2019 International Conference on Computing, Power and Communication Technologies (GUCON)
Galgotias University, Greater Noida, UP, India. Sep 27-28, 2019

An Efficient Approach to Solve Bus-Driver Assignment Problem

Mondira Chakraborty
Computer Science & Engineering
Jagannath University
Dhaka, Bangladesh
mondirachakraborty.jnu@gmail.com

Md Zahid Hassan
Computer Science & Engineering
Jagannath University
Dhaka, Bangladesh
zahidcode29@gmail.com

Md Masbaul Alam Polash
Computer Science & Engineering
Jagannath University
Dhaka, Bangladesh
mdmasbaul@gmail.com

***Abstract*—Assignment problems are very challenging combinatorial problem which are often used to compare the performance of various algorithms. In this paper, we are dealing with an interesting problem with respect to the transportation system of our country known as bus-driver assignment problem. The problem states that among different choices of the drivers, we have to assign drivers to the bus in the best possible way. The complexity of the problem is that the assignment must be the optimal one so that each bus has a driver and each driver drives a bus among their own choice. To solve this problem, we have used the well known Ford-Fulkerson and Edmond-karp algorithm. In our experimental analysis, we have compared the performance of these two algorithms and also shown the efficiency of these algorithms in solving such kinds of assignment problems.**

***Index Terms*—Assignment Problem, Edmond-karp Algorithm, Ford-Fulkerson Algorithm**

I. Introduction

Bus-driver assignment is an extremely critical part of operational planning process for the transportation system. The combinatorial nature and large size of this problem has created complexity which led to development of large number of models and techniques. This problem is a part of traditional assignment problem which is to find most possible match between driver and vehicles. Formally, Bus Driver Assignment Problem (BDAP) is defined as follows:

Definition 1 (Bus-Driver Assignment Problem)**.** *Bus Driver Assignment Problem (BDAP) is a process of assigning N number of drivers to M number of buses where $M \geq N$ in a way so that each bus has atmost one driver and each driver is assigned to a bus among their own choices.*

There are two different versions of this problem. First one is where each driver asks for a different amount of payment and the other one is where all the drivers are paid equally. In this paper, we have dealt with the later version only. Figure 1 shows a simplified, although quite general formulation of this problem. Here we consider a situation of a renowned transport company which have 4 vehicles, 5 drivers and each driver has preference of one or more number of buses. As we can see from Figure 1, driver1 wants to drive busA only, driver2 wants to drive either busA or busB, driver3 wants to drive either busC or busD, driver4 wants to drive busB only and driver5 wants to drive only busD.

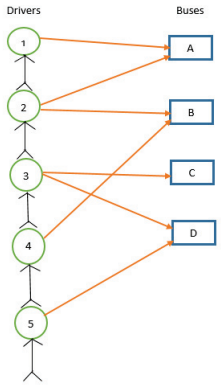
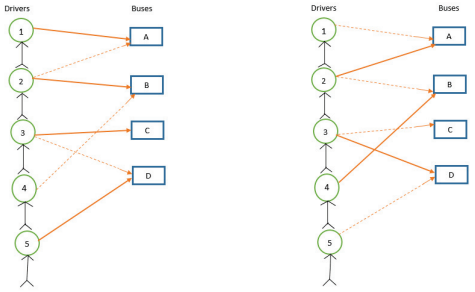


Fig. 1. Bus-driver route choice

We can assign the drivers to the buses of Figure 1 in many different ways. Among these, two possible assignments are shown in Figure 2. According to Figure 2(a), the assignment is driver2 $\rightarrow$ busA, driver3 $\rightarrow$ busD and driver4 $\rightarrow$ busB. In this way, we can assign drivers to 3 buses. On the other hand, Figure 2 shows an assignment of driver1 $\rightarrow$ busA, driver2 $\rightarrow$ busB, driver3 $\rightarrow$ busC and driver5 $\rightarrow$ busD. This assignment assigns a driver to all the buses. In this paper, our focus is to assign drivers to the most possible number of buses in shortest amount of time. We cannot provide more than one driver in each bus and there might be more than one best possible number of such assignments.

Over the years, many researchers has proposed different approaches to deal with problems related to the bus-driver assignment problem such as bus driver scheduling problem (BDSP) and Crew Rostering (CR) problem. One of the papers used mathematical formulation to solve the BDSP which is known as NP-Hard problem [1]. Another paper used Column Generation techniques combined with Genetic Algorithm [2]. Besides these approaches, some studies used Heuristic approaches like Hyper-Heuristic [3], Meta-Heuristic [4], Variable Neighborhood Search Heuristic [4], Hybrid Heuristic combined with Greedy Randomized Adaptive Search Procedure (GRASP) [5] etc. There are some other approaches namely Fuzzy Genetic [6], Unicost Set Covering Problem (SCP)

978-93-5351-098-5/19/$0.00@2019 Galgotias University



(a) (b)

Fig. 2. Different bus-driver assignments

combined with column Generation [7] which are used to solve the bus driver scheduling problem as well.

In this paper, we have used Edmond-karp algorithm and Ford-Fulkerson Algorithm to find the best assignment such that every bus has at-most one driver. Our focus is to find the maximum number of bus-driver matching from all possible assignments. We have also compared the performance of these two algorithms in handling such kind of assignment problem.

Rest of the paper is organized as follows: Section II gives a brief description of the necessary background on bus driver assignment problem. Section III presents the implementation details of our proposed approaches and Section IV outlines the comparative analysis of the two algorithms in terms of different performance indicators. Finally, section V concludes this paper with a discussion of future scope of this work.

II. Related Work

Bus Driver Assignment Problem (BDAP) is similar to Bus Driver Scheduling Problem (BDSP) or Crew Assignment Problem(CAP). The Classical Set Covering problem (SCP) and the Set Partition Problem (SPP) are mathematical solution for solving BDSP. Both of these problems are NP-hard problems and different techniques to handle these problems are extensively reported in the Operational Research Literature [1]. The basic idea of this approach is to create a large set of feasible duties called duties superset or shift-pool [7]. It uses a matrix where each row corresponds to a task and each column corresponds to a pre-compiled potential duty and a decision variable. Usually the total number of possible duties are very high [3], [7] which takes a long computational time to be built.

To solve the BDSP, few papers have reported the use of a combination of Column Generation (CG) with other approaches e.g., [8] one paper reports the use of CG combined with Genetic Algorithm(GA) to solve instances with up to 138 tasks [3] and it also outlines the use of a CG based hyper-heuristic to solve real instances with 500 tasks. [9] devised a genetic algorithm with new crossover/mutation mechanisms and were able to satisfy equal break distributions among drivers

A new mathematical formulation of BDSP under special constraints imposed by Italian transportation rules was presented instead of the SCP or the SPP [5]. Unfortunately, this formulation is only useful when applied to small or medium-sized instances (up to 136 tasks). Therefore, it is conventional to apply some heuristic techniques to reduce the number of the columns or to divide the problem into various sub-problems, solving each sub-problem separately (e.g. heuristic techniques to solve the SCP [6], [10]–[13]. Metaheuristic techniques are suitable for many optimization problems. The metaheuristic algorithms are exible, simple and scalable. However many metaheuristics algorithms working properly in continuous spaces [14]. It is not obvious how the continuous version can be applied to a discrete or binary problems. Exist dierent approach to convert a continuous metaheuristic in a discrete or binary algorithm. Examples of discrete adaptation are found using rounding o discretization method in Ant colony [15], Firey [16]. Another heuristic method called hybrid heuristic that combines Greedy Randomized Adaptive Search Procedure (GRASP) and Rollout meta-heuristics can solve instances up to 250 tasks.

An algorithm based on Iterated Local Search (ILS) [17] similar to Varible Neighborhood Search and Very Large-scale Neighborhood Search (VLNS); as well as a Self-Adjusting algorithm, based on evolutionary approach [18] are also used to tackle the BDSP problem. For larger instances, there are use of Greedy Randomized Adaptive Search Procedure (GRASP) and it can be applied to instances with at most 161 tasks [5], [19].

To solve BDSP, most of the papers found in the literature report SCP or SPP based methods. However, some studies use heuristics/meta-heuristics as well. For example: the HACS algorithm that uses a tabu search [20]; an evolutionary algorithm combined with fuzzy logic [6], and a heuristic method, called ZEST, derived from the process of manual scheduling [21]. We also found exact approaches based on the column generation (CG) technique to solve two real-world instances, both with up to 246 tasks instead of heuristic techniques. But the authors of this paper consider the unicost set covering where all columns have the same cost equal to one [3], [7], [8]. Later an improved CG algorithm was presented [7] which was applied to a set of real problem instances with up to 701 trips.

Besides, some papers solved these problems by using approaches that consider the scheduling of crews and the scheduling of vehicles simultaneously or integrally. An integration of CG and Lagrangian relaxation solves randomly generated instances and real-world data instances where number of tasks varies between 194 and 653 trips [2]; a set partition/covering-based approaches were applied to instances with up to 400 tasks [22]; a GRASP based algorithm were applied to real-world instances with up to 249 task [23] and an integrated approach to solve real-world instances with up to 653 trips [24]. A few approaches focus on the bus lines, i.e. the set of tasks are divided according to each bus line. Therefore, the problem instances size is clearly reduced and solved separately. However, some studies report an approach that focus on the network, i.e. the set of tasks is not divided according to each bus line and each task can be assigned to any driver despite considering bus line it belongs to.

TABLE I
RELATED WORK SUMMARY

Ref.	Objective Function	Approach	#tasks3	#duties3
[5]	MinCost,MinDrivers2	GRASP, VNS	161	44
[3]	MinCost+MinDrivers	Hiper-heuristic	500	145
[13]	MinCost	Heuristic+CG	500 [a]	153
[4]	MinCost	VNS Meta-heuristic	501	44
[6]	MinCost+MinDrivers	Fuzzy Genetic	613	75
[2]	MinCost	CG	< 653 [a,c]	117
[7]	MinDrivers	SC+CG	< 701 [c]	100

Table I shows the literature review of the work related to BDAP. It outlines the references, objective functions, approaches, tasks and duties of the previous work where

1) MinCost means minimizing the total cost
2) MinDrivers means minimizing the number of drivers
3) #tasks and #duties means the number of tasks and duties respectively

[a] set of artificial benchmark instances

[b] including vehicle changes and others functions are investigated

[c] number of trips

III. OUR APPROACH

As stated earlier, we have implemented two of the most well-known algorithms to handle these kinds of assignment problem named Edmond-karp algorithm and Ford-Fulkerson Algorithm. However, for the specific case of bus-driver assignment problem, we do not need the full generality of these algorithms. Since we have only considered the case where all the drivers are paid equally meaning the problem is converted into an unweighted directed graph, we can therefore use the following simplified algorithm which solves the problem for any general unweighted directed graph.

In this unweighted directed graph, we consider the drivers as left nodes and buses as right nodes. Let assume that M represents the number of drivers and N represents the number of buses where $M \geq N$. For the convenience of running our algorithm, we will create two more nodes. First one is termed as *source node* which is connected to all the drivers and all the buses are connected to another node called *Target node*. The process of converting the bus-driver assignment problem into an unweighted directed graph is shown in Figure 3.

In Figure 3 we assume that there are five drivers $(1, 2, 3, 4, 5)$ and four buses (a, b, c, d). There are also source node S and target node T. In this figure, driver1 wants to be assigned in busA, driver2 wants to be assigned either in busA or busB , driver3 wants to be assigned either in busC or busD, driver4 wants to be assigned in busB, and driver5 wants to be assigned either in busA or busD.

Both the algorithms starts from the source node and looks for a path to reach the target node. If it finds a path $p = (S, v_i, v_j \ldots T)$ from S to T, then the direction of the edges in that path p will be reversed. This process is depicted in Figure 4 and 5. Figure 4 shows that there exists a path from S to T. And the path is $S \rightarrow 1 \rightarrow a \rightarrow T$ which is shown in

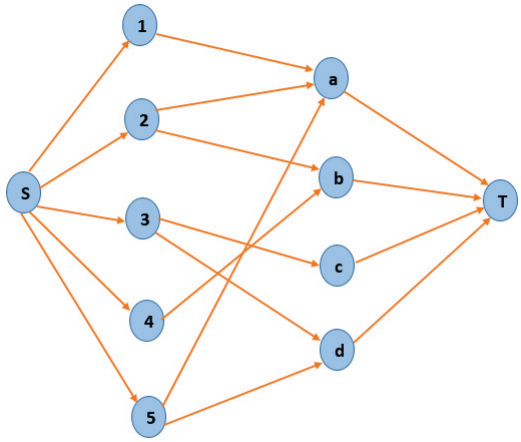


Fig. 3. Conversion of Bus-driver assignment problem into an unweighted directed graph

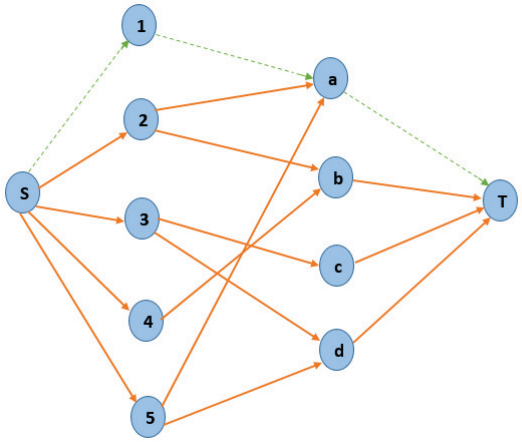


Fig. 4. Traverse graph

the figure with dotted lines. Figure 5 shows the updated graph with the direction of the edges reversed in the seleced path, p. In the updated graph, the direction $a \rightarrow T$ becomes $T \rightarrow a$, $1 \rightarrow a$ becomes $a \rightarrow 1$ and $S \rightarrow 1$ becomes $1 \rightarrow S$. This process continues as long as there exists a path from S to T.

Figure 6 shows the final graph in which there exist no other path from S to T. In this final graph, the path from bus to driver indicates which driver is assigned to which bus, i.e driver1 is assigned to busA, driver2 is assigned to busB, driver3 is assigned to busC and driver5 is assigned to busD.

Our approach starts with the idea of converting the bus-driver assignment problem into a bipartite matching problem. The entire process of finding the maximum matching in the bipartite graph is shown in Algorithm 1. First, we initialise all the necessary data structures (line 1) and then in lines 2-3, we create a source node S, a target nodde T and connect $S \rightarrow L$ where $L \in$ Drivers and $R \rightarrow T$ where $R \in$ Buses. After that,

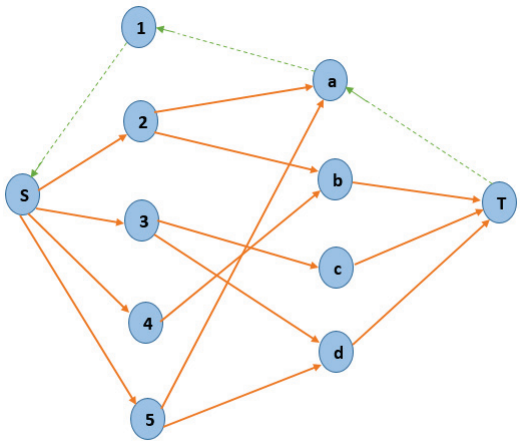


Fig. 5. Updated graph

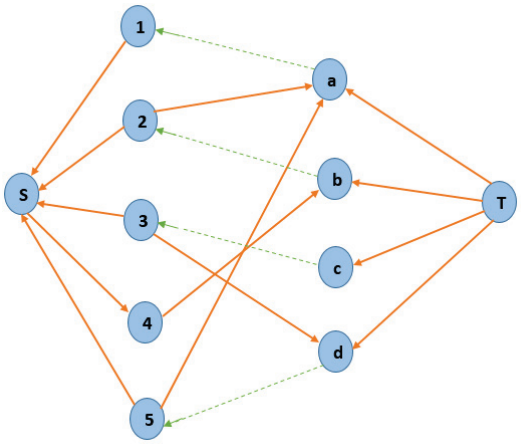


Fig. 6. Finally assigned graph

our algorithm iterates till there exists a path from S to T. In every iteration, it will find a path from S to T and update the graph accordingly. The two algorithms, named Edmond-karp algorithm and Ford-Fulkerson algorithm differ in the way of finding path. While Edmond-karp algorithm looks for the shortest path from S to T, Ford-Fulkerson algorithm chooses any path from S to T. For this reason, to implement these algorithms we have used two graph traversal algorithms named BFS and DFS. BFS is used in implementing Edmond-karp algorithm and DFS is used for Ford-Fulkerson algorithm. For better understanding, the implementation details of findPath() procedure is shown in Algorithm 2.

Algorithm 2 starts by initializing a variable *pathExists* (line 1) which indicates whether there exists a path from S to T. After that we insert the source node S into A (line 2). The data structure of A depends on the algorithm we are using. When we are implementing Edmond-karp algorithm A will be a queue and for implementing Ford-Fulkerson algorithm A will be a stack. Then we update the visited array for S to be

Algorithm 1: Main Procedure()

```
1. Initialization
2. create a source node S and a target node T
3. connect S → L ∈ Drivers and R ∈ Buses → T
4. while ( findPath( ) )
5.     updateGraph( )
6. print the assignment
```

true (line 3) and iterate our algorithm until all the nodes are visited (line 4). Then we remove the top element, u from A and move to the adjacent unvisited nodes of u (lines 6-28). Whenever the adjacent of u contains the target node T means that we have found a path from S to T, so turn the *pathExists* to true (lines 19-23). The procedure finally returns whether there exists path from S to T (line 31).

Algorithm 2: Procedure findPath()

```
1.  pathExists = false
2.  Insert(A, S)
3.  visited[S] = true
4.  while (A ≠ empty )
5.      u = RemoveTop(A)
6.      if ( u == S )
7.          for each left node i ∈ adj[u]
8.              if ( !visited[i] )
9.                  origin[i] = u
10.                 visited[i] = true
11.                 Insert(A, i)
12.     else if ( u ∈ Drivers )
13.         for each right node i ∈ adj[u]
14.             if ( !visited[i] )
15.                 origin[i] = u
16.                 visited[i] = true
17.                 Insert(A, i)
18.     else if ( u ∈ Buses)
19.         if (T ∈ adj[u])
20.             origin[T] = u
21.             visited[T] = true
22.             pathExists = true
23.             break
24.         for each left node i ∈ adj[u]
25.             if ( !visited[i] )
26.                 origin[i] = u
27.                 visited[i] = true
28.                 Insert(A, i)
29.     else
30.         break
31. return pathExists
```

After finding path in line 4 of Algorithm 1, we update the graph (line 5 in Algorithm 1) by reversing the direction of the edges along the selected path. After updating the graph

we try to find another path in the updated graph. This process continues until it finds all the possible paths from S to T. If all paths are found, we find best possible matches from right nodes to left nodes which means a particular driver is assigned to a particular bus.

IV. RESULT

All of the mentioned algorithms are implemented using C++ and compiled by g++. All experiments are conducted on a GNU/Linux machine, using 4 cores of intel i5 @2.40GHz and 5.7 GByte RAM. We have run every instance with a timeout of 500 seconds. We have generated different instances of this problem where the number of drivers was between 50 to 100 and the number of buses was between 50 to 100. We have created 20 instances where the edges between drivers and buses were selected randomly and then we compared the performance of two algorithms on those instances.

Figure 7 shows the run-time comparison of two algorithms with different number of edges. Here, x-axis represents the number of edges and y-axis represents run time in milli seconds. For both the algorithms, with the increasing number of edges, the run time increases. However, in most of the cases, Ford-Fulkerson algorithm performs better meaning takes less time than the Edmond-karp algorithm.

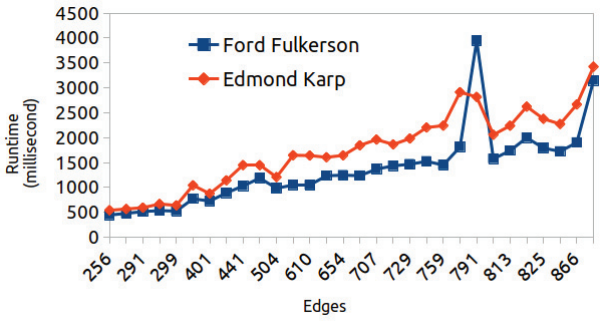


Fig. 7. Run-time comparison of two algorithms with different number of edges

Figure 8 shows the run-time comparison of two algorithms with different number of nodes. Here, x-axis represents the number of nodes and y-axis represents run time in milli seconds. In this case, the performance of the algorithms are similar as Figure 7.

In figure 9, x-axis represents the edge and y-axis represents the path length. The figure shows that for any number of edges the path length of Edmond-karp is much smaller than the path length of Ford-Fulkerson algorithm. We found similar behavior of these two algorithms for increasing number of nodes and thus we do not show the graph again here.

In figure 10 and 11, x-axis represents the run time in milli seconds and y-axis represents the path length. We see that the path length of Ford-Fulkerson algorithm is much greater than Edmond-karp algorithms' path length while the run time of Ford-Fulkerson algorithm is less than Ford-Fulkerson algorithm.

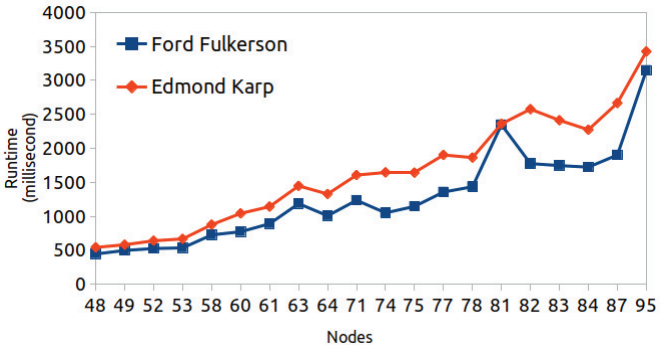


Fig. 8. Run-time comparison of two algorithms with different number of nodes

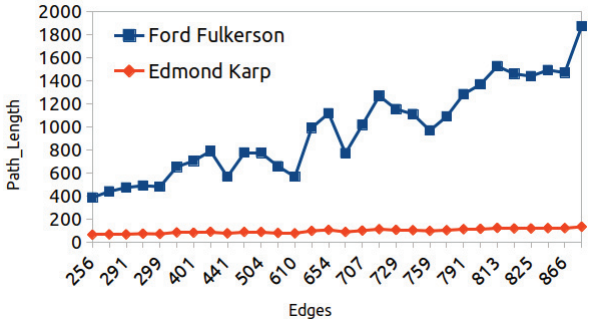


Fig. 9. Path-length comparison of two algorithms with different number of edges

The experimental analysis demonstrates that the Ford-Fulkerson algorithm outperforms Edmond-karp algorithm with respect to time although the path length of Edmond-karp algorithm is less than the path length of Ford-Fulkerson algorithm. There is a trade-off between run time and path length of these two algorithms in this case. In summary, since the run time of Ford-Fulkerson algorithm is less than Edmond-karp in all cases, we can decide that Ford-Fulkerson algorithm is better than Edmond-karp algorithm to solve the bus-driver assignment problem.

V. CONCLUSION

Due to complexity of assigning huge number of bus drivers, determining an appropriate assignment solution is a must need for increasing transport companies. In this research, we proposed an efficient algorithm, for solving the assignment of equally paid bus drivers in the best possible manner so that each bus have atmost one driver. We tried two different methods like Ford Fulkerson and Edmond Karp for finding better solution. Both methods have their own pros and cons. But Ford Fulkerson Method is mostly preferred as it takes less time than the other algorithm. Our algorithm for finding appropriate drivers for every bus is quite simple and doesn't require any complicated information which can be confusing to user.

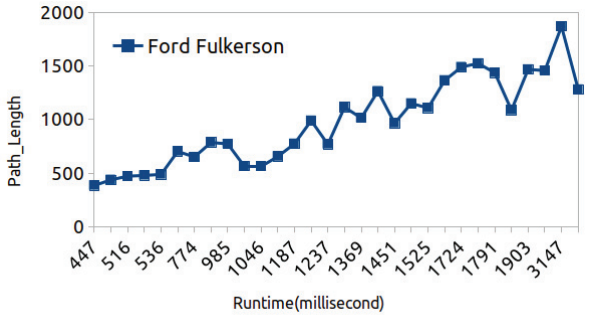


Fig. 10. Run time of Ford-Fulkerson algorithm with respect to path-length

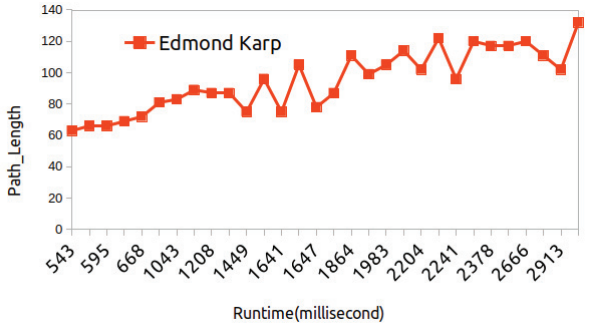


Fig. 11. Run Time of Edmond-karp algorithm with respect to path-length

In future, we hope to implement this for second version of situation where each drivers have their own sets of preferred salary. We also hope to compare other assignment algortihms for better solutions. After preparing both versions of this situations, we want this to be in a user friendly available format for transport companies.

REFERENCES

[1] R. Borndörfer, T. Schlechte, and S. Weider, "Railway track allocation by rapid branching," in *OASIcs-OpenAccess Series in Informatics*, vol. 14. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2010.

[2] D. Huisman, R. Freling, and A. P. Wagelmans, "Multiple-depot integrated vehicle and crew scheduling," *Transportation Science*, vol. 39, no. 4, pp. 491–502, 2005.

[3] H. Li, Y. Wang, S. Li, and S. Li, "A column generation based hyper-heuristic to the bus driver scheduling problem," *Discrete Dynamics in Nature and Society*, vol. 2015, 2015.

[4] J. Ma, A. Ceder, Y. Yang, T. Liu, and W. Guan, "A case study of beijing bus crew scheduling: a variable neighborhood-based approach," *Journal of Advanced Transportation*, vol. 50, no. 4, pp. 434–445, 2016.

[5] R. De Leone, P. Festa, and E. Marchitto, "A bus driver scheduling problem: a new mathematical model and a grasp approximate solution," *Journal of Heuristics*, vol. 17, no. 4, pp. 441–466, 2011.

[6] J. Li and R. S. Kwan, "A fuzzy genetic algorithm for driver scheduling," *European Journal of Operational Research*, vol. 147, no. 2, pp. 334–344, 2003.

[7] S. Chen and Y. Shen, "An improved column generation algorithm for crew scheduling problems," *Journal of Information and Computational Science*, vol. 10, no. 1, pp. 175–183, 2013.

[8] A. A. Constantino, C. F. X. de Mendonça Neto, S. A. de Araujo, D. Landa-Silva, R. Calvi, and A. F. dos Santos, "Solving a large real-world bus driver scheduling problem with a multi-assignment based heuristic algorithm," *Journal of Universal Computer Science*, vol. 23, no. 5, pp. 479–504, 2017.

[9] R. Ramli, H. Ibrahim, and L. T. Shung, "Innovative crossover and mutation in a genetic algorithm based approach to a campus bus driver scheduling problem with break consideration and embedded overtime," *Applied Mathematics & Information Sciences*, vol. 7, no. 5, p. 1921, 2013.

[10] J. E. Beasley and P. C. Chu, "A genetic algorithm for the set covering problem," *European journal of operational research*, vol. 94, no. 2, pp. 392–404, 1996.

[11] A. Caprara, M. Fischetti, and P. Toth, "A heuristic method for the set covering problem," *Operations research*, vol. 47, no. 5, pp. 730–743, 1999.

[12] M. Ohlsson, C. Peterson, and B. Söderberg, "An efficient mean field approach to the set covering problem," *European Journal of Operational Research*, vol. 133, no. 3, pp. 583–595, 2001.

[13] G. R. Mauri and L. A. N. Lorena, "A new hybrid heuristic for driver scheduling," *International Journal of Hybrid Intelligent Systems*, vol. 4, no. 1, pp. 39–47, 2007.

[14] M. Lozano, D. Molina, and F. Herrera, "Editorial scalability of evolutionary algorithms and other metaheuristics for large-scale continuous optimization problems," *Soft Computing*, vol. 15, no. 11, pp. 2085–2087, 2011.

[15] A. Y. Abdelaziz, R. A. Osama, S. El-Khodary, and B. K. Panigrahi, "Distribution systems reconfiguration using the hyper-cube ant colony optimization algorithm," in *International Conference on Swarm, Evolutionary, and Memetic Computing*. Springer, 2011, pp. 257–266.

[16] N. Bacanin, I. Brajevic, and M. Tuba, "Firefly algorithm applied to integer programming problems," *Recent Advances in Mathematics*, vol. 888, p. 999, 2013.

[17] G. P. Silva and A. F. d. S. Reis, "A study of different metaheuristics to solve the urban transit crew scheduling problem," *Journal of Transport Literature*, vol. 8, no. 4, pp. 227–251, 2014.

[18] J. Li, "A self-adjusting algorithm for driver scheduling," *Journal of Heuristics*, vol. 11, no. 4, pp. 351–367, 2005.

[19] R. De Leone, P. Festa, and E. Marchitto, "Solving a bus driver scheduling problem with randomized multistart heuristics," *International Transactions in Operational Research*, vol. 18, no. 6, pp. 707–727, 2011.

[20] Y. Shen and R. S. Kwan, "Tabu search for driver scheduling," in *Computer-Aided Scheduling of Public Transport*. Springer, 2001, pp. 121–135.

[21] L. Zhao, "A heuristic method for analyzing driver scheduling problem," *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans*, vol. 36, no. 3, pp. 521–531, 2006.

[22] M. Mesquita and A. Paias, "Set partitioning/covering-based approaches for the integrated vehicle and crew scheduling problem," *Computers & Operations Research*, vol. 35, no. 5, pp. 1562–1575, 2008.

[23] B. Laurent and J.-K. Hao, "Simultaneous vehicle and crew scheduling for extra urban transports," in *International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems*. Springer, 2008, pp. 466–475.

[24] S. W. de Groot and D. Huisman, "Vehicle and crew scheduling: solving large real-world instances with an integrated approach," in *Computer-aided Systems in Public Transport*. Springer, 2008, pp. 43–56.