Performance evaluation for Kruskal's and Prim's Algorithm in Minimum Spanning Tree using Networkx Package and Matplotlib to visualizing the MST Result

Article · May 2019

CITATION
1

READS
5,805

1 author:

Im Rafid
University of Indonesia
3 PUBLICATIONS **1** CITATION

SEE PROFILE

Performance evaluation for Kruskal's and Prim's Algorithm in Minimum Spanning Tree using Networkx Package and Matplotlib to visualizing the MST Result

Ilham Mulya Rafid
Department of Electrical Engineering
Universitas Indonesia
Depok, Indonesia
Email : ilham.mulya@ui.ac.id

***Abstract*—Minimum Spanning Tree is well known computational methodology that calculated the sum of all edges in the spanning tree which can be solved in a given time Algorithm. Widely the algorithms that are implemented that being used are Kruskal's Algorithm and Prim's Algorithm. Both of them have different time complexity, computational methodology and resource consumption.**

Keywords—Minimum Spanning Tree, Kruskal Algorithm, Prim's Algorithm, Networkx Package, Algorithm Evaluation.

I. INTRODUCTION

Recently, many of IoT (Internet of Things) Devices is developed using Microcontroller such as ARM, MCU with Arduino Bootloader, etc that have a limited amount resources such as registers, clock speed, bus, electricity source, etc. This problem will lead a developer to choose the most efficient algorithm, computational methodology, and resource management in order to get the optimum solution for their devices.

The minimum spanning tree is playing an important role in this field. For example, it will determine the best route for every vertex to connect between them based on the edge that is possible. It has direct application especially in the design of the networks, it used the algorithm such as Prim's algorithm and Kruskal's algorithm in order to approximate the travel salesman problem, multi-terminal minimum cut problem, and minimum cost – weighted perfect matching, Cluster Analysis, Handwriting recognition, Image segmentation, etc.

This experiment will compare both algorithms using Networkx, a python package for studying data structure and network connection analysis also Matplotlib for visualizing and plotting the Minimum Spanning Tree Result. This two package will use a pretty huge amount of resources such as CPU and Memory Consumption, so it is only for study and analysis purpose.

This paper is divided into 4 Main Parts. Chapter I will cover a brief introduction to this experiment based paper. Chapter II will cover a Theoretical Basis for the Experiment. Chapter III will cover an experiment methodology and result also the result analysis from the experiment, and Chapter IV will cover an Experiment Conclusion.

II. THEORETICAL BASIS

A. Algorithm Complexity

As in the Algorithm Theory that has been verified, Kruskal Algorithm and Prim's Algorithm having different time complexity and computational methodology. The difference of time complexity between them can be described like this table.

Table 1

Kruskal's and Prim's Algorithm Time Complexity

Algorithm	Time Complexity
Kruskal's Algorithm	O(E Log E or E Log V)
Prim's Algorithm	O(E log V) using binary Heap / O(E + V log V) using Fibonacci Heap

The program that is used in this experiment will only use the binary heap for Prim's Algorithm. So, the Prim's algorithm theoretically having a smaller growth rate of algorithm time complexity.

B. Sample Algorithm Description

Kruskal's Algorithm is one of the minimum spanning tree algorithm that based on greedy methods algorithm approach which finds an edge of the least possible weight that connects between two trees in the forest by finding a subset of the edges that forms a tree, includes every vertex where the total weight of all edges in the tree is minimized.

The main steps of this Algorithm are:

1. Create a Forest (a set of trees), where each vertex is on a separate tree.
2. Create a Set containing all of the edges in the graph.
3. Creating a recursion function to remove an edge with minimum weight from the set and combining two trees into a single tree and add it into Forest if the removed edges connecting two different trees while the Set is nonempty and Forest is not yet spanning.

The Pseudocode for this algorithm can be described like this

Algorithm 1 : Kruskal's Algorithm

```
Kruskal ( G ):
1:  I = ∅
2:  foreach v ∈ G.V:
3:      MAKE-SET(v)
4:  foreach (u, v) in G.E ordered by weight(u, v),
    increasing:
5:      if FIND-SET(u) ≠ FIND-SET(v):
6:          I = I ∪ {(u, v)}
7:          UNION(FIND-SET(u), FIND-SET(v))
8:  return I
```

Prim's Algorithm is one of the minimum spanning tree algorithm that based on greedy methods algorithm approach which finds a subset of the edges that forms a tree that includes every vertex, where the total weight of all edges in the tree is in minimum value by constructing the tree in one vertex at a time, from arbitrary starting vertex, and adding the minimum value of edges in possible connection from the tree to another vertex for each step.

The main steps of this Algorithm are:

1. Initialize a tree with a single vertex. (Chosen arbitrarily from the graph)
2. Growing the tree using a single edge that connects the tree to the vertices that not yet listed in the forest, then find the minimum – weight for every edge and then transfer it into the main tree.
3. Recursively repeat the steps 2 until all the vertex is connected through the edges.

The Pseudocode for this Algorithm can be described like this

Algorithm 2: Prim's Algorithm

```
1:  prims( adjacencyList, vertices, startVertex, MST )
2:    current = startVertex
3:      for i = 0 to vertices
4:        visited[i] = false
5:      i = 0
6:  while i < vertices
7:        if  !visited[current]
8:            visited[current] = true
9:            temp = adjacencyList[current]
10:
11:           while temp != NULL
12:                 var.u = current
13:                 var.v = temp -> val
14:                 var.weight = temp -> weight
15:                 PriorityQueue.enqueue(var)
16:                 temp = temp -> next
17:           var = PriorityQueue.extractMin();
18:           newVertex = var.v
19:           current = var.u
20:
21:           if !visited[newVertex]
22:               MST[current] =
    addEdge(MST[current], newVertex, var.weight)
23:               MST[newVertex] =
    addEdge(MST[newVertex], current, var.weight)
24:           current = newVertex
25:           ++i
26:     else
27:           var = PriorityQueue.extractMin();
28:           newVertex = var.v
29:           current = var.u
30:           if !visited[newVertex]
31:               MST[current] = addEdge(MST[current],
    newVertex, var.weight)
32:               MST[newVertex] =
    addEdge(MST[newVertex], current, var.weight)
33:               current = newVertex
```

C. *The Integrity of the Experiment*

The data for this experiment can manually be inputted by the user in .txt files and certain formats. So, the user can check it manually for the result. The Minimum Spanning Tree is also visualized using Matplotlib and Networkx so, the user can determine themselves whether the output is relevant or not.

III. The Experiment and The Analysis

It is important to install Python 3.x with Networkx and Matplotlib Package to run this Program. The Networkx is used for simplifying the analysis and visualization for network communication, and Matplotlib is used to Visualized and Plotting the final Graph. To check the CPU and Memory Consumption, it checked using Windows Task Manager so, the actual needed for processing can be seen and for the Running Time, it is checked by program execution time.

A. *Tools for Experiment*

This experiment is run on Asus A455L Notebook Series with Intel(R) Core(TM) i5-5200U CPU @ 2.20 GHz (4 CPUs), Memory : 8192 Mb DDR3L RAM, Intel (R) HD Graphics 5500 Dedicated Graphic Card, NVIDIA Geforce 930m Integrated Graphic Card, Samsung 860 EVO SATA SSD (with Windows 10 Education Operating System) and 500 Gb Seagate Barracuda Slim SATA HDD.

B. *Experiment Data*

The sampling data is inputted by the user through the .txt file such as the vertex name, the possible connection between vertex, and the weight for every edge. The Set of Data that is used in this experiment can be seen in table 2, table 3 and table 4.

Table 2

Set Data 1

Main Vertex	Connected Vertex	Weight Edges
A	B	3
	C	4
	E	5
B	C	3
	D	4

C	D	3
D	E	4

Table 3

Set Data 2

Main Vertex	Connected Vertex	Weight Edges
A	D	10
	E	5
B	C	6
C	A	7
	D	12
D	B	2

Table 4

Set Data 3

Main Vertex	Connected Vertex	Weight Edges
A	C	1
B	C	2
	D	3
	E	4
C	E	5
D	E	6
E	A	7

C. Experiment Result

The Result for Prim's Algorithm of this experiment :

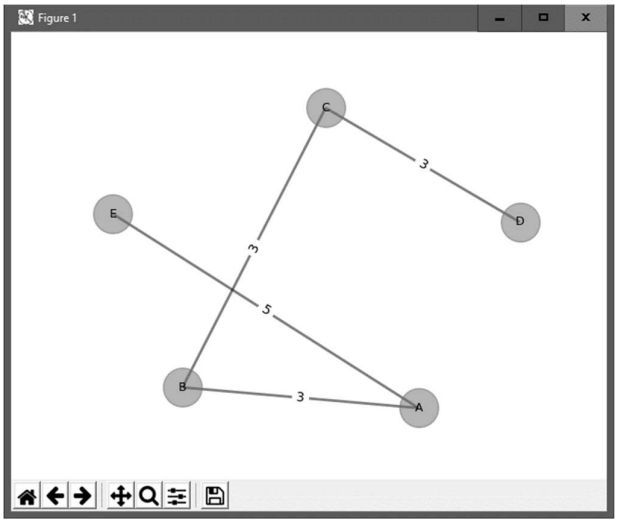


Figure 1.0 Example of Program graph output from Set Data 1.

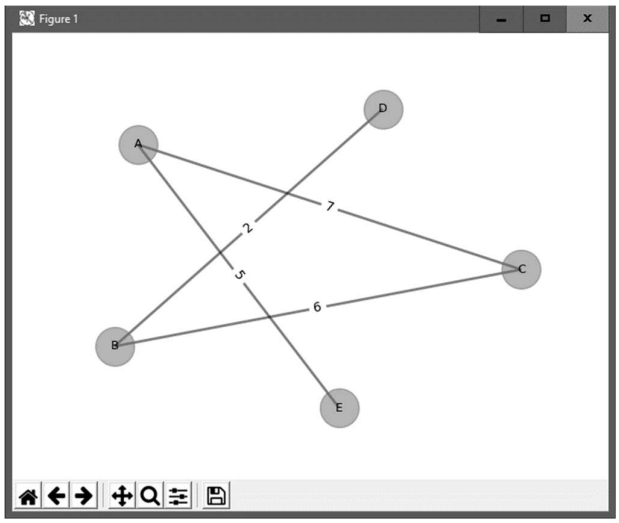


Figure 1.1 Example of Program graph output from Set Data 2.

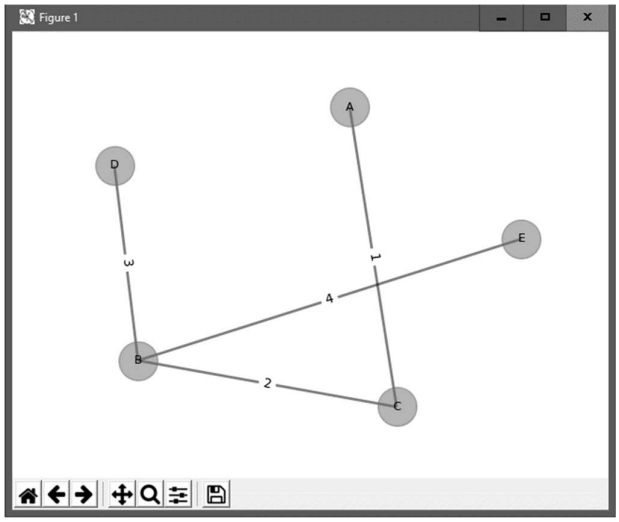


Figure 1.2 Example of Program graph output from Set Data 3.

The Result for the total weight of every connected edge in MST Graph is shown in the table below.

Table 5

Total Weight of the Edges from MST Graph using Prim's Algorithm

Set Data	Number of Experiment	Total Weight for MST
1	1	13
	2	14
	3	Error
2	1	Error
	2	17

	3	20
3	1	10
	2	Error
	3	10

Resource Consumption for every data set in Prim's Algorithm Experiment is shown in the graph below.

Table 7

Resource Consumption for Prim's Algorithm

Set Data	Number of Experiment	CPU Consumption	Memory Consumption	Running Time
1	1	±16%	±1%	±1,72s
	2	±18%	±1%	±2,37s
	3	±19%	±1%	±2,59s
2	1	±20%	±1%	±2,91s
	2	±16%	±1%	±1,68s
	3	±18%	±1%	±2,63s
3	1	±18%	±1%	±1,61s
	2	±19%	±1%	±2,05s
	3	±17%	±1%	±1,84s

Notes that this Result is computed on the computer with specification that already mentioned before in chapter III subchapter a. To find the exact amount of that, scale it into that specification.

The Result for Kruskal's Algorithm of this experiment :

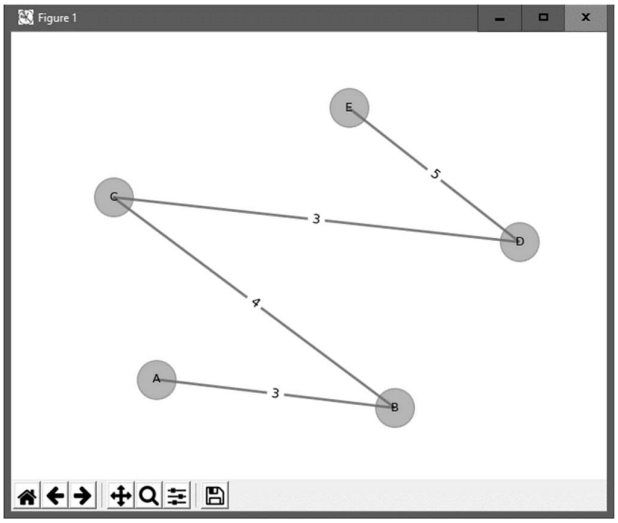


Figure 1.4 Example of Program graph output from Set Data 1.

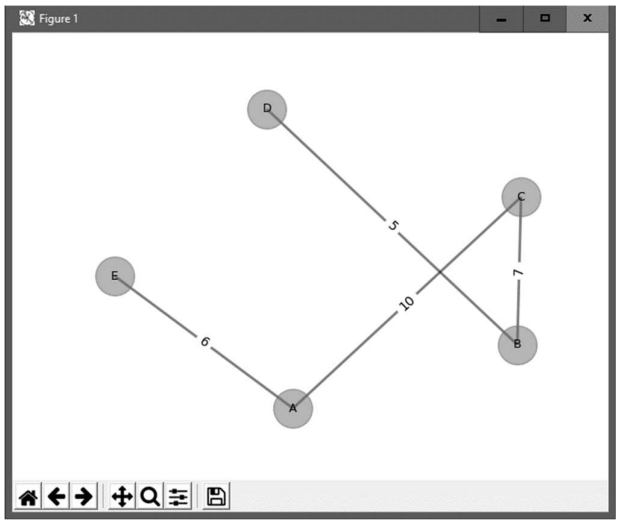


Figure 1.5 Example of Program graph output from Set Data 2.

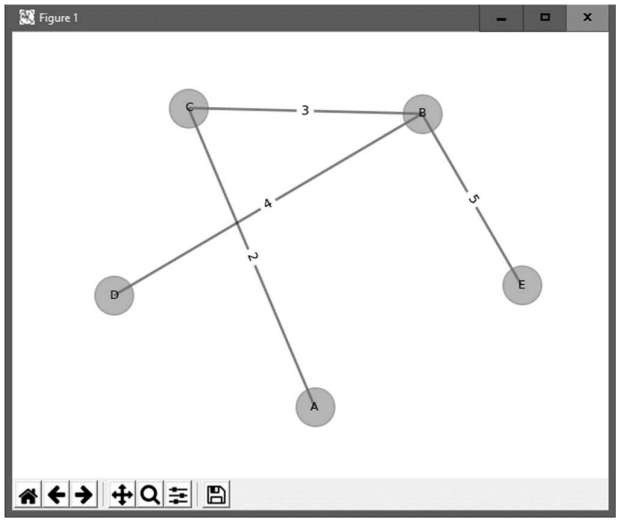


Figure 1.6 Example of Program graph output from Set Data 3.

The Result for the total weight of every connected edge in MST Graph is shown in the table below.

Table 8

Total Weight of the Edges from MST Graph using Kruskal's Algorithm

Set Data	Number of Experiment	Total Weight for MST
1	1	15
	2	15
	3	15

2	1	28
	2	28
	3	28
3	1	14
	2	14
	3	14

Resource Consumption for every data set in Prim's Algorithm Experiment is shown in the graph below.

Table 9

Total Weight of the Edges from MST Graph using Kruskal's Algorithm

Set Data	Number of Experiment	CPU Consumption	Memory Consumption	Running Time
1	1	±15%	±1%	±1,39s
	2	±16%	±1%	±1,15s
	3	±14%	±1%	±1,48s
2	1	±14%	±1%	±1,43s
	2	±15%	±1%	±1,27s
	3	±16%	±1%	±1,19s
3	1	±14%	±1%	±1,52s
	2	±14%	±1%	±1,57s
	3	±16%	±1%	±1,16s

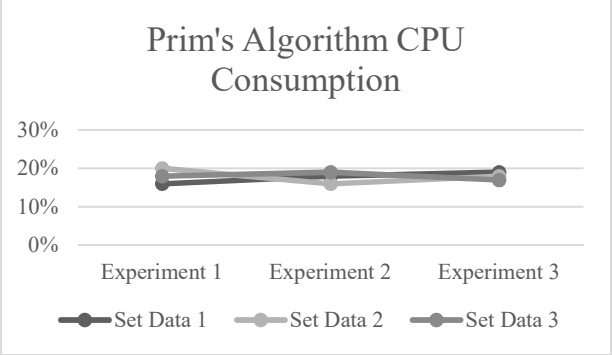
Notes that this Result is computed on a computer with the specification that already mentioned before in chapter III subchapter a. To find the exact amount of that, scale it into that specification.

D. Result Analysis

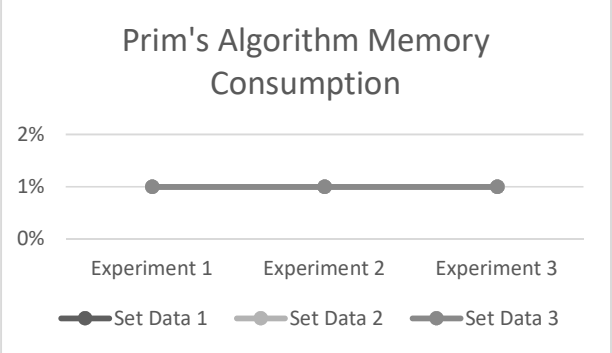
As a result of the experiment shown above, for Prim's Algorithm, there is an error output for each experiment in the variant dataset. And also, the total weight result for Prim's and Kruskal's Algorithm showing different results. This condition might occur because of the Prim's algorithm stability and methodology is different from the Kruskal's algorithm. This problem is still being discussed between computer scientist.

The Result above is also showing that Kruskal's algorithm having better stability than Prim's algorithm. It because correspond to the Algorithm Theory which Kruskal's algorithm overall is having growth rate rather than Prim's algorithm.

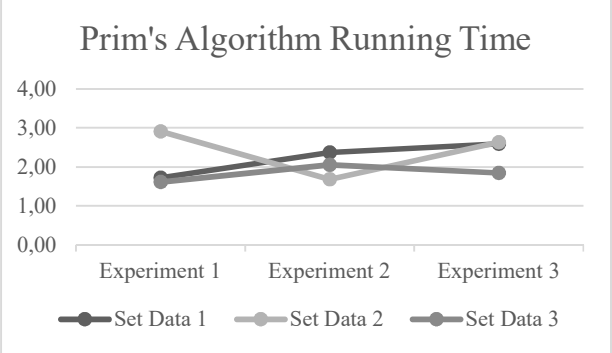
The result from the experiment using Prim's Algorithm can be plotted in Graph likes below.



Graph 1.0 Prim's Algorithm CPU Consumption

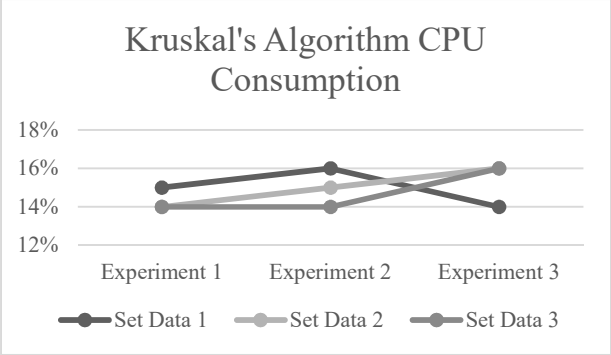


Graph 1.1 Prim's Algorithm Memory Consumption

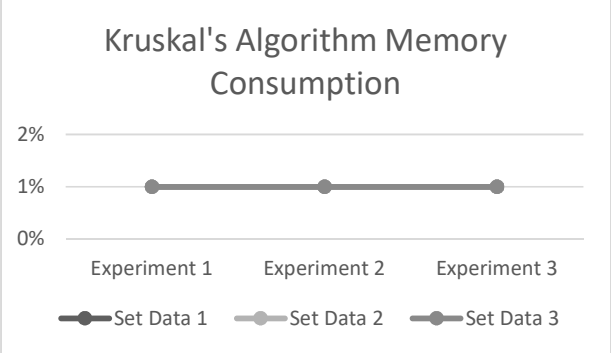


Graph 1.2 Prim's Algorithm Running Time

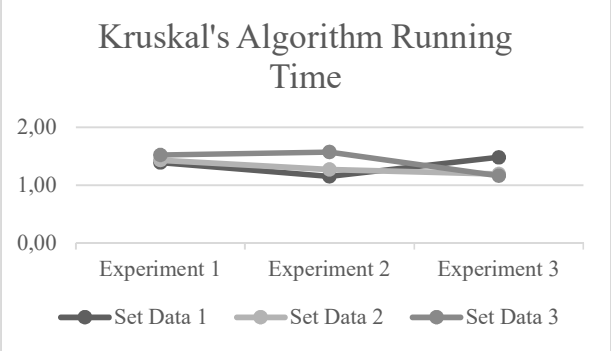
The result from experiment using Kruskal's Algorithm can be plotted in Graph likes below.



Graph 2.1 Kruskal's Algorithm CPU Consumption



Graph 2.2 Kruskal's Algorithm Memory Consumption



Graph 2.2 Kruskal's Algorithm Running Time

From that plotting Graph, we can see that the memory consumption for each algorithm is same and stable whether the CPU Consumption is having inverse linearity with the Running, the more CPU Consumption, the less Running Time execution.

IV. Conclusion Statement

From this Experiment, we can conclude that The Kruskal's Algorithm is having better stability than Prim's Algorithm this is based on the Time Complexity Theory to determine algorithm Growth Rate. This theory also shows that the Greater Time Complexity, the more Resource Consumption needed. Also, the more resources consumption used the faster running time for the Algorithm.

Prim's Algorithm is having worse stability than Kruskal's Algorithm, but Prim's Algorithm can find the total weight of the MST accurately rather than Kruskal's Algorithm.

Both of this Algorithm is having their own advantages and disadvantages so, to choose which is best to implement it is based on the developer/application requirements, whether it requires the less resource consumption or the accuracy from the data.

Acknowledgment

For more information about the program that is used in this experiment, you can visit the author repository at https://github.com/WolfDroid/MST_Prim-s_and_Kruskal-s.

References

[1] Gilles Brassard, Paul Bratley, "Algorithms: Theory and Practice", Prentice Hall Professional Technical Reference, 1988.

[2] Thomas H. Cormen, "Introduction to Algorithms", 3rd Edition, MIT Press, 2009.

[3] Robert Sedgewick & Kevin Wayne, "Algorithms", 4th Ed., Addison-Wesley Professional, 2011.

[4] G. Eason, B. Noble, and I. N. Sneddon, "COMPUTATIONAL METHODS FOR MINIMUM SPANNING TREE ALGORITHMS," Phil. Trans. Roy. Soc. London, vol. A247, pp. 529–551, April 1955. *(references)*

[5] H. J. Greenberg, Greedy Algorithms for Minimum Spanning Tree. Denver, 1998.

[6] B. Munier, M. Aleem, M. Islam, M. Iqbal and W. Mehmood, "A Fast Implementation of Minimum Spanning Tree Method and Applying it to Kruskal's and Prim's Algorithms", Sukkur IBA Journal of Computing and Mathematical Sciences, vol. 1, no. 1, p. 58, 2017. Available: 10.30537/sjcms.v1i1.8 [Accessed 26 May 2019].

[7] R. Graham and P. Hell, "On the History of the Minimum Spanning Tree Problem", IEEE Annals of the History of Computing, vol. 7, no. 1, pp. 43-57, 1985. Available: 10.1109/mahc.1985.10011 [Accessed 26 May 2019].R. Nicole, "Title of paper with only first word capitalized," J. Name Stand. Abbrev., in press.

[8] G. Santi and L. De Laurentis, Evaluating a sublinear-time algorithm for the Minimum Spanning Tree Weight problem. 2017.