Kurdistan Journal of Applied Research (KJAR) | Print-ISSN: 2411-7684 – Electronic-ISSN: 2411-7706 | kjar.spu.edu.iq
Volume 2 | Issue 3 | August 2017 | DOI: 10.24017/science.2017.3.15

Optimization Algorithm's Problems: Comparison Study

Rebaz M. Nabi
Network Dept.
Technical College of informatics
Sulaimani Polytechnic University
Kurdistan Technical Institute
rebaz.nabi@spu.edu.iq; rebaz.nabi@kti.edu.krd

Rania Azad
Network Department
Computer Science Institute
Sulaimani Polytechnic University
Sulaimani, Iraq
Rania.azad@spu.edu.iq

Soran Saeed
Vice president
Sulaimani Polytechnic University
Sulaimani, Iraq
soran.saeed@spu.edu.iq

Rebwar M. Nabi
Deputy Dean
Kurdistan Technical Institute,
Sulaimani Polytechnic University
Sulaimani, Iraq
Rebwar.nabi@kti.edu.krd

Abstract:***Currently, in various fields and disciplines problem optimization are used commonly. In this concern, we have to define solutions which are two known concepts optimal or near optimal optimization problems in regards to some objects. Usually, it is surely difficult to sort problems out in only one step, but some processes can be followed by us which people usually call it problem solving. Frequently, the solution process is split into various steps which are accomplishing one after the other. Therefore, in this paper we consider some algorithms that help us to sort out problems, for exemplify, finding the shortest path, minimum spanning tree, maximum network flows and maximum matching. More importantly, the algorithm comparison will be presented. Additionally, the limitation of each algorithm. The last but not the least, the future research in this area will be approached.***

Keywords: Algorithm, problem solving, shortest path, minimum spanning tree, network flows.

1. INTRODUCTION

Computation algorithms take a primitive role in developing computer applications, which contributes greatly to their accuracy and efficiency. Therefore, optimization becomes one of the most important subjects that are considered when the applications are developed. However, it seems very difficult to give a good solution in dealing with some areas such as making a working timetable, producing a plan for production and routing on the network. It is also noticed that these problems can be represented as graphs [2] [10] [4] [13] [1]. Thus, this article aims to discuss some common algorithms which help to solve the combinatorial problems in terms of finding the shortest path, minimum spanning tree, maximum network flows and maximum matching.
In particular, the first part will introduce Dijkstra's algorithm for finding the shortest path in a graph. The second part will give the general idea of algorithms to minimize the spanning tree. The next part illustrates how the maximum network flow is determined by Fork-Fulkerson and Edmons-Karps algorithms. The maximum matching problem is discussed in the final part, which can be solved by Hopcroft-Karp and Edmonds algorithms.

1. Shortest Path Problem

The shortest path algorithm is defined for network/graphs whether directed, undirected, or both which solves the shortest path problem [11] [9] [10]. In order to find path between two vertices' problem the shortest path problem is used (source and destination nodes) the lowest cost path with their summation [4] [10], for example: Google map shows the shortest path between source and destination place.

Algorithms

- **Dijkstra's algorithm:** When edge weights are non-negative, Dijkstra's algorithm is used to overcome the single-source shortest path issues on a weighted, concentrating graph G = (V, E) [3].
- **Bellman-Fond algorithm:** On negative edge weights Bellman-Fond algorithm is used to solve the single-source problem [3]
- **Floyd's algorithm:** This algorithm all pairs shortest paths.

This paper only examines Dijkstra's algorithm for solving shortest path problem.

1.1. Dijkstra's Algorithm

Dijkstra's algorithm is a "graph based shortest path search algorithm and it solves the shortest path problem for a graph or network with non-negative edge costs, producing a shortest path tree" [12] [14]. This algorithm is used for generating routing table in the network.
In a graph, it traverses the unvisited nodes with the lowermost cost remoteness, computes the distance through it to each unused fellow node, and modernizes the neighbor's space if it is slighter [12].
For example, Figure 1 demonstrates a directed weighted graph G= (V,E) with non-negative weight **w** and source node **s**. The issue is to identify the shortest path from basis node to any end node with bottommost weight/cost in the

graph [14].

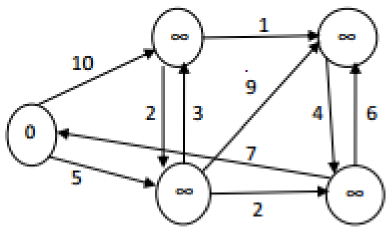


Figure 1 Weighted directed graph G=(V, E)

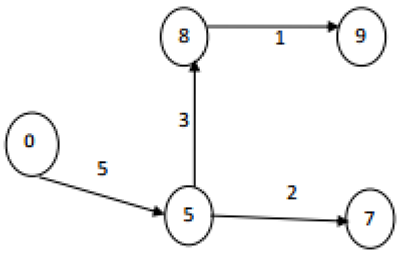


Figure 2 The shortest path for given graph

The solution is given in figure 2, which shows the shortest path from vertex 0 to others with the minimum weights.

Time complexity

Dijkstra's algorithm preserves the min-priority queue Q and classified into exactly three precedence queue operation, INSERT, EXTRACT-MIN, and DECREASE-KEY [3].

Dijkstra's algorithm's running time relies on min-priority queue Q. "In a graph G= (V, E) edges E and vertices V can be articulated as a meaning of │E│ and │V│. First, the min-priority queue takes the vertices being numbered 1 to │V│. We keep d[v] in array, each INSERT and DECREASE_KEY process takes O (1) time, and EXTRACTMIN job takes O(V) time, for a total running time of O(V2+E)=O(V2)"[3] .

2. Minimum Spanning Tree

The "Minimum Spanning Tree" (MST)is defined for finding a subtree spanning of all the nodes, whose total weight might be minimal [16] [11]. MST is also known as minimum connector, economy tree.

"A weighted graph is a graph where we associate with each edge a real number, called the weight. Thus, the Spanning tree of G is a subgraph T of G which is a tree that spans G. The weight of a spanning tree T is the sum of the lowest weights of its edges" [16].

There are two common algorithms to construct Minimal Spanning Tree, which are "Kruskal's algorithm" and "Prim's algorithm". The main concept of Kruskal's algorithms is as below:

- Category all the edges in cumulative weight/cost order.
- If we take the edges consequently, if the cycle has not been created by any node, and formerly add into the spanning tree. Otherwise, abandon it.
- We need to add n-1 edges in the minimal spanning tree.

Figure 3 illustrates a weighted undirected graphs G= (V, E) which comprises of n number of nodes and m the number of ends. The minimum spanning tree has cost at 33.

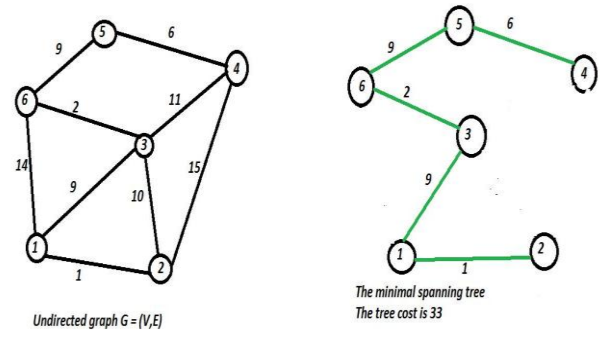


Figure 3 Example of Minimum spanning tree [11]

The operating time of "Kruskal's algorithm" works on the disconnect set tree data structure which each vertex holds a reference to its parent node. First, we assume the disjoint set A in line 1 spend "O (1)" time, and the time to category the ends in stroke 4 is O (E lg E). For instance, it needs to add for the price of the |V| group processes in the for loop 2-3 lines. The for-loop of lines 5-8 accomplishes O(E) FIND-SET and UNION actions on the disjoint-set. "Along with the |V| MAKE-SET operations, these take a total of O ((V + E) α (V)) time, where α is the very slowly growing function" [11]. Graph G is supposed to be associated, we have |E| ≥ |V| - 1, and therefore, the disjoint-set tasks use O (E α(V)) time. Subsequently α(|V|) = O (lg V) =O (lg E), the whole running time of "Kruskal's algorithm" is O (E lg E). Witnessing that |E| < |V|2, we have lg |E| = O (lg V), and thus we can repeat the running time of "Kruskal's algorithm" as O (E lg V) [3].

The "Minimal Spanning Tree" matter is to choose a number of edges so as to there is a path between each node [16] [17]. The edge lengths summation is to be reduced. In the meantime, the Shortest Path Tree problematic is to discover the group of edges linking all nodes in order to achieve the objective that the sum of the edge distances from the origin to each node is decreased [17] [11]. Additionally, shortest path tree depends on stating node but MST is not.

3. Maximum flow network

In the actual network, network nodes and edges have capacity limitations [18]. It needs to know how much traffic transmission is in a limited network capacity of two specified nodes (called up to between sources and sinks) in many cases, and determine to achieve the maximum flow transmission strategy. Maximum network flow

problem is the mathematical model to describe this problem. The maximum flow problem is an important part of the network flow theory, and it is a classic combinatorial optimization problem, but can also be seen as special linear programming problems [19]. In addition, to solve the problems in the real network, the maximum flow problem has a several uses in many areas of *engineering, the sciences of physics, chemistry, biology, management science and applied mathematics*. Therefore, the maximum flow problem is an important research content of computer science and operations research.
The research of maximum flow algorithm has 40 years of history. The earliest algorithm is network simplex method proposed by Dantzig in 1951 and upload rail algorithms increased by Ford and Fulkerson in1956 [4]. They are pseudo-polynomial time algorithm. A polynomial time algorithm began in 20 century 70's, respectively proposed by Dinic (1907, 1973), Edmonds and Karp (1970, 1972) [5] [4]. In 1973, Dinic got the time complexity of core factor algorithms for the first time. Decades after, the maximum flow algorithm obtained a lot of progress, and a lot of good algorithms were constantly being proposed.

3.1. Ford-Fulkerson Algorithm

The Ford-Fulkerson technique which is named for L.R. Ford, Jr. and D.R.Fulkerson, calculates the thoroughgoing movement in a flow network. It was issued in 1954 [20]. The name "Ford-Fulkerson" is frequently adopts by the "Edmonds-Karp" algorithm, which is narrowing down of "Ford-Fulkerson".

- **Augmenting path**

Is a path from the basis edge to sink point with the size is always greater than Zero which means that sending more flows via this path is possible?

- **Residual capacity**

The residual capacity of an edge is equal to the capacity minus the flows: "$c_f(u, v) = c(u, v) - f(u, v)$".

- **Residual network**

According to the flow *f*, the residual network is a graph of thenetwork in which use all the capacity of the path to subtract the minimum of the capacity on the path and then add or expand the ant direction of the capacity.

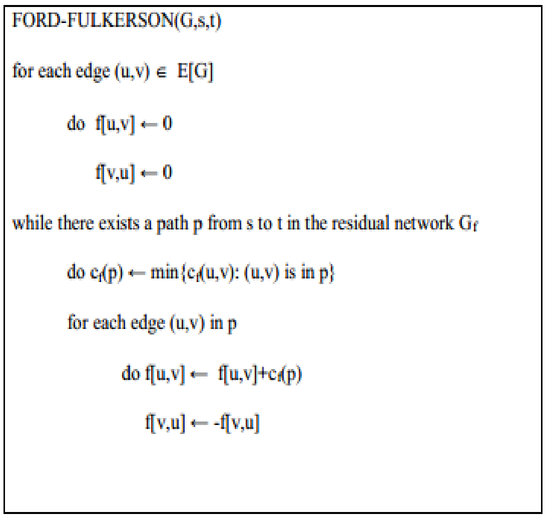
Algorithms

Based on the idea of Ford-Fulkerson method, there are several algorithms introduced to solve the maximum network flow problem.

The following part focuses on the basic Ford-Fulkerson algorithm and the extended one called Edmonds-Karp algorithm. These algorithms are based on the assumption that there is always at least a path in the graph direct G, which has set of edges E and vertexes V from vertex s to vertex t.

3.3.1The basic Ford-Fulkerson Algorithm

As defined in [11], the maximum flow f is determined by adding up with the residual capacity ($c_f(p)$) of each augmenting path p, which is the path from theoriginal vertex to the destination vertex in the residual graphs (G_f). This algorithm can be implemented as below:



```
FORD-FULKERSON(G,s,t)
for each edge (u,v) $\in$ E[G]
        do f[u,v] $\leftarrow$ 0
            f[v,u] $\leftarrow$ 0
while there exists a path p from s to t in the residual network $G_f$
        do $c_f(p) \leftarrow \min\{c_f(u,v): (u,v) \text{ is in } p\}$
        for each edge (u,v) in p
                do f[u,v] $\leftarrow$ f[u,v]+$c_f(p)$
                    f[v,u] $\leftarrow$ -f[v,u]
```

Figure 4 *Ford-Fulkerson(G,s,t)*

- Step 1: initialize the flow of each edge in original graph G with 0.
- Step 2: produce residual G_f from graph G based on the flows of edges.
- Step3: determine the augmenting path p in G_f
- If there is no augmenting path found, the algorithm is ended; otherwise, move to next step.
- Step 4: calculate the residual capacity of path p, which is the minimum of flows on this path.
- Step 5: increase the flows of all edges, which appear in path p, in G_f by the residual capacity got from step 4.
- Step 6: repeat step 3-5.

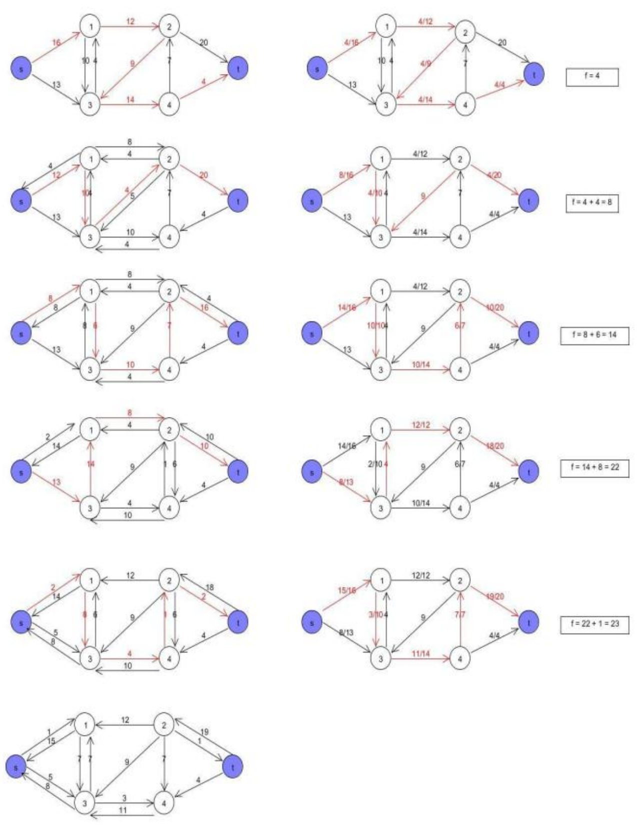


Figure 5 Example of maximum network flow using Fork-Fulkerson algorithm

The efficiency of Ford-Fulkerson algorithm is assessed in terms of time complexity [11] [20]. The time consuming of this algorithm is calculated by the sum of time-consuming in initiation flow values in step 1 and performing aloop in step 3 to 6. Firstly, the complexity is got from step 1 is E, the number of edges in graph G. Secondly, the outer while loop in step 3 can be run E time if all paths from **s**to **t** is one edge of G. Finally, in the worst case where the flow is increased by one for each time running step 5, the number of times isup to the maximum flow value. Therefore, the complexity of this algorithm is O(E|f|). Figure 6 shows that the maximum flow is 2N, which is increased by 1 that causes the complexity is 2N.

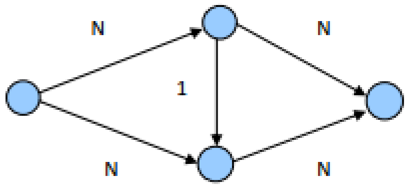


Figure 6: The maximum flow is 2N in which N is a capacity of each edge

Besides, the complexity also depends on how the augmenting path is chosen [11]. Choosing the arbitrary path can lead to un-stopped running ofthealgorithm in which the maximum flow keeps increasing. In addition, the algorithm also cannot terminate if the capacity is anirrational number [6]. Therefore, solving the selection problem will not only help to ensure the correctness of the Ford-Fulkerson method but also improve the efficiency of algorithms.

3.2. Edmons-Karp Algorithm

Introduced by Edmons and Karp (1972), the Edmons-Karp algorithm improves the Ford-Fulkerson algorithm by replacing choosing thearbitrary augmenting path by the shortest one [6]. The Edmons-Karp algorithm takes advantage of breadth-depth examine to investigate a path from s to t that has the minimum edges [11] [21]. He proves that in the graph which all edges' capacity is a unit, if the shortest path is selected, it is obviously that the distance of next shortest path will not be less than one of the previous path. Furthermore, because after augmenting flow, there is at least one edge (u,v), which has capacity is equal to the residual capacity of this path, is removed from the next residual graph. Thus, the distance of u can be up to |V|-2. Accordingly, the complexity of Edmons-Karp procedure is O(E.V. E) = O(VE^2).

4. Maximum matching

Let G= (V,E) be directionless graph. The graph G is matching with a subcategory of edges M ⊆ E when no two edges in M have the same vertex. In other words no two edges in M can be adjacent to each other. A vertex *v* is coordinated by M when it is connected by an edge E to another vertex, otherwise it is unmatched. The edges that connect a pair of vertices is said to be a part of matching.

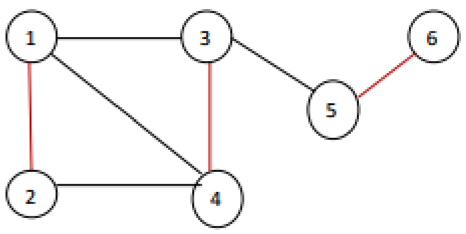


Figure 7 The edges in red (1 →2, 3→4,5→6)form matching

- **Maximum Matching**

A maximum matching is achieved when the matching has themaximum size or the maximum cardinality. A graph can have one or more matching of maximum cardinality.

For example, in figure 7, there are six vertices. When they are paired, we should get 3 edges. From the figure, we get 3 edges, which apparently make it a maximum matching.

- **Maximal Matching**

When no more possible matching to vertex*v*can be attained, thematching is said to be maximal.

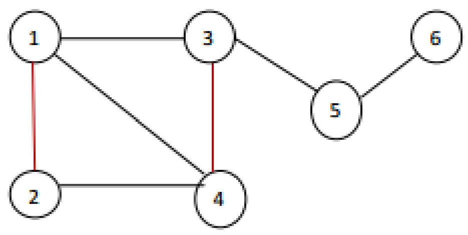


***Figure 8** The edges in red form matching.*

Here the vertices 5 and 6 could not be pairedwithout violating the matching property. Hence in this case, only a maximal matching can be attained as shown in the fig.

Perfect Matching
When all the vertices of the Graph G are matched, it complete orperfect matching.

For instance, in figure 7, all the vertices are connected by an edge to another. Such a matching is a perfect matching.

Alternating Path
Let G = (V, E) be a graph. A path P can be an alternating pathwith respect to M(matching) if and only if among every two consecutive edges along the path, exactly one belongs to M. An alternating cycle C is defined similarly.
In other words, an alternating path is an alternative way in which the edges (E) connect to the matching (M).
Augmenting Path
An augmenting path P with respect to a matching M is analternating path that starts and ends in unmatched vertices.
Theorem (Berge'57)
Let M be a matching in the graph G = (V,E). Then either M isa maximum cardinality matching or there exists an M-augmenting path.

4.1. Hopcroft-Karp Algorithm:
The algorithm finds the maximum matching for the given bipartite graph. This was found by John Hopcroft and Richard Karp in the year 1973[22][23][24][25]. Let G = (S; T; E) be a bipartite graph and let M be a matching in G. Instead of building the alternating trees one at a time, we can use breadth-first search to simultaneously build alternating trees from all unmatched vertices of S. This allows finding shortest augmenting paths. A maximal collection of shortest augmenting paths can also be found.
The Hopcroft-Karp algorithm uses the augmenting path technique and it works in phases [24]. In each phase, it constructs a highest collection of vertex-disjoint shortest augmenting paths and uses them to enhance the matching.
Though the algorithm uses augmenting path technique it does not search the path one by one, to reduce the runtime. Instead, it searches for many paths at the same time. The length of the path grows in each step. Searching for long disjoint paths will not take more time because there will not be many of them.
The algorithm finds a maximal set of shortest vertices disjoint augmenting paths with respect to M in O(|E|) time. O(|E|) running time is the result of the construction of the partial-DFS procedure that considers each vertex, and each edge only once. The visited vertices are removed, so the paths in P are disjoint.

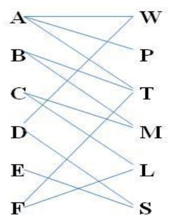
4.2. Edmonds Algorithm
To the problem of finding a maximum match in the non-bipartite graphs, we use blossom's algorithm. Jack Edmonds discovered the "Blossom algorithm" for finding maximum matches in graphs [26][27]. Given in a graph G=(V,E) the algorithm finds maximum matching. It reduces the odd length alternating cycles to a contracted graph and searches iteratively the contracted graph.
In a given graph G in which M is the matching and E is the edges. An M-alternating forest is a forest L such that, the root of each tree in L is a vertex from E and every tree doesnot contain other vertices from E and every vertex in E belongs to one tree in L and every edge at anodd distance from E belongs to M.
The Edmond's algorithm firstly grows the forest F by two edges, secondly if tow components are connected then increases the Matching M and contracts the graph/blossom.
These operations take O(m) times, hence the runtime of the blossom algorithm to find a maximum matching is given as ***O(n^2m).***

Applications
It is largely used in real world substantially in areas of image feature matching, Dual Processor Application. The following part will discuss a simple problem and how the solution is produced by using maximum matching algorithm.

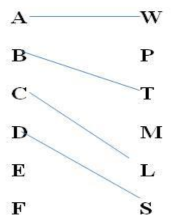
Problem: Students Ali (A), Baram (B), Canan (C), Dana(D), Eman (E),Fenk (F)should be assigned to the tasks Washing (W), Painting (P), Training (T), Mending (M), Lighting (L), Singing(s) as per their expertise

1. A □ W, P, T
2. B□T, M
3. C□L, M
4. D□S,W,L
5. E□S
6. F□L,T

Bipartite Graph for the problem



Initial Graph:



Initially, A, B, C,Dwill be assigned to the first job.

Figure 9 *Example of Maximum Matching Problem*

- **<u>Step 1:</u>**Start with the non-trivial initial matching
  A=W,B=T,C=L,D=S
- **<u>Step2:</u>**Find an alternating path
  (i)E-S=D-W=A-P (E=P)
  (ii)F-T=B-M (F=M)
- **<u>Step 3:</u>**When an alternating path is found list the path and change the status of the other arcs.If an alternating path cannot be found, we have reached the maximal matching
- **<u>Step 4:</u>**Rewrite the new matching with the changed arcs from the alternating path.
  E=S, D=W, A=P, S≠D, W≠T from (i)
  F=T, B=M, B≠T from (ii)

Hence the maximum matching is achieved.

- **<u>Step5:</u>**If the matching is complete stop, else repeat step 3

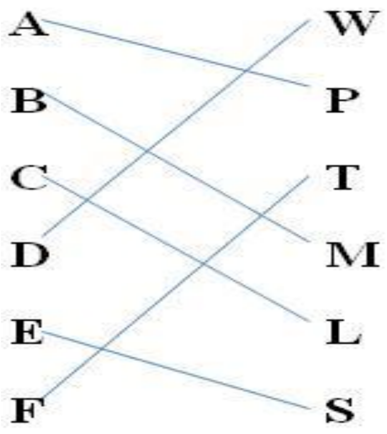


Figure 10 *The new bipartite graph after applying the theorem*

5. Algorithm Comparison

It evident that the concepts and mechanisms of algorithms are different which it may lead to different output and scenarios. Firstly, Dijkstra's algorithm is comparatively faster than Bellman-Ford. On the other hand, negative "distances" can be handled by Bellman Ford's algorithm which is not the case with Dijkstra's algorithm. Therefore, when the negative distances in the calculation is need to be considered, the Bellman-Ford algorithm should be used. Nevertheless, if negative distance is not required, the Dijkstra's algorithm is usually preferred especially for large networks. Outstandingly, both algorithms can have the same result when same topologies are used whereas, different outcomes will be produced when different topologies considered.

Additionally, there is difference between Dijkstra and Kruskal algorithm in which, Dijkstra's finds a connection between two vertices (shortest path between two vertices), while Kruskal's will find a connection between and a number of vertices (finding the minimum spanning tree between all the given vertices).

6. CONCLUSION

Combinatorial optimization emerged from more than 50 years ago, [6][15] [12] [7]. It not only helps to solve real-life problems but also tends to give efficient algorithms, especially in working with the large and complex data. In other words, algorithms for optimization have to produce accurate results and take less time as much as possible. Thus, these discussed algorithms can be seen as abasic approach for further improvement in solving four classical combinatorial problems, including shortest path, spanning tree, network flow, and matching.

Last but not least,it can be identified that there is a big room of improvement and researchin this area. So, further comparisons can be conducted experimentally to achieve better accurate results.

REFERENCE

[1]. R. Kümmerle, G. Grisetti, H. Strasdat, K. Konolige and W. Burgard, "G^{2}o: A general framework for graph optimization," *2011 IEEE International Conference on Robotics and Automation*, Shanghai, 2011, pp. 3607-3613.

[2]. G. S. Halford, R. Baker, J. E. McCredden, and J. D. Bain, “How Many Variables Can Humans Process?,” *Psychological Science*, vol. 16, no. 1, pp. 70–76, Jan. 2005.

[3]. X.-S. Yang, “Flower Pollination Algorithm for Global Optimization,” in *Unconventional Computation and Natural Computation*, 2012, pp. 240–249.

[4]. R. Ahuja, T. Magnanti, and J. Orlin, *Network Flows: Theory, Algorithms, and Applications*. Prentice Hall, 1993.

[5]. A. V. Goldberg, “Recent developments in maximum flow algorithms,” in *Algorithm Theory — SWAT’98*, 1998, pp. 1–10.

[6]. B. Korte and J. Vygen, *Combinatorial Optimization*, 1st ed. Berlin: Springer Berlin, 2014.

[7]. A. Alazzam and H. W., "A New Optimization Algorithm For Combinatorial Problems", *International Journal of Advanced Research in Artificial Intelligence*, vol. 2, no. 5, 2013.

[8]. X.-S. Yang, M. Karamanoglu, and X. He, “Multi-objective Flower Algorithm for Optimization,” *Procedia Computer Science*, vol. 18, pp. 861–868, Jan. 2013.

[9]. Xin-She Yang and Amir Hossein Gandomi, “Bat algorithm: a novel approach for global engineering optimization,” *Engineering Computations*, vol. 29, no. 5, pp. 464–483, Jul. 2012.

[10]. Y.-Q. Cheng, V. Wu, R. Collins, A. R. Hanson, and E. M. Riseman, “Maximum-weight bipartite matching technique and its application in image feature matching,” 1996, vol. 2727, pp. 453–462.

[11]. T. Cormen, C. Leiserson, R. Rivest and C. Stein, *Introduction to algorithms*, 1st ed. Cambridge, Massachusetts: The MIT Press, 2014.

[12]. T. Weise, R. Chiong, and K. Tang, “Evolutionary Optimization: Pitfalls and Booby Traps,” *J. Comput. Sci. Technol.*, vol. 27, no. 5, pp. 907–936, Sep. 2012

[13].T. Weise *et al.*, "Benchmarking Optimization Algorithms: An Open Source Framework for the Traveling Salesman Problem," in *IEEE Computational Intelligence Magazine*, vol. 9, no. 3, pp. 40-52, Aug. 2014.

[14]. T. Weise and K. Tang, "Evolving Distributed Algorithms With Genetic Programming," in *IEEE Transactions on Evolutionary Computation*, vol. 16, no. 2, pp. 242-265, April 2012.

[15]. T. Weise, M. Wan, P. Wang, K. Tang, A. Devert and X. Yao, "Frequency Fitness Assignment," in *IEEE Transactions on Evolutionary Computation*, vol. 18, no. 2, pp. 226-243, April 2014.

[16]. K. Weicker, *Evolutionary algorithms and dynamic optimization problems*, 1st ed. Osnabrück: Der AndereVerl., 2003.

[17]. A. Singh, “An artificial bee colony algorithm for the leaf-constrained minimum spanning tree problem,” *Applied Soft Computing*, vol. 9, no. 2, pp. 625–631, Mar. 2009.

[18]. F. Ortega and L. A. Wolsey, “A branch-and-cut algorithm for the single-commodity, uncapacitated, fixed-charge network flow problem,” *Networks*, vol. 41, no. 3, pp. 143–158, May 2003.

[19]. D. S. Hochbaum, “The Pseudoflow Algorithm: A New Algorithm for the Maximum-Flow Problem,” *Operations Research*, vol. 56, no. 4, pp. 992–1009, Aug. 2008.

[20]. J. M. Drake and D. M. Lodge, “Global hot spots of biological invasions: evaluating options for ballast–water management,” *Proceedings of the Royal Society of London B: Biological Sciences*, vol. 271, no. 1539, pp. 575–580, Mar. 2004.

[21]. V. Vineet and P. J. Narayanan, "CUDA cuts: Fast graph cuts on the GPU," *2008 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*, Anchorage, AK, 2008, pp. 1-8.

[22]. G. Menichetti, L. Dall’Asta, and G. Bianconi, “Network Controllability Is Determined by the Density of Low In-Degree and Out-Degree Nodes,” *Phys. Rev. Lett.*, vol. 113, no. 7, p. 078701, Aug. 2014.

[23]. S. Szeider, “Minimal unsatisfiable formulas with bounded clause-variable difference are fixed-parameter tractable,” *Journal of Computer and System Sciences*, vol. 69, no. 4, pp. 656–674, Dec. 2004.

[24]. C.-G. Quimper, A. López-Ortiz, P. van Beek, and A. Golynski, “Improved Algorithms for the Global Cardinality Constraint,” in *Principles and Practice of Constraint Programming – CP 2004*, 2004, pp. 542–556.

[25]. N. Blum, *A simplified realization of the Hopcroft Karp approach to maximum matching in general graphs*, 1st ed. Bonn: Inst. fürInformatik, 1999.

[26]. V. Kolmogorov, “Blossom V: a new implementation of a minimum cost perfect matching algorithm,” *Math. Prog. Comp.*, vol. 1, no. 1, pp. 43–67, Jul. 2009.

[27]. G. Bebek, V. Patel, and M. R. Chance, “PETALS: Proteomic Evaluation and Topological Analysis of a mutated Locus’ Signaling,” *BMC Bioinformatics*, vol. 11, p. 596, 2010.