The Reduction of Directed Cyclic Graph for Task Assignment Problem

W.N.M. Ariffin[1]

[1]Institute of Engineering Mathematics, Universiti Malaysia Perlis.

Abstract. In this paper, a directed cyclic graph (DCG) is proposed as the task graph. It is undesirable and impossible to complete the task according to the constraints if the cycle exists. Therefore, an effort should be done in order to eliminate the cycle to obtain a directed acyclic graph (DAG), so that the minimum amount of time required for the entire task can be found. The technique of reducing the complexity of the directed cyclic graph to a directed acyclic graph by reversing the orientation of the path is the main contribution of this study. The algorithm was coded using Java programming and consistently produced good assignment and task schedule.

1 Introduction

Task assignment is one of the most challenging problems in distributed computing environment. An optimal task assignment guarantees minimum turnaround time for a given architecture. Several approaches and techniques of optimal task assignment have been proposed by various researchers ranging from graph partitioning based tools to heuristic graph matching. A good task assignment algorithm and mapping strategy ensure turnaround time minimization. Thus, many researchers have attempted to produce good algorithm to solve the task assignment problem. Previous work put efforts on solving a directed acyclic task graph. But, how about directed cyclic graph ?

Cordella et al. [1] proposed a new algorithm which can handle subgraph isomorphism test efficiently in large graphs. Tong et al. [2,3] proposed an algorithm to find best effort matching in a large graph based on random walk. Another type of work that similar to Linhong et al. which use index to find matches in a large data graph efficiently, Tian and Patel [4] utilize degree information and neighborhood connectivity to filter unmatched node pairs. Zhang et al. [5] proposed another indexing approach based on graph distance. Their methods are not efficient enough to handle graphs with up to millions or billions of nodes and edges. In the preprocessing stage of [5], given a data graph, they generate a set of intersecting subgraphs may increase significantly when the size of data graph grows. Tian et al. [4] used a maximum weighted bipartite graph matching algorithm during the stage of matching important nodes, which can be costly if the bipartite graph is large.

Gutman and Wagner [6] defined the matching energy of a graph and gave some properties and asymptotic results of the matching energy. Shuli and Weigen [7] characterized the connected graph G with connectivity k has the maximum matching energy by introducing n-matching n-partite graph. Mohan and Gupta [8] established a methodology for heuristic graph matching, but restricted to a small number of test cases.

As the number and complexities of graph based applications increase, rendering the graphs more compact, easier to understand navigate through are becoming crucial tasks. Another one approach to graph simplification is to partition the graph into smaller parts, so that instead of the whole graph, the partitions and their inter-connections need to be considered. Common approaches to graph partitioning involve identifying sets of edges (edge-cuts) or vertices (vertex-cuts) whose removal partitions the graph into the target number of disconnected components. While edge-cuts result in partitions that are vertex disjoint,in vertex-cuts the data vertices can serve as bridges between the resulting data partitions; consequently, vertex-cut based approaches are especially suitable when the vetices on the vertex-cut will be replicated on all relevant partitions. A significant challenge in vertex-cut based partitioning, however, is ensuring the balance of the resulting partitions while simultaneously minimizing the number of vertices that are cut (and thus replicated).

Ariffin and Salleh [9] put an effort to reduce the graph onto directed acyclic graph. A Kernighan-Lin algorithm is applied to obtain the partition of tasks. Combining the technique of reduction and partitioning lead to an efficient graph-mapping concept. In [10], considered a directed-weighted cyclic task graph. Combining the technique of reduction and bi-partitioning led to an efficient graph-mapping concept.

* Corresponding author: munirah@unimap.edu.my

In this study, a directed cyclic graph (DCG) is proposed as the task graph. It is undesirable and impossible to complete the task according to the constraints if the cycle exists. Therefore, an effort should be done in order to eliminate the cycle to obtain a directed acyclic graph (DAG), so that the minimum amount of time required for the entire task can be found. The main contribution in this study is the problem itself. The previous work did not solve for directed cyclic graph for task assignment problem. The technique of reducing the complexity of the directed cyclic graph to a directed acyclic graph is also the contribution of this study.

2 Problem Statement

An arbitrary task graph, with eight number of nodes is proposed as in Figure 1. A cycle exists for {4,5,7,4} .

How the cycle can be eliminated from the graph to form an acyclic graph?

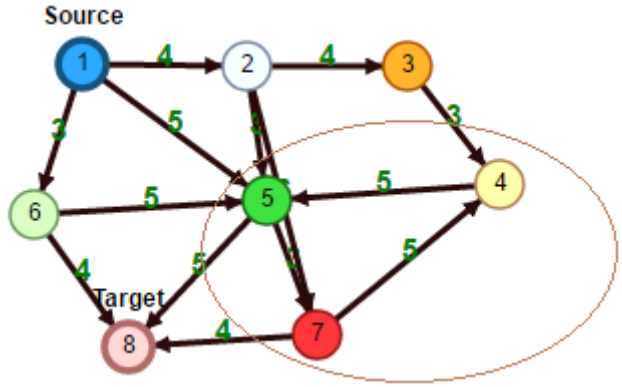


Fig. 1: A DCG with eight number of nodes.

From Figure 1 above, let consider the three nodes that form the cycle {4,5,7,4}.

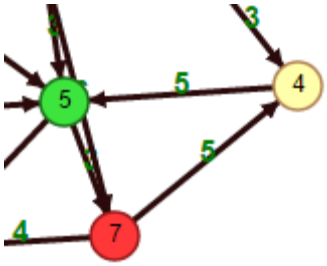


Fig. 2: A cycle in {4,5,7,4}.

The main objective is to eliminate the cycle in the task graph, so that it becomes acyclic. One way to do this is simply drop edges from the task graph to break the cycle. A feedback arc set or feedback edge set is a set of edges which when removed from the graph will leave a DAG. Put another way, it is a set containing at least one edge of every cycle in the graph. Closely related are the feedback vertex set, which is a set of vertices containing at least one vertex from every cycle in the directed graph and the minimum spanning tree, which is the undirected variant feedback arc set problem.

A minimal feedback arc set where one that cannot be reduced in size by eliminating or removing any edges has the additional property that, if the edges in it are reversed rather than removed, then the graph remains acyclic.

Sometimes, it is desirable to drop as few edges as possible, obtaining a minimum feedback arc set or dually a maximum acyclic subgraph. This is a hard computational problem, for which several approximate solutions have been devised.

3 The Decomposition of Strongly Connected Component

A graph $G=(V,E)$ has set of nodes $V=\{1,2,3,...n\}$ and an edge set E is a collection of unordered pairs of nodes called the edges of the graph. An orientation or directed graph $\vec{G}$ is obtained by assigning an orientation (ordering) to each edge of G. Each edge has a tail and a head. It is oriented from tail to head in the directed graph. A directed graph might have multiple directed edges between the same two nodes, y and z. This is called directed multi-graphs.

A cycle of a graph G is a subset of the sedge set of G that forms a path such that the start node and the end node are the same. A directed cycle is an oriented cycle such that all directed edges are oriented in the same direction along the cycle. An acyclic directed graph does not contain any directed cycle. The degree of a node y is the number of edges hat have y as an end node. The out-degree of a given node in a directed graph is the number of edges directed out of the node. A node with out-degree 0 is called a sink (however, in this paper the term *sink* is replaced by *target*).

A directed graph is called strongly connected if and only if any two nodes i and j in $\vec{G}$, there exists a directed path from i to j to and from j to i. The strongly connected components of a graph are its maximal strongly connected sub graphs. It is maximal in the sense that a sub graph cannot be enlarged to another strongly connected sub graph by including additional nodes and its associated edges.

4 Equivalent of the Orientation

In this part, choices in orientations of edges which conserve a fixed out-degree of each of the vertices do not alter the decompositions. This is a useful observation for the proofs in the next section and will assist the verification of the algorithm to generate the directed graphs and related decompositions by A. Sljoka [11].

Definition 4.1.

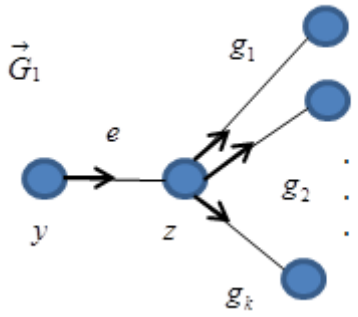
Given a multi-graph G and two orientations $\vec{G}_1$ and $\vec{G}_2$ are equivalent orientations on G if the corresponding nodes have the same out-degree.

Lemma 4.1.

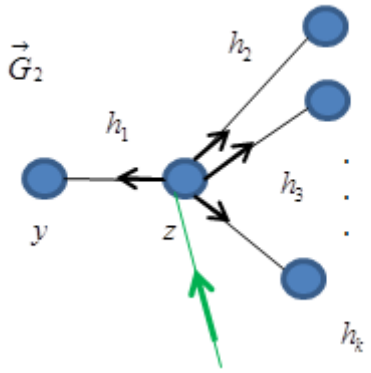
Given two equivalent orientations $\vec{G}_1$ and $\vec{G}_2$ on G, then the two orientations differ by reversals on a set of directed cycles.

Proof 4.1.

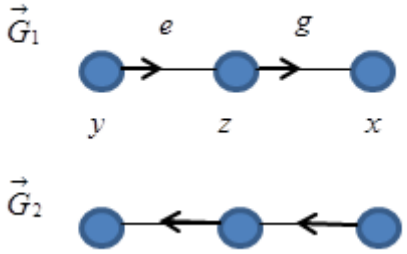
Choose an edge $e=(y,z)$ in $\vec{G}_1$ that is oppositely directed in $\vec{G}_2$. Thus, in $\vec{G}_1$ edge e is incoming at node z. Assume that there are k outgoing edges at z in $\vec{G}_1$. Because z has same out-degree k in $\vec{G}_2$, and this edge is reversed to be outgoing in $\vec{G}_2$, there must exist an out-going edge from z in $\vec{G}_1$, say $g=(z,x)$, that is oppositely directed in $\vec{G}_2$. This situation is illustrated in Figure 3 (a) and (b). Walk out of z along g in $\vec{G}_1$ is shown in Figure 3 (c).



(a)



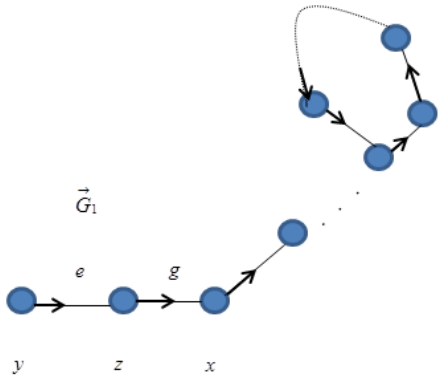
(b)



(c)

Fig. 3: Node y has same out-degree in G_1 and G_2. Thus, there exists an edge in G_1 (a) that is oppositely oriented in G_2 (b). The selection of f is shown in (c).

As a new node is inserted, the same argument is applied. This identies a directed path in $\vec{G}_1$ that is oppositely directed in $\vec{G}_2$. As there is only a finite collection of edges that have opposite direction, walking along this directed path until coming back to the same node on this path, identifying a directed cycle in $\vec{G}_1$ that has an opposite orientation in $\vec{G}_2$. Reversing the orientation of such a cycle from the orientation in $\vec{G}_1$ towards the orientation in $\vec{G}_2$. This decreases the number of edges in $\vec{G}_1$ that are oppositely directed from $\vec{G}_2$. The illustration can be seen in Figure 4. Reversing identified cycles is continued until all edges are directed following $\vec{G}_2$.



(a)

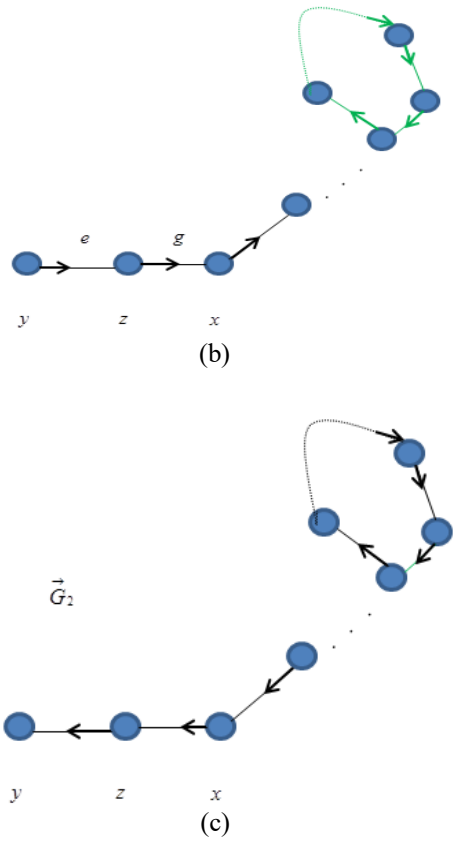


Fig. 4: Locating a directed path in G_1 that is oppositely oriented in G_2 (a), reversing a first cycle in G_1 (b), which now has the same orientation as in G_2 (c).

A valuable corollary to the lemma:

Corollary 4.1.

Given two equivalent orientations $\vec{G}_1$ and $\vec{G}_2$, the strongly connected components of the decompositions are the same.

Proof.

From Lemma 4.1, $\vec{G}_1$ and $\vec{G}_2$ differ by reversal of set of directed cycles. As cycle reversals do not change the strongly connected components, the decompositions are the same.

5 Proposed Technique to the Directed Cyclic Graph

The discussion in Section 4 is applied to solve the problem in Figure 2. The nodes in Section 4 (v_y, v_z, v_x) are renamed as v_4, v_5, v_7 which indicate node 4, node 5 and node 7 respectively. Note that node v_7 is inserted to form $\vec{G}_2$.

In this study, the technique of removing cycle is presented by reversing the direction of at least for one directed edge. The algorithm is explained as follows:

Step 1: Fix the priority of the tasks (nodes), label them from 1 to n.

Step 2: The task graph contains edges (y, z), where $y < z$.

Step 3: Choose one node as the starting point (parent) with the property that having the least computational time from the previous pointed edge(s).

Step 4: Decompose the nodes with two different edges and share the same child.

Step 5: Recombine the two decomposition in Step 4, and change the direction of the edge.

Step 6: Check again the task graph from node 1 to node n. If cycle exists, repeat Step 1 to Step 5. Else, stop the iteration.

As can be seen in Figure 2, the node {4,5,7,4} consists of a cycle. One can also say that [5,7,4,5] and {7,4,5,7} create the cycle. But, according to the algorithm in Step 3, one node with the least amount of computational time from the previous node is chosen as the starting point, called parent. This is important because a minimum amount of time to complete the task is required. Therefore, from Figure 1, node 3 is directed to node 4 with computational time of three units. Node 5 is directed from node 1, 2, 4, and 6 with bigger amount of computational time, compared to the computational time to node 4. The last one is node 7 is directed from node 2, which is obviously with bigger computational time as it is directed to node 5 previously. Therefore, node 4 is chosen as the parent node, so that the cycle is read as {4,5,7,4}. Then, the nodes are decomposed as shown in Figure 5.

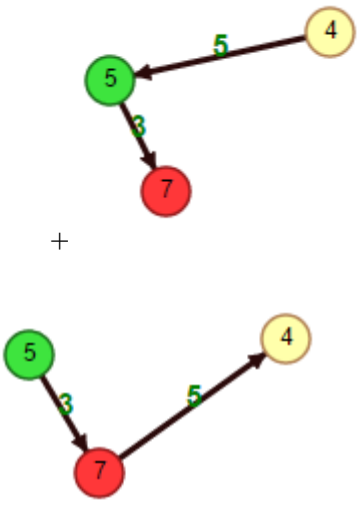


Fig. 5: Decomposition of the three nodes to two set of directed graphs.

The parent node is node 4 is directed to node 5. Therefore, node 5 becomes it child. Next, node 5 is directed out to the node 7. As stated Step 4, with the same child node, it is directed to node 7 and finally back to node 4. Since node 5 consists of directed edge from node 4 and to node 7, therefore the direction remains unchange. Thus, the direction from node 7 to node 4 is changed or reversed, so that it becomes from node 4 to node 7. This can be illustrated as in Figure 6.

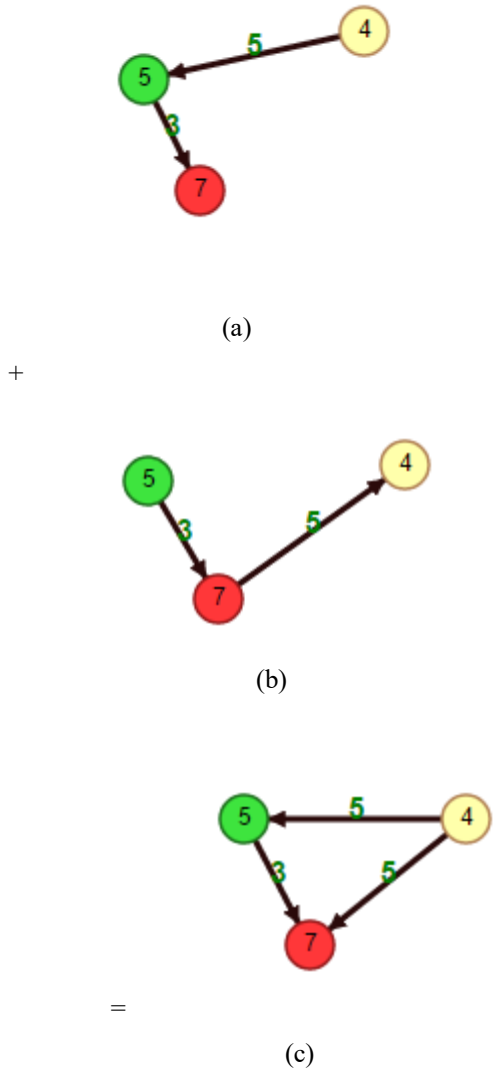


Fig. 6: Decomposition of the three nodes to two set of directed graphs, (a) and (b). Combination of (a) and (b), (c).

6 Formation of Directed Acyclic Graph

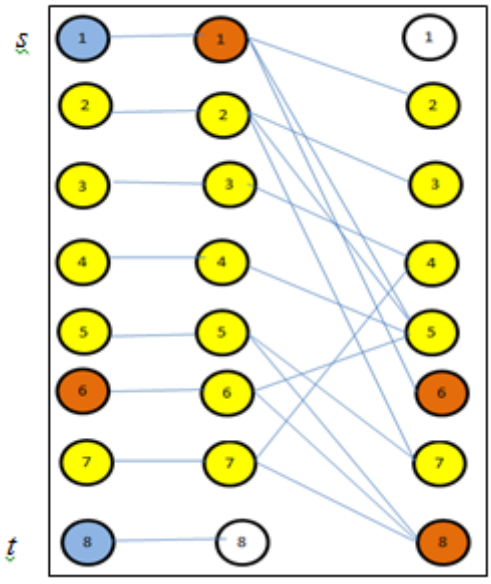
The proposed task graph in Figure 1 is now becomes a directed acyclic graph with the reverse direction from node 4 to node 7. Back to the task assignment problem, the nodes represent the beginning and the end of the tasks. Each task takes a certain estimated time to complete, and this is represented by assigning each edge uv a weight $WT(uv)$, being the amount of time required for that particular task.

6.1 Topological Order

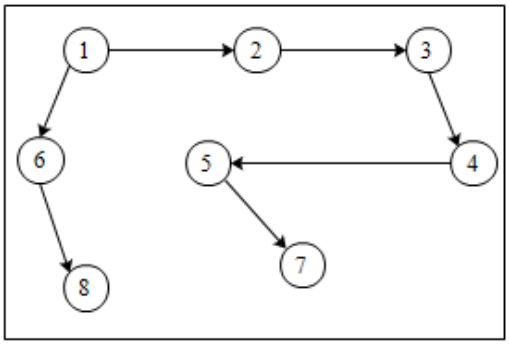
A topological ordering of an acyclic digraph G is a permutation of $V(G)=\{1,2,3,...,n\}$ such that $o(u) < o(v)$ whenever $u \rightarrow v$. Thus all edges are directed from smaller to higher node numbers. Notice that only acyclic digraph have topological orderings, since a directed cycle cannot be ordered in this way. Topological orderings are easy to find.

The algorithm is easy to be implemented. It is called a *DCGSimplify*. The topological order is built on the queue. The algorithm begins by placing all vertices with in-degree 0 on the queue. All nodes are arranging on two columns like wrapped-butterfly, inspired by Liao [16]. A source-destination (target) nodes are chosen from the graph. $InDegree[v]$ is then adjusted so that it counts the in-degree of v only from vertices not yet on the queue. This is done by decrementing $InDegree[v]$ according to its in-edges from the queue. When all n nodes are on the queue, the nodes are in the topological order. Note that in order to find the path from the source to the target node, the algorithm does not allow sharing common intermediate nodes and less amount of time on the edge is chosen.

Figure 7 (a) shows the arrangement of the task graph in two columns, while Figure 7 (b) shows its directed acyclic graph based on the proposed algorithm as discussed in Section 5.



(a)



(b)

Fig. 7: All n nodes are arranged in two-column (a) and the directed acyclic graph.

In this problem, node 1 is set as the source node and node 8 as the destination node. The target is to obtain the minimum amount of time to complete the task. This can be refered to find the shortest path from the source node 1, denote as s to the destination node 8, denote as t. So, the path from s to t must be found such that all nodes must be chosen and they must not share common intermediate nodes.

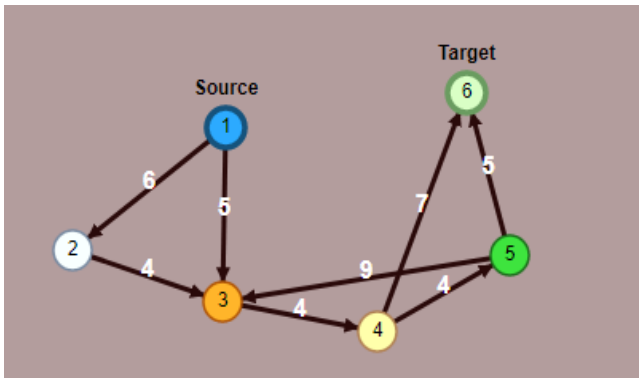
As can be seen from Figure 7 (a), there are three nodes can be chosen from the source node 1. Let say, node 2 is chosen from node 1. Next, from node 2, node 3, 5 and 7 can be reached.

Next, node 3 is chosen. From node 3, only node 4 can be chosen. Next, node 5 is chosen from node 4. After that, node 7 is chosen from node 5. Note that there are two nodes that are not chosed yet. So, again from the source node, node 6 is chosen. From node 6, either node 5 or node 8 can be chosen. Notice that, same intermediate nodes are not allowed. Thus, node 5 is strictly not available to be chosed. Next, from node 6, node 8 can be reached, which is the destination node. Now, the directed acyclic graph is obtained. It is illustrated as in Figure 7 (b).

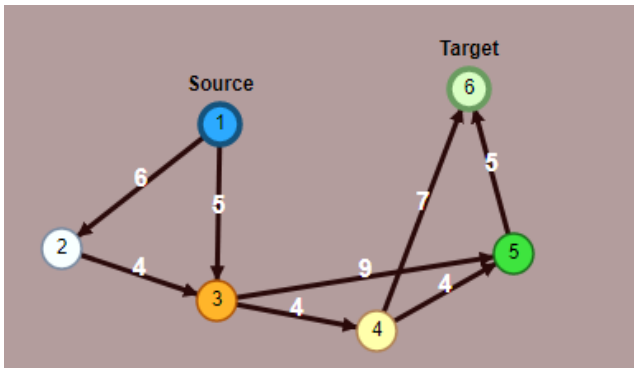
7 Results

To evaluate our proposed algorithm, we have implemented it using Intel(R) Core(TM) i5 (2.3 GHz) using Javascript programming language with D3 library. The algorithm is applied to test for several number of nodes.

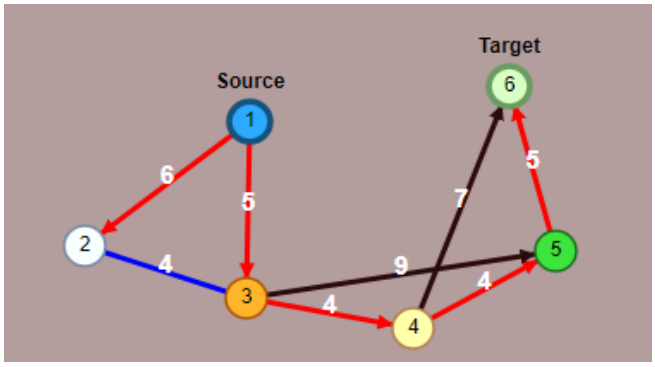
The task graph with small number of nodes (six nodes) is shown in Figure 8 below.



(a)



(b)



(c)

Fig. 8: Figure 8 (a) The directed cyclic graph, 8 (b) removing the cycle and 8 (c) the directed acyclic graph to be mapped on the processors.

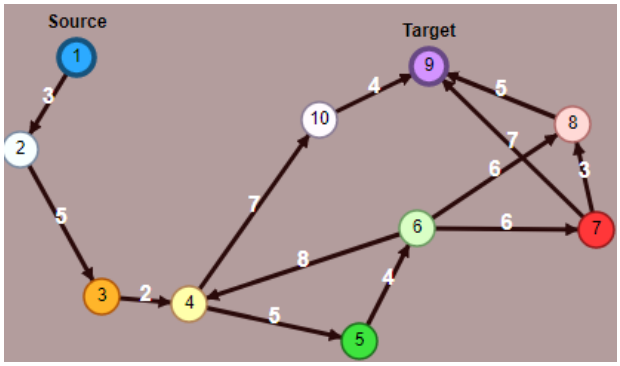
The task graph in Figure 8 (a) is the directed cyclic graph while in (b) after applying the proposed algorithm. Figure 8 (c) shows the final nodes (tasks) that need to be mapped onto processors. This is shown in Figure 9.

P1	2	1		
P2	3	4	5	6

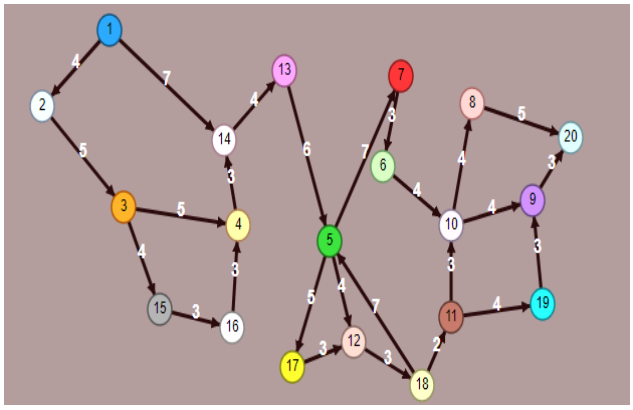
Fig. 9: Mapping the nodes onto two processors.

From Figure 9 above, node 1 and node 2 are located in processor one, while node 3, node 4, node 5 and node 6 are located in processor two. The total schedule length is 18 units.

The other two testing have been done on ten and twenty number of nodes. The task graphs are shown in Figure 10 (a) and (b).



(a)



(b)

Fig. 10: The task graph of directed cyclic graph for (a) ten and (b) twenty number of nodes.

The results are scheduled in Figure 11 below.

P1	10	1	4	7	8
P2	9	2	3	5	6

(a)

P1	12	19	6	8	5	2	3	4	13	15	18
P2	17	9	10	20	7	1	11	14	16		

(b)

Fig. 11: Task schedule of (a) ten nodes and (b) twenty nodes.

The algorithm is found easy to be implemented and consistently mapped the task onto prcessors and produce good schedule length.

In the next paper, we will extend the research on load balancing problem.

This work was supported by the Universiti Malaysia Perlis for Short Term Grant under project code 9001-00529.

References

[1] L.P. Cordella, P. Foggia, C. Samsone, M. Vento, *A (Sub)Graph Isomorphism Algorithm for Matching Large Graphs*, IEEE Transaction on Pattern Analysis and Machine Intelligence 26 (10) (2004), 1367-1372.

[2] H. Tong, C. Faloutsos, B. Callagher, T. Eliassi-Rad, *Fast Best-Effort Pattern Matching in Large Attributed Graphs,* Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM, New York, NY, USA, (2007), 737-746.

[3] H. Tong, C. Faloutsos, *Center-piece Subgraphs: Problem Definition and Fast Solutions*, Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and data Mining, ACM, New York, NY, USA, (2006), 404-413.

[4] Y. Tian, J.M. Patel, Tale: *A Tool for Approximate Large Graph Matching*. Proceedings of the 24th International Conference on Data Engineering, IEEE Computer Society, Washington, DC, USA, (2008), 963-972.

[5] S.Zhang, S. Li, J. Yang, Gaddi, *Distance Index Based Subgraph Matching in Biological Net- works* Proceedings of the 12th International Conference on Extending Database Technology ACM, New York, NY, USA, (2009), 192-203.

[6] I. Gutman, S. Wegner, *The Matching Energy of a Graph*, Discrete Appl. Math. 160 (2012), 2177-2187.

[7] L. Shuli, Y. Weigen, *The Matching Energy of Graphs with Given Parameters*, Discrete Applied Mathematics 162 (2013), 415-420.

[8] R. Mohan, A. Gupta, *Graph Matching Algorithm for Task assignment Problem*, International Journal of Computer Science, Engineering and Applications (IJCSEA) (2011), Vol. 1, No. 6.

[9] W.N.M. Ariffin and S. Salleh, *The Partitioning Technique Of Directed Cyclic Graph For Task Assignment Problem*, AIP Conference Proceedings 1750, 020010 (2016); doi: 10.1063/1.4954523

[10] W. N. M. Ariffin and S. Salleh, *Bi-Partition Approach Of Directed Cyclic Task Graph Onto Multicolumn Processors For Total Completion Time Minimization Task Assignment Problem*, AIP Conference Proceedings 1775, 030072 (2016); doi: 10.1063/1.4965192.

[11] A. Sljoka, O. Shai, W. Whiteley, *Checking mobility and Decomposition of Linkages Via Pebble Game Algorithm*, in; ASME 2011 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference, Washington, USA, August 28-31, 2011.