 'r');

% ??
while ~feof(fileID)

    tline = fgetl(fileID);
    tline = strtrim(tline);

    if (strlength(tline) == 0) || (tline(1) == '#')
        continue;
    end

    temp_A = sscanf(tline, '%d');

    m = (real2hid_Map(temp_A(1)));
    n = (real2hid_Map(temp_A(2)));

    cross = temp_A(3);
```

```
    if (cross == 1) % this is a T-line

        i = map_old_to_new(m);
        %?
        % vertex_new_weight2(i) = vertex_new_weight2(i) + TLINE_TERMINAL_LOAD;
        j = map_old_to_new(n);
        %?
        % vertex_new_weight2(j) = vertex_new_weight2(j) + TLINE_TERMINAL_LOAD;

    end
end

fclose(fileID);

% write the input file for METIS

FILE_NAME = 'METIS_input.txt';
fileID = fopen(FILE_NAME, 'w');
fprintf(fileID, '%d  %d  011 2¥n', vertex_new_count,branch_new_count);

for i=1:vertex_new_count

    % every vertex deserves one line
    % first are the vertex weights
    fprintf(fileID, '%d %d ', vertex_new_weight(i), vertex_new_weight2(i));
    % each branch counted for both connected vertexs
    for j=1:branch_new_count
        if (branch_new(j,1) == i )
            fprintf(fileID, '%d %d ', branch_new(j,2), branch_new_weight(j));
        end

        if (branch_new(j,2) == i )
            fprintf(fileID, '%d %d ', branch_new(j,1), branch_new_weight(j));
        end
    end
    fprintf(fileID, '¥n', vertex_new_weight(i));

end


fclose(fileID);

disp('METIS input file written');
```

APPENDIX II

IEEE 39 bus system data files('hid_real_mapping.txt')

```
# from split.txt
# hid real
    1       1
    2       2
    3       3
    4       4
    5       5
    6       6
    7       7
    8       8
    9       9
   10      10
   11      11
   12      12
   13      13
   14      14
   15      15
   16      16
   17      17
   18      18
   19      19
   20      20
   21      21
   22      22
   23      23
   24      24
   25      25
   26      26
   27      27
   28      28
   29      29
   30      39
```

IEEE 39 bus system data files('reduced_system_topology.txt')

```
# from split.txt
# from to 1-TRL, 0-RL
    1     2 1
    1    39 1
    2     3 1
    2    25 1
    3     4 1
    3    18 1
    4     5 1
    4    14 1
    5     6 0
    5     8 1
    6     7 1
    6    11 1
    7     8 1
    8     9 1
    9    39 1
   10    11 0
   10    13 0
   13    14 1
   14    15 1
   15    16 1
   16    17 1
   16    19 1
   16    21 1
   16    24 1
   17    18 1
   17    27 1
   21    22 1
   22    23 1
   23    24 1
```

```
25     26 1
26     27 1
26     28 1
26     29 1
28     29 1
12     11 0
12     13 0
19     20 0
```

IEEE 39 bus system data files('transformers.txt')

```
# real bus number in this file
    12     11 2
    12     13 2
    19     20 2
```

IEEE 39 bus system data files('generator.txt')

```
# real bus number in this file
# genbus hvbus
    30          2
    31          6
    32         10
    33         19
    34         20
    35         22
    36         23
    37         25
    38         29
    39          0
```

APPENDIX III

MATLAB M script to postprocess a METIS output file

```
clc;
clear all;

BUS_NUMBER = 30; % reduced
BRANCH_NUMBER = 37;
vertex = 1:BUS_NUMBER;
weight = ones(1,BUS_NUMBER);
branch = zeros(BRANCH_NUMBER,2);

% read in branch info
cross=0;
k=1;

real_hid = zeros(BUS_NUMBER,2);

% read the mapping between hid and real

FILE_NAME = '.¥InputFiles¥hid_real_mapping.txt';

fileID = fopen(FILE_NAME, 'r');
real_bus_number =[];
hid_bus_number =[];

while ~feof(fileID)

    tline = fgetl(fileID);
    tline = strtrim(tline);
```

```
    if (tline(1) == '#') || (strlength(tline) == 0)
        continue;
    end

    temp_A = sscanf(tline, '%d');
    real_hid(k, :) = [temp_A(2),  temp_A(1)];
    hid_bus_number(k) = temp_A(1);
    real_bus_number(k) = temp_A(2);
    k = k + 1;

end

fclose(fileID);

real2hid_Map = containers.Map(real_bus_number, hid_bus_number);
hid2real_Map = containers.Map(hid_bus_number, real_bus_number);

cross_count = 0;

FILE_NAME = '.¥InputFiles¥reduced_system_topology.txt';
fileID = fopen(FILE_NAME, 'r');

k=1;

while ~feof(fileID)

    tline = fgetl(fileID);
    tline = strtrim(tline);

    if (tline(1) == '#') || (strlength(tline) == 0)
        continue;
    end

    temp_A = sscanf(tline, '%d');
    i = real2hid_Map(temp_A(1));
    j = real2hid_Map(temp_A(2));
    cross = temp_A(3);

    % ensure i<j
    if (i>j)
        m=i;
        i=j;
        j=m;
    end

    % merge
    if (cross == 0)
        if(weight(i) == 0)
            i = vertex(i);
        end
        if(weight(j) == 0)
            j = vertex(j);
        end

        % already merged
        if(i==j)
            continue;
        end

        if (i~=vertex(i) || j ~=vertex(j) || weight(i) ==0 || weight(j)==0)
            disp('Bug 1');
            return;
        end

        % merge vertex j to i
        if(i < j)
            for (m = 1:BUS_NUMBER)
                if (vertex(m) == j)
                    vertex(m) = i;
                end
            end
            weight(i) = weight(i)+weight(j);
```

```
            weight(j) = 0;
        % merge vertex i to j
        else
            for (m = 1:BUS_NUMBER)
                if (vertex(m) == i)
                    vertex(m) = j;
                end
            end
            weight(j) = weight(i)+weight(j);
            weight(i) = 0;
        end
    else % cross line
        cross_count = cross_count+1;
        branch(cross_count, :) = [i  j];
    end

end

% weight(i): 본 모선에 머지된 모선 갯수 + 1, 피머지된 모선은 0
fclose(fileID);

% update the branches (cross lines) after vertex merge

branch2 = zeros(cross_count,2);
cross_count2 = 0;
for i = 1: cross_count
    m = vertex((branch(i,1)));
    n = vertex((branch(i,2)));
    if (m == n)
        continue; % the nodes on the sides of this crossline have merged
    end

    cross_count2 = cross_count2 +1;

    if(m<n)
        branch2(cross_count2,:)= [m, n];
    else
        branch2(cross_count2,:)= [n, m];
    end
end


% create the new vertex and the map between old and new
vertex_new_count = 0;
check=0;

for i = 1: BUS_NUMBER

    if (weight(i) ~= 0)
        check = check + weight(i);
        vertex_new_count = vertex_new_count+1;
        if (i ~= vertex(i))
            disp('Bug 3');
            return;
        end

        vertex_new(vertex_new_count) = i;
        vertex_new_weight(vertex_new_count) = weight(i);

        for (m = 1:BUS_NUMBER)
            if (vertex(m) == i)
                % map_old_to_new: 히든모선번호와 프로그램내 내부모선번호간 맵
                map_old_to_new(m) = vertex_new_count;
            end
        end

    end
end

% read the generators info
% real bus number in this file
FILE_NAME = '.¥InputFiles¥generators.txt';
```

```
fileID = fopen(FILE_NAME, 'r');
merged_generators =[];
merged_generator_count=0;

while ~feof(fileID)
    tline = fgetl(fileID);
    tline = strtrim(tline);

    if (strlength(tline) == 0) || (tline(1) == '#')
        continue;
    end

    temp_A = sscanf(tline, '%d');
    m = temp_A(1);
    n = temp_A(2);

    if (n~=0)

        % these generators have terminal transformers and been merged in
        % reduce system
        merged_generator_count = merged_generator_count+1;
        merged_generators(merged_generator_count,:) = [m,n];

    end

end

fclose(fileID);


%%%%

[file,path] = uigetfile('*.*');

if isequal(file,0)
   disp('User selected Cancel');
else
   disp(['Selected file', fullfile(path,file)]);
end

FILE_NAME = fullfile(path,file);
fileID = fopen(FILE_NAME, 'r');
vertex_new_partiton =[];

for k = 1: vertex_new_count
    tline = fgetl(fileID);
    temp_A = sscanf(tline, '%d');
    vertex_new_partiton(k) = temp_A + 1;   % rack start for 1
end
fclose(fileID);

% write the 'sub-alloc' file
FILE_NAME = 'sub_alloc';
fileID = fopen(FILE_NAME, 'w');

for k = 1: BUS_NUMBER

    % k: 히든 모선번호
    m = map_old_to_new (k);

    if (m ==0)
        disp('Bug 6');
        return;
    end

    n = vertex_new_partiton(m);
    o = hid2real_Map(k);

    fprintf(fileID, '%d         %d ¥n', o, n);

end
```

```
fclose(fileID);

% extra part
% generators -- > terminal transformer been merged
% slack generator (i.e., a generator directly connected to HV bus or no
% UTRF is excluded

FILE_NAME = 'sub_alloc';
fileID = fopen(FILE_NAME, 'a+');

for k = 1: merged_generator_count
    % merged_generators 의 모선 인덱스는 리얼
    i = merged_generators(k,1);
    m = merged_generators(k,2);

    for kk=1:BUS_NUMBER
        if (real_bus_number(kk) == m)
            break;
        end
    end

    kk = real2hid_Map(m);

    if (kk ==0)
        disp('Bug 7')';
        return;
    end

    n = vertex_new_partiton(map_old_to_new(kk));

    fprintf(fileID, '%d         %d ¥n', i, n);
end

fclose(fileID);

disp('sub_alloc file written');
```