WORK ZONE COST ANALYSIS UNDER DETOUR SETTINGS IN LANE MILE LEVEL

A Thesis

Presented to the Faculty of the Graduate School

Of Cornell University

In Partial Fulfillment of the Requirements for the Degree of

Master of Science

by

Enhua Liu

August 2016

© 2016 Enhua Liu

ALL RIGHTS RESERVED

ABSTRACT

The execution of road maintenance projects requires a determination of what sections of lanes on specific facilities will not be available over time. Generally speaking this problem is referred to as the work zone design problem. The cost associated with a work zone design are delay costs, set up costs for the road work, accident costs and the construction costs themselves. The construction costs are generally based on the work to be performed and the number of lane miles involved, therefore are independent of the work zone design. This thesis develops an optimization model and solution method to automatically design work zones as to minimize these costs. The formulation and solution procedure were tested on a version of the Sioux Falls network. The resultant plan was 33% cheaper in comparison to two intuitive construction schedules.

Keywords: Work Zone, User Delay, Genetic Algorithm, Frank-Wolfe Algorithm, Bi-level Optimization, Integer Programming, Monetary Value.

BIOGRAPHICAL SKETCH

Enhua Liu was born in Shijiazhuang, capital city of Hebei Province in China. He has great passion in learning and solving problems. He completed his junior high school as a top student, stands out as top 100 out of 70,000 student in the entrance exam of high school in the city. And he kept facing new challenges, competed in the Chinese Physics Olympiad, won the silver prize. In the National College Entrance Examination 2010, he ranked 700 out of 560,000 students in the Province of Hebei. He has great interest in Structural Engineering, which motivated him to apply for Civil Engineering in Tongji University, a best Civil Engineering college in China. After 4 years of undergraduate study, he wants to change the severe congestion in China, so he changed his major into Transportation System Engineering at the time he applied for Cornell. As a Master of Science student at Cornell, he developed his path to observe and think critically.

ACKNOWLEGEMENTS

This thesis is made possible through the support of everyone: professors at Cornell, friends and family.

The first and foremost, I want to express my sincere gratitude to my advisor Prof. Linda K. Nozick, who has been very patient and considerate to give me guidance and discuss the flaws when we was building this model. As a Civil Engineering student in undergrad, Prof. Nozick enlightened me of the path to Transportation Systems Engineering. I would also like to thank for Prof. Huaizhu Gao, my minor advisor, who is very knowledgeable and friendly.

Secondly, I want to thank Jupang Li, Qingyuan Zhao and Tong Zhu, they helped me with the coding algorithms as well as debugging, Which is very time consuming and painful. I am very fortunate to have reliable friends like them.

Last but not least, I wish to express my profound gratitude to my parents. Without their encouragement and support throughout all 20 years, I could never go this far. I am grateful for their endless love and care.

TABLE OF CONTENTS

Biographical Sketch i

Acknowlegement ii

Table of Contents iii

List of Figures iv

List of Tables v

Introduction and Literature Review 1

Model Formulation 4

Solution Procedure 7

3.1 Gene and Chromosome Definition 7

3.2 Fitness of a chromosome 8

3.3 Selection 8

3.4 Crossover 8

3.5 Mutation 10

Case Study 11

Conclusion 17

LIST OF FIGURES

FIGURE 1 Demonstration of genes 8

FIGURE 2 Crossover to create children 9

FIGURE 3 Mutation of genes 10

FIGURE 4 Sioux Falls Network..... 12

FIGURE 5 Construction schedules 13

FIGURE 6 Best construction schedule with 1.5 times of OD flows. 15

LIST OF TABLES

TABLE 1 Comparison with Intuitive Schedules.....144

TABLE 2 Cost with 1.5 OD Trip Volume 15

CHAPTER 1
Introduction and Literature Review

In order to keep highway facilities in good condition, maintenance and reconstruction is necessary. Since these activities require the temporary closure of these facilities, these activities can induce substantial congestion. Hence, the determination of which lanes to close is of substantial importance. In contrast, agency costs are predominately fixed because much of it is driven by the per-mile costs of the maintenance; however there are set-up costs that vary based on the construction plan adopted.

Given the potential impacts on congestion, there has been substantial research into the staging of maintenance in recent years. Some of that research has focused on queueing delays caused by changes in capacity and speed in work zones. Fazil T. Najafi and Roberto Soares (2001) conducted an analysis to evaluate work zone cost using the five most common approaches; they indicated that traffic flow and roadway capacity are the most important parameters influencing work zone user cost (*1*).

Additionally, construction plans may affect user cost in work zones. Jiang and Adeli (2003) developed a model to minimize freeway total work zone cost in terms of work zone segment length and starting time of construction (*2*). They also considered other factors like the number of lane closures and darkness. In a later study Jiang and Adeli (2004) presented an object-oriented model for freeway work zone capacity and queue delay estimation (*3*). Chien and Schonfeld (2001) proposed a simplified and useful mo1del for estimating the delay cost using traffic flow and work zone speed and

finding the optimum work zone segment length in a four-lane freeway with partial lane closure (*4*). Their results showed that shortening work zone segments would mitigate user delay but increase agency cost; therefore, the schedule of work zones is important to both agency and user cost.

It is important to notice that none of these papers includes the possibility that drivers will change their route. With the reduction of capacity in work zones, users which originally used links that are now under construction may choose alternative routes to get to their destination. Also, their diversion onto other roads will increase traffic volumes and therefore travel times. Hence individuals traveling on routes that did not use the road segments that are under construction may experience higher travel times. Some of these travelers may also find alternative routes as a response to these travel time increases. Hence to compute the user delay costs, the change in travel time across all origin-destination pairs must be considered. For the purposes of this thesis, we focus on user equilibrium as the route choice rule for travelers. That is, we assume a stable condition is reached only when no traveler can improve his travel time by unilaterally changing routes (*5*). The user equilibrium assignment is an essential element to calculate the user delay cost.

This thesis makes two important contributions to the literature. First, the potential for drivers to select different routes is considered when calculating user delay costs. Second, it develops an optimization model and solution procedure to determine the optimal scheduling of construction on multiple links so as to minimize total costs.

This paper formulates a bi-level optimization model to determine the best

construction schedule for several links in a large road network. We assume that the upper level represents the concerns of the agency; which are expected to be to minimize set up costs, accident costs and user delay costs. The lower level model represents the route selection decisions of the travelers that creates the user delay costs. A solution procedure is developed using a genetic algorithm for which the evaluation of a given solution requires the solution of a user equilibrium traffic assignment problem (*6*). The formulation and solution procedure is tested on the Sioux Falls network. That network consists of 24 nodes and 76 links.

This thesis has 5 sections. The second section gives the model formulation. The third section gives the solution procedure. The fourth section gives a case study. The final section gives conclusions and opportunities for future work.

CHAPTER 2
Model Formulation

As mentioned previously, the model structure is a bi-level optimization where the upper level is an integer program and the lower level is a user equilibrium model. The next two subsections give the formulation of these two models.

A transportation network represented by *G= (N,A)*, where *N* is the set of nodes and *A* is the set of directed links. Now, suppose all links are divided in miles, which we call each mile as a sub-link, there are multiple lanes on each sub-link.

Suppose there is a construction project that requires each lane mile on several different links to be removed from service for a fixed duration *d*. Let the length of the planning horizon available to perform all of the construction be divided up into *T* time periods each of which is of length *d*. Then the project may be specified by the input data, $Q_{a,l}$ which is the number of lanes on sub-link *l* of link a. Now, there is a decision variable $y_{a,l,t}$ that refers to the number of lanes under construction in time period *t*. Hence constraints (1) and (2) must hold.

$$\sum_t y_{alt} = Q_{al} \quad \forall a, l, t \tag{1}$$

$$0 \leq y_{alt} \leq 5000 \times x_{alt} \quad \forall a, l, t \tag{2}$$

Where

$x_l = \{0,1\}$ is a decision variable of y_{alt}.

Equation (1) requires that the maintenance work on each lane for each link is

accomplished at some point over the planning horizon. Equation (2) is the restrictions on the variables.

We assume that if construction on a lane-mile is paused, a new set-up cost is incurred. Let S_{at} be a binary variable that takes on a value of 1, if a set up cost is incurred for the construction on sub-link *l* of link a in time period *t*, where n_a is the number of lanes on link a. Then Equation (3) identifies in which periods a set-up cost on each link is incurred.

$$\frac{\sum_l y_{alt} - \sum_l y_{al(t-1)}}{n_a} \leq S_{at} \quad \forall a, t \tag{3}$$

Suppose C_{at} is capacity of link a in time period *t*. Equation (4) below gives the computation of capacity on link a in time period *t* (*7*).

$$C_{alt} = 1450 \times (Q_{al} - y_{alt}) \times x_{alt} + 2000 \times Q_{al} \times (1 - x_{alt}) \; \forall a, l, t \tag{4}$$

$$C_{at} = \sum_l C_{alt} \; \forall a, t \tag{5}$$

Now, suppose that the variable V_{at} is the volume on link a during time period *t*. The goal of the upper level problem is to minimize the following.

$$\sum_{ta}(0.5)(1.65)(MHI)V_{at}t_a\,(1 + 0.15(\frac{V_{at}}{C_{at}})^4) + \sum_{at} ZS_{at} + \sum_{ta} NCV_{at}t_a\,(1 + 0.15(\frac{V_{at}}{C_{at}})^4) \tag{6}$$

Where t_a is the free flow travel time on link a, MHI is hourly median household income, *Z* is the set-up cost, *N* is the accident rate per mile and *C* is the cost per accident.

Notice that the MHI is multiplied by 0.5 and 1.67 (*8*). In the case study we use half of median household income. Further, 1.67 is the average vehicle occupancy.

The upper level problem is then to minimize (6) subject to (1)-(5) where the link volumes are obtained from a user equilibrium problem for each time period. In order to solve for user equilibrium condition, we used Dijkstra's algorithm to find the shortest path for each O-D pair (*9*). Then we used Frank-Wolfe algorithm to solve for traffic assignment (*10*).

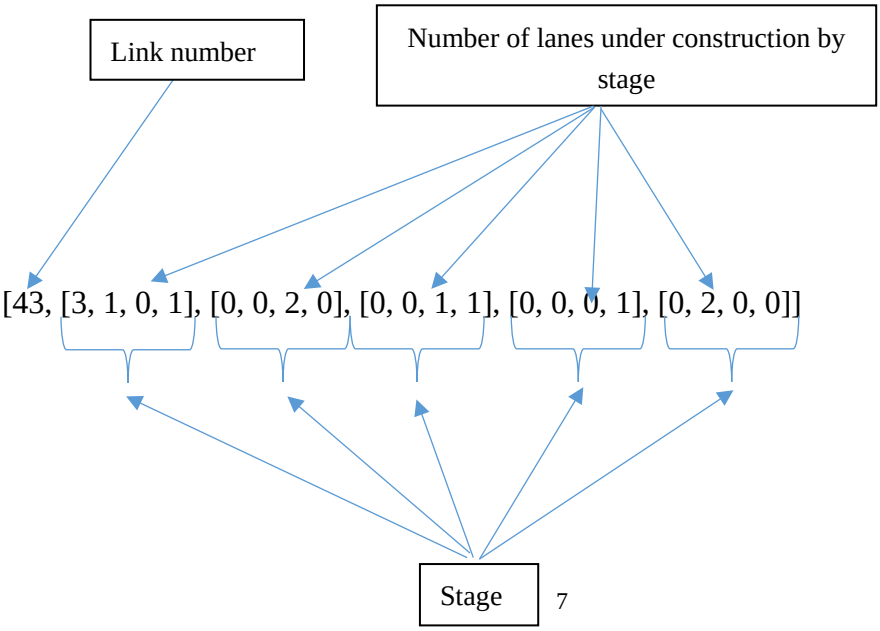
CHAPTER 3

Solution Procedure

As mentioned previously, a genetic algorithm is used to solve the model. The key steps in the specification of a genetic algorithm are, (1) specify the chromosome; (2) evaluate the fitness of a gene; (3) selection; (4) crossover; and (5) mutation. The remainder of this section describes each of these steps in turn.

3.1 Gene and Chromosome Definition

The gene used gives the order for the construction on a link. The entire chromosome therefore gives the entire project plan. First element is the link number, and then a list of the number of lanes under construction in each stage. Hence length of each list equals the length of that link. An example of gene is displayed in Figure 1. For example, in the first stage, on link 43 for sub-link a, three lanes are under construction. In stage five for link 43 for sub-link b, two lanes are under construction.



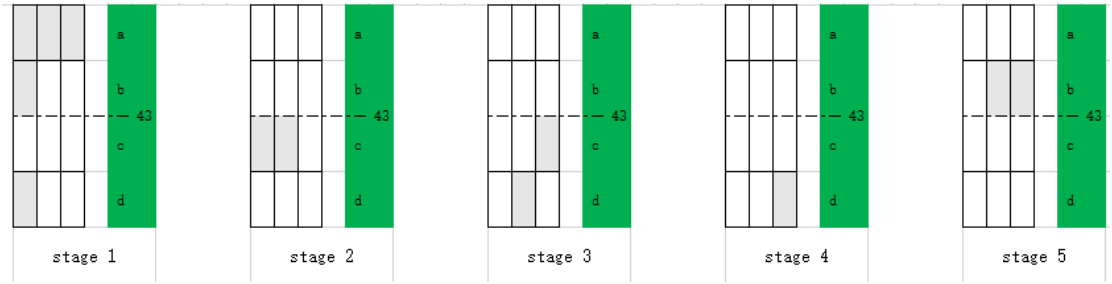


FIGURE 1 Demonstration of genes

3.2 Fitness of a chromosome

The fitness of a chromosome is evaluated based on the objective function given in equation (6). This evaluation requires the computation of the travel times that stem from user equilibrium. In the example given in Figure 1 above this requires 5 user equilibrium computations.

3.3 Selection

The chromosomes in the pool are ordered based on decreasing fitness value and only the top 20% are considered for mating. Within the top 20% two different individuals are selected repetitively until a new generation is constructed.

3.4 Crossover

Each gene is extracted from each of the parent chromosomes and cross-over is performed on each separately (i.e. on each link separately). The pair of genes for the

same link (one from each parent) is cut at a random point that gives the number of lanes under construction in each lane mile that will be taken from the first in contrast to the second parent. The reverse assignment is used for the second child. Consider the following example. Suppose the crossover point is at the end of the second mile. Notice that for stage 1 on link 43 stage 1, the first child receives from the first parent the chromosome for sub-link a and b whereas the second parent contributes the chromosome for sub-link c and d. This process is repeated for each stage to construct the gene for the first child. For the second child, the assembly is the same with the roles of the parents reversed.

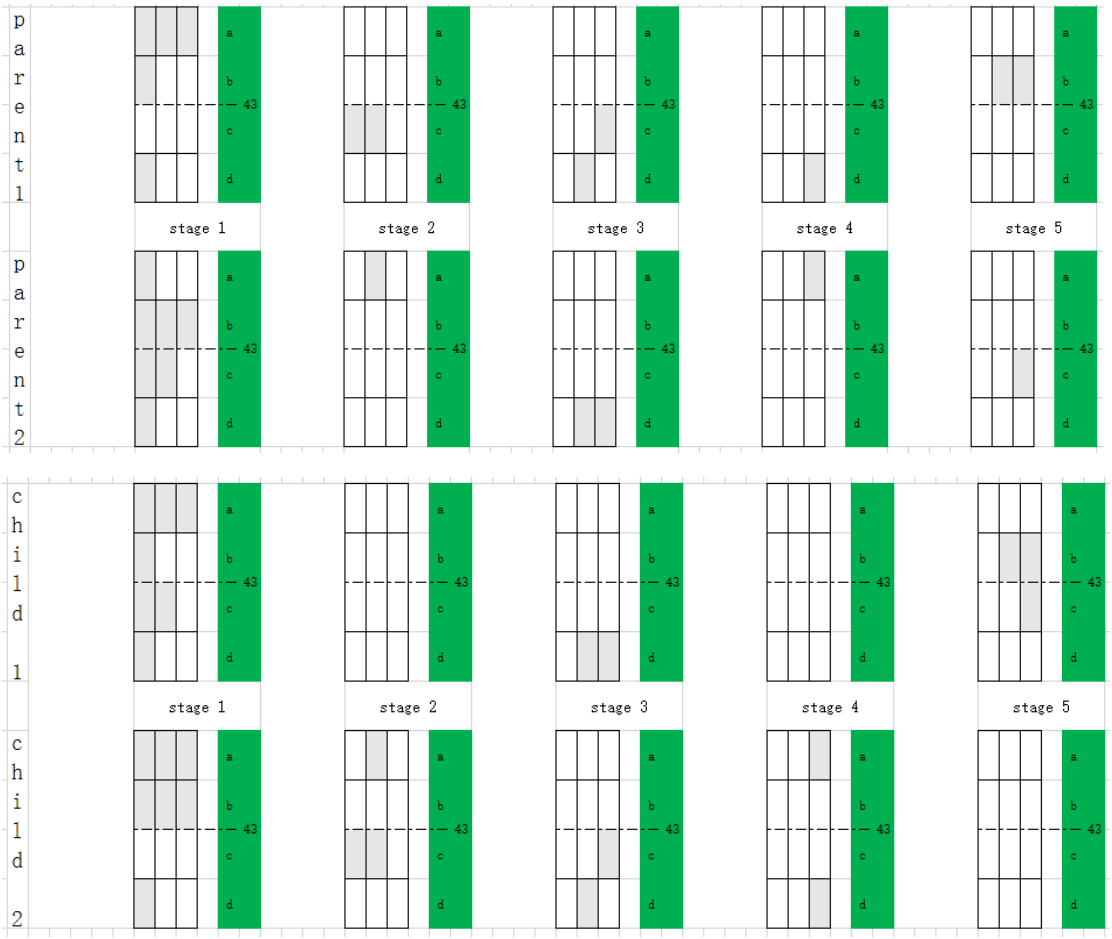


FIGURE 2 Crossover to create children

3.5 Mutation

Finally, in order to maintain diversity a sub-link on a link is randomly chosen, the existing construction plan is removed and a randomly generated one is used in its place. Figure 3 illustrates and instance of random mutation on the gene for child 1 from Figure 2.

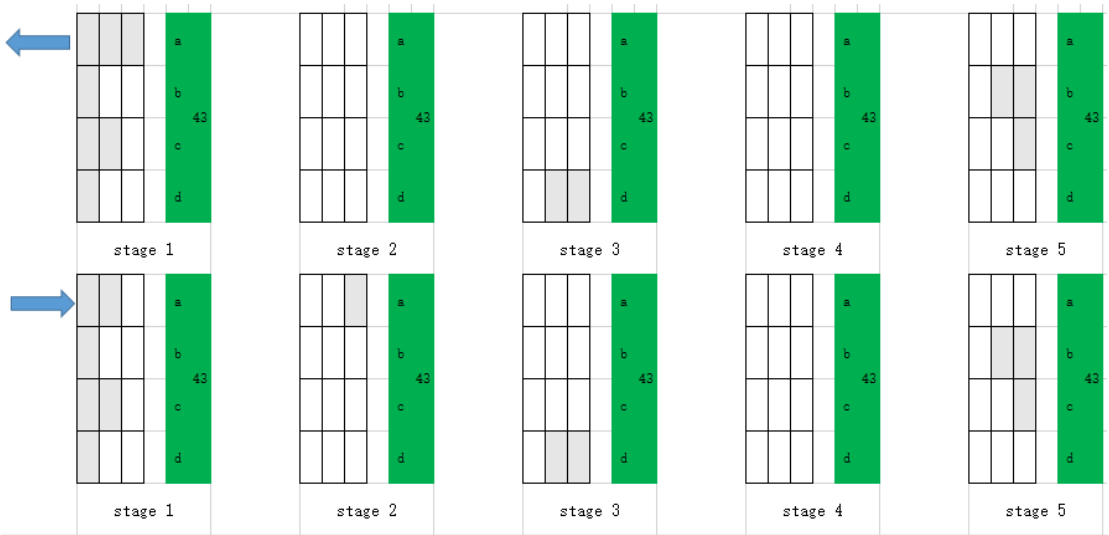


FIGURE 3 Mutation of genes

CHAPTER 4
Case Study

Median household income was about $53,657 in 2014. This yields an hourly median income of about $26. The accident rate used is 40 per100 million vehicle hours and the average cost of an accident is assumed to be $142,000.

This study focused on the Sioux Falls freeways (FIGURE 4). This network is represented by 24 nodes and 76 links, where nodes are the beginnings and endings of links (*11*). Each node has an O-D trip to other nodes. A road with 5 links was considered to be under construction, and there are multiple lanes on each link. A heuristic calculation for lowest cost was done in order to determine the best construction plan.

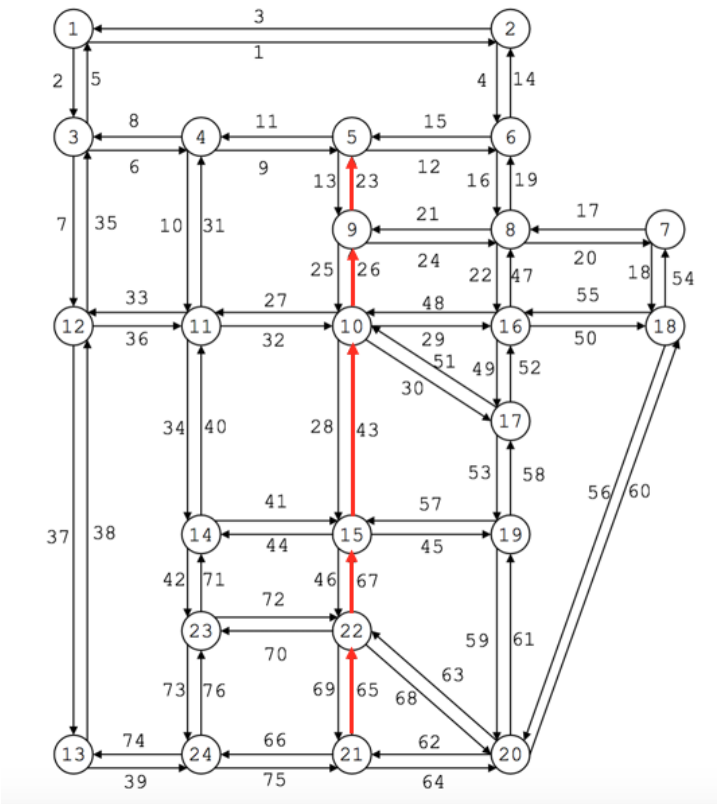
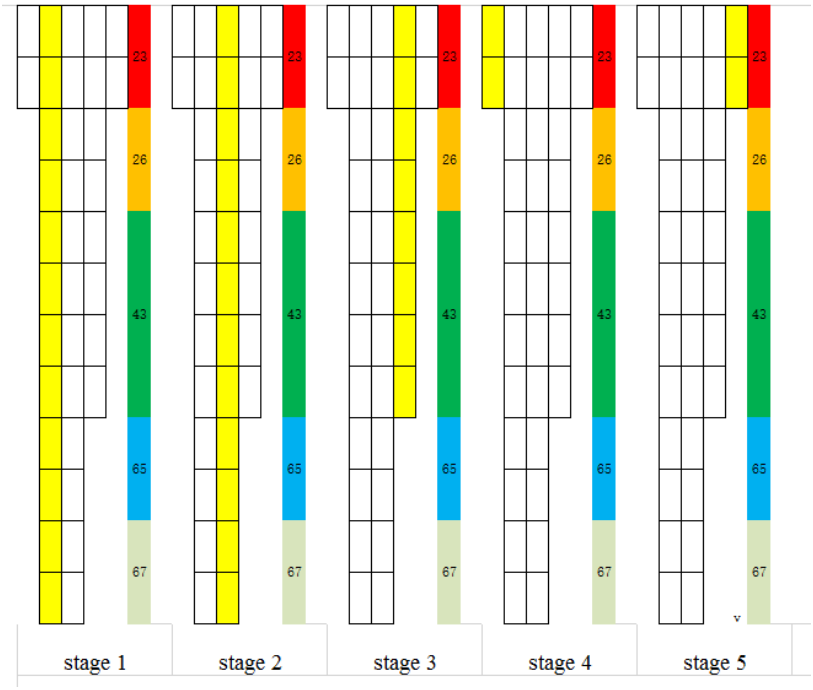
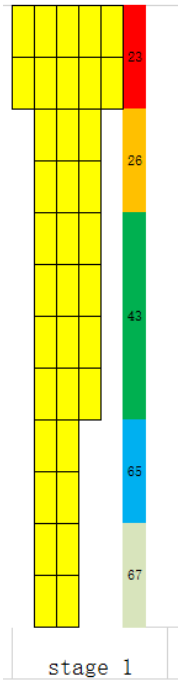


FIGURE 4 Sioux Falls Network

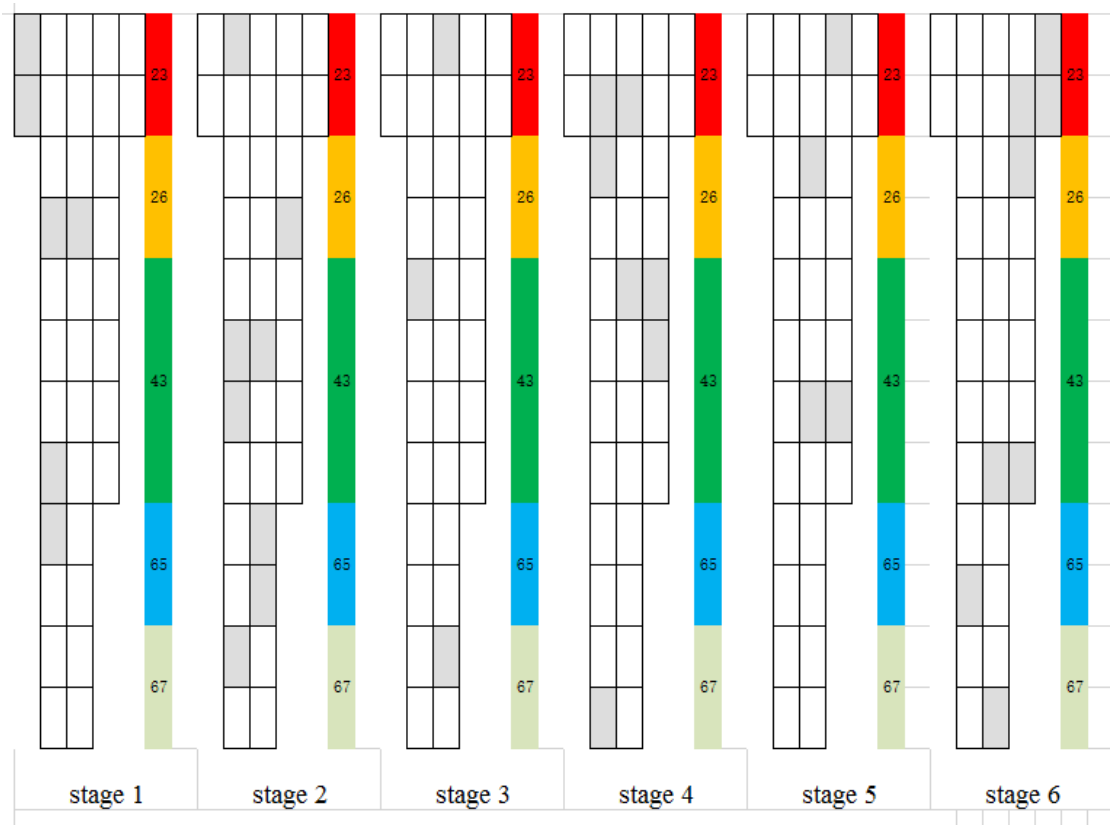
We use a population of 1000, a mutation rate of 0.3 and a random selection rate of 0.05 as the parameters of the GA. The results are demonstrated in Figure 5, the shaded area refers the lane-mile closure in that stage. Figure 5 (c) gives the best schedule produced by the model and solution procedure. Figures 5 (a) and (b) give two intuitive schedules. Figure 5 (a) is the “do one lane in each stage” schedule whereas Figure 5 (b) is the “do the entire project at once” closing the entire facility to traffic.



(a)



(b)



(c)

FIGURE 5 Construction schedules: (a) close one lane each stage, (b) full lane closure

and (c) best schedule found.

Table 1 shows the detailed cost comparison between the best schedule we achieved by GA and intuitive schedules.

TABLE 1 Comparison with Intuitive Schedules

Schedule	Total cost ($)	User delay cost ($)	Setup cost ($)	Construction cost ($)	Accident cost ($)
(a)	30,888,420	12,805,849	48,000	18,000,000	34,570
(b)	89,149,665	70,910,235.	48,000	18,000,000	191,430
(c)	20,611,031	2,524,216	80,000	18,000,000	6,814

The result indicates the intuitive plan results in higher user delay costs. This is due to the enormous change in the user cost when multiple lanes of different links are closed simultaneously. When more lanes are shut down, the change in user cost becomes more sensitive to lane closure. It should be noted that links advance through the stages of construction at varying paces. The schedule indicates the stage by stage development of construction, not its precise chronology.

As mentioned earlier the lane closure strategies may have an appreciable impact on work zone cost. Compared to the intuitive construction plans, the chromosome generated greatly reduces the cost. Assuming that closing all facilities at the same time is not reasonable, we conclude that total costs are reduced by as much as 33%, which, in this example, translates into 10.3 million dollars.

It is interesting to consider how these benefits my change as the congestion on the

network increases. In this example the origin-destination table is multiplied by 150%. Once this is done, the average V/C ratio during peak hours rises to 1.155 from an average of 0.77. This corresponds to a decline in service from level of service C to F. Figure 6 gives the construction plan suggested by the solution procedure. It is useful to notice that a 50% increase in the OD table has led to a tripling of the user delay costs. This is consistent with a study conducted by Fazil T. Najafi and Roberto Soares, which indicated that traffic flow and capacity are the most important parameters for a work zone (*2*).

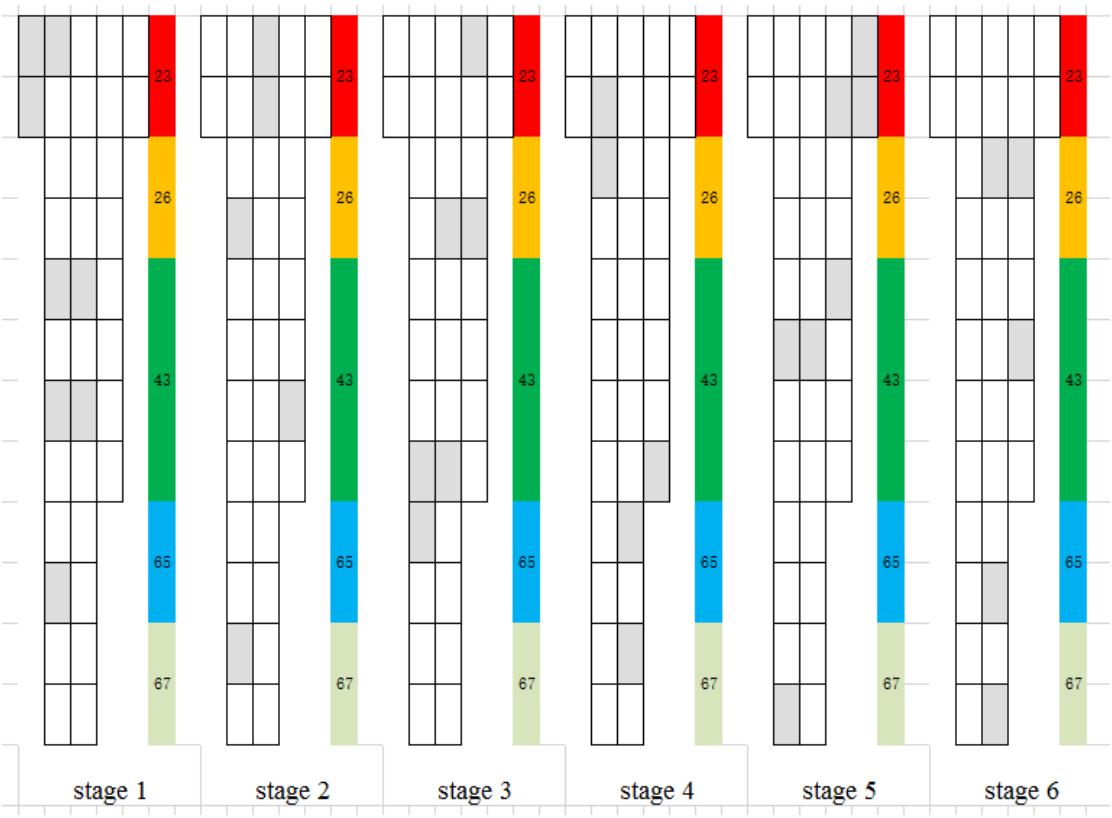


FIGURE 6 Best construction schedule with 1.5 times of OD flows.

TABLE 2 Cost with 1.5 OD Trip Volume

Schedule	Total cost ($)	User delay cost ($)	Setup cost ($)	Construction cost ($)	Accident cost ($)
(a)	43,197,022	25,081,312	48,000	18,000,000	67,710
(b)	739,882,132	719,890,701.	48,000	18,000,000	1,943,431
Figure 6	24,787,399	6,697,319	72,000	18,000,000	18,080

CHAPTER 5
Conclusion

This study provides a new approach to optimizing the construction plan for road maintenance with explicit consideration of the opportunities for drivers to select alternative routes. The Sioux Falls network was used understand how these optimized plans would perform in comparison to easily developed “intuitive” schedules. The analysis illustrated that well-constructed work zone designs can lead to significant savings, primarily in user delay and accident costs. Set up costs may rise but the magnitude of those costs is generally substantially smaller than the other costs.

It is important to note that there are opportunities for additional research. This model used a static link assignment model. It would be useful if the traffic dynamics were included by replacing that model with a dynamic traffic assignment algorithm. To make that algorithmic change more potent, a dynamic origin-destination table would be desirable. Related to the idea of incorporating a dynamic traffic assignment algorithm, this modeling effort assumed all drivers have perfect information and there is no representation of driver information as to the construction plan. It would be very beneficial to relax the assumption of perfect information and consider the impacts of different methods of conveying construction information to drivers.

REFERENCES

1. Najafi, F.T. and Soares, R. User costs at the work zone. *Canadian Journal of Civil Engineering*, Vol 28, No. 4, 2001, pp. 747-751.
2. Jiang, Xiaomo and Adeli, Hojjat. Freeway work zone traffic delay and cost optimization model. *Journal of Transportation Engineering*, Vol 129, No. 3, 2003, pp. 230-241.
3. Jiang, Xiaomo and Adeli, Hojjat. Object-oriented model for freeway work zone capacity and queue delay estimation. *Computer-Aided Civil and Infrastructure Engineering*, Vol 19, No. 2, 2004, pp. 144-156.
4. Chien, S. and Schonfeld, P. Optimal work zone lengths for four-lane highways. *Journal of Transportation Engineering*, Vol 127, No. 2, 2001, pp. 124-131.
5. Yossi Sheffi. *Urban Transportation Networks.* Prentice Hall, 1985.
6. John H. Holland. *Adaptation in Natural and Artificial Systems*. A Bradford Book, 1982
7. *Guide for Highway Capacity and Operations Analysis of Active Transportation and Demand Management Strategies.* Appendix D: Speed/Capacity for Work Zones. http://ops.fhwa.dot.gov/publications/fhwahop13042/appd.htm. Accessed Feb 25.
8. Work Zone Road User Costs - Concepts and Applications. Chapter 2, 2.2.2. FHWA.USDOT. http://ops.fhwa.dot.gov/wz/resources/publications/fhwahop12005/sec2.htm,
9. A Note on Two Problems in Connexion with Graphs. *Numerische Mathematik*, 1959, 269-271

10. LeBlanc, L.J.; Morlok, E.K. and Pierskalla, W.P. An efficient approach to solving the road network equilibrium traffic assignment problem. *Transportation Research*, Vol 9, No. 5, 1975, 309-18.

11. Bar Gera, Transportation test problems, Sioux Falls network. https://github.com/bstabler/TransportationNetworks/tree/master/SiouxFalls. Accessed Feb 20, 2016.

12. Tapan P. Bagchi. *Multiobjective Schedule by Genetic Algorithms*. Springer Science+Business LLC. 1999

APPENDICES

Appendix A: Network parameters.

link number	Length	free flow time	capacity	lanes
1	8	0.13333333	25900.2006	2
2	2	0.03333333	23403.4732	6
3	8	0.13333333	25900.2006	2
4	2	0.03333333	4958.18093	1
5	2	0.03333333	23403.4732	6
6	2	0.03333333	17110.5237	4
7	4	0.06666667	23403.4732	3
8	2	0.03333333	17110.5237	4
9	3	0.05	17782.7941	3
10	4	0.06666667	4908.82673	1
11	3	0.05	17782.7941	3
12	3	0.05	4947.99547	1
13	2	0.03333333	10000	3
14	2	0.03333333	4958.18093	1
15	3	0.05	4947.99547	1
16	2	0.03333333	4898.58765	1
17	3	0.05	7841.81131	1
18	2	0.03333333	23403.4732	6

19	2	0.03333333	4898.58765	1
20	3	0.05	7841.81131	1
21	3	0.05	5050.19316	1
22	2	0.03333333	5045.82258	1
23	2	0.03333333	20900.2006	5
24	3	0.05	5050.19316	1
25	2	0.03333333	13915.7884	3
26	2	0.03333333	13915.7884	3
27	3	0.05	10000	2
28	4	0.06666667	13512.0016	2
29	3	0.05	4854.91772	1
30	3	0.05	4993.51069	1
31	4	0.06666667	4908.82673	1
32	3	0.05	10000	2
33	2	0.03333333	4908.82673	1
34	4	0.06666667	4876.50829	1
35	4	0.06666667	23403.4732	3
36	2	0.03333333	4908.82673	1
37	8	0.13333333	25900.2006	2
38	8	0.13333333	25900.2006	2
39	2	0.03333333	5091.25615	1
40	4	0.06666667	4876.50829	1
41	3	0.05	5127.52612	1
42	2	0.03333333	4924.79061	1

43	4	0.06666667	25512.0016	3
44	3	0.05	5127.52612	1
45	3	0.05	14564.7532	2
46	2	0.03333333	9599.18057	2
47	2	0.03333333	5045.82258	1
48	3	0.05	4854.91772	1
49	2	0.03333333	5229.91006	1
50	3	0.05	19679.8967	3
51	3	0.05	4993.51069	1
52	2	0.03333333	5229.91006	1
53	2	0.03333333	4823.95083	1
54	2	0.03333333	23403.4732	6
55	3	0.05	19679.8967	3
56	9	0.15	23403.4732	1
57	3	0.05	14564.7532	2
58	2	0.03333333	4823.95083	1
59	4	0.06666667	5002.60756	1
60	9	0.15	23403.4732	1
61	4	0.06666667	5002.60756	1
62	3	0.05	5059.91234	1
63	4	0.06666667	5075.69719	1
64	3	0.05	5059.91234	1
65	2	0.03333333	8229.91006	2
66	3	0.05	4885.35756	1

67	2	0.03333333	9599.18057	2
68	4	0.06666667	5075.69719	1
69	2	0.03333333	5229.91006	1
70	3	0.05	5000	1
71	2	0.03333333	4924.79061	1
72	3	0.05	5000	1
73	2	0.03333333	5078.50844	1
74	2	0.03333333	5091.25615	1
75	3	0.05	4885.35756	1
76	2	0.03333333	5078.50844	1

Appendix B: O-D trips.

1				
0	100	100	500	200
300	500	800	500	1300
500	200	500	300	500
500	400	100	300	300
100	400	300	100	
2				
100	0	100	200	100
400	200	400	200	600
200	100	300	100	100

400	200	0	100	100
0	100	0	0	
3				
100	100	0	200	100
300	100	200	100	300
300	200	100	100	100
200	100	0	0	0
0	100	100	0	
4				
500	200	200	0	500
400	400	700	700	1200
1400	600	600	500	500
800	500	100	200	300
200	400	500	200	
5				
200	100	100	500	0
200	200	500	800	1000
500	200	200	100	200
500	200	0	100	100
100	200	100	0	

6				
300	400	300	400	200
0	400	800	400	800
400	200	200	100	200
900	500	100	200	300
100	200	100	100	
7				
500	200	100	400	200
400	0	1000	600	1900
500	700	400	200	500
1400	1000	200	400	500
200	500	200	100	
8				
800	400	200	700	500
800	1000	0	800	1600
800	600	600	400	600
2200	1400	300	700	900
400	500	300	200	
9				
500	200	100	700	800
400	600	800	0	2800

1400	600	600	600	900
1400	900	200	400	600
300	700	500	200	
10				
1300	600	300	1200	1000
800	1900	1600	2800	0
4000	2000	1900	2100	4000
4400	3900	700	1800	2500
1200	2600	1800	800	
11				
500	200	300	1500	500
400	500	800	1400	3900
0	1400	1000	1600	1400
1400	1000	100	400	600
400	1100	1300	600	
12				
200	100	200	600	200
200	700	600	600	2000
1400	0	1300	700	700
700	600	200	300	400
300	700	700	500	

13				
500	300	100	600	200
200	400	600	600	1900
1000	1300	0	600	700
600	500	100	300	600
600	1300	800	800	
14				
300	100	100	500	100
100	200	400	600	2100
1600	700	600	0	1300
700	700	100	300	500
400	1200	1100	400	
15				
500	100	100	500	200
200	500	600	1000	4000
1400	700	700	1300	0
1200	1500	200	800	1100
800	2600	1000	400	
16				
500	400	200	800	500
900	1400	2200	1400	4400

1400	700	600	700	1200
0	2800	500	1300	1600
600	1200	500	300	
17				
400	200	100	500	200
500	1000	1400	900	3900
1000	600	500	700	1500
2800	0	600	1700	1700
600	1700	600	300	
18				
100	0	0	100	0
100	200	300	200	700
200	200	100	100	200
500	600	0	300	400
100	300	100	0	
19				
300	100	0	200	100
200	400	700	400	1800
400	300	300	300	800
1300	1700	300	0	1200
400	1200	300	100	

20				
300	100	0	300	100
300	500	900	600	2500
600	500	600	500	1100
1600	1700	400	1200	0
1200	2400	700	400	
21				
100	0	0	200	100
100	200	400	300	1200
400	300	600	400	800
600	600	100	400	1200
0	1800	700	500	
22				
400	100	100	400	200
200	500	500	700	2600
1100	700	1300	1200	2600
1200	1700	300	1200	2400
1800	0	2100	1100	
23				
300	0	100	500	100
100	200	300	500	1800

1300	700	800	1100	1000
500	600	100	300	700
700	2100	0	700	
24				
100	0	0	200	0
100	100	200	200	800
600	500	700	400	400
300	300	0	100	400
500	1100	700	0	

Appendix C: Python code main function.

```
import readmethod

from scipy.optimize import minimize_scalar

from odread import *

from dijkstra import *

import math

import xlrd

import odread

import readlanes

import random

from random import randint
```

```
import copy
"""
Clarification:
lanes refers to the number of lanes of each link
construction refers to construction plan of the road need to be shut down,
which means select what section and how many lanes in what period, the construction has
the attributes that sum of each part of the lists will be the lane number of
the link under construction
e.g. [21,[2,1,0,2],[0,1,2,0],[1,1,1,1]] is a 4 miles road with 3 lanes in one direction, link number is 21
construction: [[21,[2,1,0,2],[0,1,2,0],[1,1,1,1]],[9,[1,1],[1,1],[0,0]]]
"""
# using input lanes list, length list, cap list, construction list of lists of lists, capnew list of lists
# lanes list


def initial_X(edges,ODtable):
    """
    Get initial x for the network based on the free flow time cost
    """
    n = len(edges)
```

```
    x = [0]*n
    for o,d,q in ODtable:
        a,b,link = dijkstra(edges,o,d)
        for i in range(len(link)):
            index = link[i]
            x[index-1] += q



    return x


def cost(x,tf,cap):
    """
    Get the cost on each link i according to the volume x[i],
    the tf is the freeflow time on each link i and the cap is capacity of each link i
    """
    c = []


    for i in range(len(x)):
        cost = tf[i]*(1+0.15*(x[i]/cap[i])**4)
        c.append(cost)
    return c
```

```
def objectiveF(x,tf,cap):
    """
    Calculate the total cost for the network with volume x[i] on each link i
    the tf is the freeflow time on each link i and the cap is capacity of each link i
    """
    obj = 0
    c = cost(x,tf,cap)
    for i in range (len(x)):
        obj += tf[i]*(x[i]+0.03*x[i]**5/cap[i]**4)
    return obj


def updedges(edges,x,tf,cap):
    """
    Update time cost in list edges
    """
    g = defaultdict(list)


    for l,r,c,n in edges:
        g[n] = [l,r]




    neweds = []
```

```
        for i in range(len(x)):
            tmp = g[i+1]
            costi = tf[i]*(1+0.15*x[i]**4/cap[i]**4)
            tmp.extend([costi,i+1])
            neweds.append(tuple(tmp))



        return neweds


    def findy(x,ODtable,tf,cap):
        """
        With given x caculate the cost of each link and use dijkstra algorithm to find
the shortest path
        for each OD pair and assign all trips on it, get the link volume as y
        """


        y = [0]*len(x)
        edgesnew = updedges(edges,x,tf,cap)
        for o,d,q in ODtable:
            a,b,link = dijkstra(edgesnew,o,d)
            for i in range(len(link)):
                index = link[i]
```

```
            y[index-1] += q
        return y


    def findnewx(x,y,tf,cap):
        """
        With x and y of 2 feasible network flow, find z to minimize total cost on the
network,
        then save it as new x value
        Create a new list of traffic volume on link based on x and y and lmd
        """
        newx = []


        def trans(lmd):
            z = []
            for i in range(len(x)):
                zi = lmd*x[i]+(1-lmd)*y[i]
                z.append(zi)


            temp = objectiveF(z,tf,cap)
            return temp


        res = minimize_scalar(trans,bounds=(0,1),method='bounded')
```

```
    lmd = res.x
    #print ("====Total cost and lambda each iteration====")
    #print (trans(lmd))
    #print (lmd)
    #print ('\n')
    for j in range(len(x)):
        newx.append(lmd*x[j] + (1-lmd)*y[j])
    return newx


def frankwolfe_Iteration(x,ODtable,tf,cap):
    """
    Perform Frankwolfe algorithm iteration to get convergent point of link flow
    """
    y = findy(x,ODtable,tf,cap)
    xnew = findnewx(x,y,tf,cap)
    return xnew


def frankwolfe(edges,ODtable,tf,cap):
    """
    Get UE flow on each link with initial free flow cost and O-D talbe,
    put all trips on the shortest path identified by dijkstra algorithm,
    then update the cost on each link and reassign the trip volume in each iteration
```

```
"""
x = initial_X(edges,ODtable)


y = findy(x,ODtable,tf,cap)
diffsqrt = 10
xsum = 2


while diffsqrt/xsum > 0.001:
    xtmp = x
    xnew = frankwolfe_Iteration(x,ODtable,tf,cap)
    diff = 0
    xsum = 0
    for i in range(len(x)):
        diff += (xnew[i] - x[i])**2
        xsum += x[i]
    diffsqrt = math.sqrt(diff)
    x = xnew
return xnew
```

```
def gen_individual(linknum, teams):
    """
    Generate a feasible construction plan for given link number and number of phases
    """
    lanemile_set = []
    for i in range(len(linknum)):
        index = linknum[i]-1
        l = int(length[index])
        m = int(lanes[index])
        for j in range(l):
            for k in range(m):
                lanemile_set.append([index+1,j,k])
    totalLM = len(lanemile_set)
    global stages
    stages = totalLM/teams
    const_sche = []
```

```
    for s in range(int(stages)):
        const_sche.append([])
        while len(const_sche[s]) != teams:
            idx = randint(0,len(lanemile_set)-1)
            const_sche[s].append(lanemile_set[idx])
            lanemile_set.pop(idx)

    return const_sche

def convert_to_construction(linknum,teams,const_sche):
    links = []
    for i in range(len(linknum)):
        index = linknum[i]-1
        links.append([index+1])
        l = int(length[index])
        m = int(lanes[index])
        for j in range(int(stages)):
            links[i].append([])
            for k in range(l):
                links[i][j+1].append(0)
```

```
        # links = [[23, [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0]], [26, [0, 0], [0, 0], [0, 0],
[0, 0], [0, 0], [0, 0]],
        # [43, [0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0]],
        # [65, [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0]], [67, [0, 0], [0, 0], [0, 0], [0, 0], [0,
0], [0, 0]]]
        for d in range(int(stages)):
            """
            [[26, 1, 0], [67, 1, 1], [23, 1, 4], [67, 1, 0], [23, 1, 1], [43, 0, 1]] one stage of
const_sche
            """
            for lk in range(len(const_sche[d])):
                idx = const_sche[d][lk][0]
                idxx = linknum.index(idx)
                coordi = const_sche[d][lk][1]
                links[idxx][d+1][coordi] += 1

        return links

    def gen_population(linknum,teams,count):
        """
        Generate a bunch of construction lists to use in genetic algorithm
        """
```

```
    list = []
    for i in range(count):
        const_sche = gen_individual(linknum, teams)
        b = convert_to_construction(linknum,teams,const_sche)
        list.append(b)



    return list



def change_cap(construction,cap_init,length,lanes):
    """
    Return the new capacity of tranportation system under each phase of construction
    """
    ##i.e. [[21,[2,1,0,2],[0,1,2,0],[1,1,1,1]],[9,[1,1],[1,1],[0,0]]]
    capchange = dict()
    phasenum = len(construction[0]) - 1
    constsche = dict()
```

```
for i in range(len(construction)):
    index = construction[i][0] - 1
    capchange[index+1] = list()
    constsche[index+1] = list()
    for j in range(1,len(construction[i])):
        phase = construction[i][j]
        a = 0
        constlanes = 0
        for k in range(len(phase)):

            numworklanes = lanes[index] - phase[k]



            constlanes += phase[k]
            if phase[k] ==0:
                capmile = cap_init[index]/len(phase)
            elif numworklanes == 0:
                capmile = 1000000000
            else:
                capmile = 1450 * numworklanes
            a += capmile
```

```
                if a > 500000:
                    b = 10
                else:
                    b = a

                constsche[index+1].append(constlanes)
                capchange[index+1].append(b)


        capnew = []

        for s in range(len(construction[0])-1):
            cap_copy = copy.deepcopy(cap_init)
            capnew.append(cap_copy)
            for key in capchange:
                capnew[s][key-1] = capchange[key][s]
        return capnew

def diff_user_timecost(cap,edges,ODtable,tf_init):
    """
    Return the cost difference of user in system when under construction
    """
```

```
    tf = tf_init
    xue = frankwolfe(edges,ODtable,tf,cap)
    # the user_cost_orig is based on the origin network with no construction and same
OD trips
    cost_ue = cost(xue,tf,cap)
    user_cost = 0
    # c = []
    for i in range(len(xue)):
        user_cost += cost_ue[i]*xue[i]
        # c.append(cost_ue[i]*xue[i])
    # print(c)
    # print(cost_ue[22],cost_ue[25],cost_ue[42],cost_ue[64],cost_ue[66])
    # print(xue[22],xue[25],xue[42],xue[64],xue[66])




    diff = user_cost - user_cost_orig
    print(diff)
    return diff


def total_delay_timecost(construction,cap_init,length,edges,ODtable,tf_init):
    """
```

```
    Return the total cost of users in whole construction period
    """
    capnew = change_cap(construction,cap_init,length,lanes)
    peaktime= []
    work = []
    setuptime = [0]
    # construction =  [[23, [1, 1], [1, 1], [1, 1], [1, 1], [1, 1],[0,0]],
      # [26, [1, 1], [1, 1], [1, 1], [0, 0], [0, 0],[0,0]],
      # [43, [1, 1, 1, 1], [1, 1, 1, 1], [1, 1, 1, 1], [0, 0, 0, 0], [0, 0, 0, 0],[0, 0, 0, 0]],
      # [65, [1, 1], [1, 1], [0, 0], [0, 0], [0, 0],[0,0]], [67, [1, 1], [1, 1], [0, 0], [0, 0],
[0, 0],[0,0]]]
    for p in range(1,len(construction[0])):
      workload = 0
      for t in range(len(construction)):
        for m in range(len(construction[t][p])):
          workload +=construction[t][p][m]

      # print(workload)
      work.append(workload*20*2)

    for t in range(len(construction)):
      for m in range(len(construction[t][1])):
```

```
        if construction[t][1][m]!=0:
            setuptime[0] += 5*2



for p in range(2,len(construction[0])):
    setupsum = 0
    for t in range(len(construction)):
        for m in range(len(construction[t][p])):
            if construction[t][p][m] != 0 and construction[t][p-1][m] == 0:
                setupsum += 1


    setuptime.append(setupsum*5*2)
print(setuptime)


for i in range(len(work)):
    tmp = work[i] + setuptime[i]
    peaktime.append(round(tmp/teams))
# print(peaktime)
setuptotal = 0
for i in range(len(setuptime)):
    setuptotal += setuptime[i]/10
```

```
    tf = tf_init
    total_delay = 0
    rec = []
    for i in range(len(capnew)):
        cap = capnew[i]
        delay = diff_user_timecost(cap,edges,ODtable,tf_init) * peaktime[i]
        # the user_cost_orig is based on the origin network with no construction and
same OD trips
        # cap[i][0] stores the duration time of that stage of capacity of network
        # consider 2 peak hour in a day

        total_delay += delay
    return total_delay,setuptotal


def total_cost(construction,cap_init,tf_init,length,edges,ODtable):
```

```
    """
    Calculate the total cost of the construction, constitute of monetary user delay cost,
    work zone set up cost and accidental cost
    """
    capnew = change_cap(construction,cap_init,length,lanes)
    total_delay,setuptotal =
total_delay_timecost(construction,cap_init,length,edges,ODtable,tf_init)
    #print (total_delay)
    monetary_value = 21.04
    delay_cost = total_delay * monetary_value
    #print (delay_cost)
    workzone_setupcost = setuptotal * 4000#####@@@@@
    workzone_length = 0
    for i in construction:
        workzone_length += length[i[0]-1] * lanes[i[0]-1]
    workzone_buildcost = workzone_length * 500000#####@@@@
    #print (workzone_buildcost)
    accidental_cost = total_delay * 40 * 142000/100000000#####@@@@
    #print (accidental_cost)
    total_cost = delay_cost + workzone_setupcost + workzone_buildcost +
accidental_cost
    #print ((workzone_setupcost + workzone_buildcost)/delay_cost)
```

```
        return
total_cost,delay_cost,workzone_setupcost,workzone_buildcost,accidental_cost


    def fitness(construction):
        """
        Evaluate the fitness of individuals in population
        """


        total,delay_cost,workzone_setupcost,workzone_buildcost,accidental_cost        =
total_cost(construction,cap_init,tf_init,length,edges,ODtable)
        print(construction)


print(total_cost,delay_cost,workzone_setupcost,workzone_buildcost,accidental_cost)
        return total


    def grade(pop):
        """
        Get average fitness of a population, this is a measurement to evaluate the average
performance of population
        """
        summed = 0
        for i in range(len(pop)):
```

```
        summed += fitness(pop[i])
    return (summed/(len(pop)*1.0))


def evolve(pop,retain, random_select, mutate):
    """
    Use genetic algorithm to find new and better solution based on random generated
population
    """
    fitpop = []
    summed = 0
    for i in range(len(pop)):
        tmp = fitness(pop[i])
        fitpop.append((tmp,pop[i]))
        summed += tmp
    gradeavg = summed/len(pop)*1.0
    graded = [ x[1] for x in sorted(fitpop)]
    retain_length = int(len(graded)*retain)
    parents = graded[:retain_length]
    # randomly add other individuals to
    # promote genetic diversity
    for individual in graded[retain_length:] :
        if random_select > random.random():
```

```
        parents.append(individual)
    else:
        parents = parents
#####need a mutate




        # this mutation is not ideal, because it
        # restricts the range of possible values,
        # but the function is unaware of the min/max
        # values used to create the individuals,


# crossover parents to create children
parents_length = len(parents)
desired_length = len(pop) - parents_length
children = []
while len(children) < desired_length:
    maleidx = randint(0, parents_length-1)
    femaleidx = randint(0, parents_length-1)
    if maleidx != femaleidx:
        male = parents[maleidx]
        female = parents[femaleidx]
```

```
        child = []
        for i in range(len(male)):
            child.append([male[i][0]])
            for j in range(1,len(male[i])):
                half = round(len(male[i][j])/2)
                phase = male[i][j][:half] + female[i][j][half:]
                child[i].append(phase)
        # for stage in range()
        children.append(child)



parents.extend(children)
#print ("""""""""""""""""""new parents""""""""""""""""""")
#print (parents)
#print (""""new cost""""")
#print (grade(parents))
#print(""""new parents""""")
#print(parents)
for individual in parents:
    if mutate > random.random():
        pos_to_mutate_1 = randint(0, len(individual)-1)
        pos_to_mutate_2 = randint(1,len(individual[pos_to_mutate_1][1])-1)
```

```
        a = len(individual[0]) -1  #phase number
        b = 0 #lane number of mutate position to be calculated
        for i in range(a):
            b += individual[pos_to_mutate_1][1+i][0]

        sumlan = 0
        for i in range(a):
            if i != a-1:
                tmp = randint(0,b-sumlan)
                sumlan += tmp
                individual[pos_to_mutate_1][1+i][pos_to_mutate_2] = tmp
            else:
                individual[pos_to_mutate_1][1+i][pos_to_mutate_2] = b-sumlan

    return gradeavg,parents



def geneticA(linknum,tf_init,count,retain,random_select,mutate,teams):
    """
    Use genetic algorithm to determine the optimal workzone set up plan for the road,
    objective is to minimize the total cost of the construction
    """
```

```
pop = gen_population(linknum,teams,count)

#print (pop)

#print (grade(pop))

result = []

n = 10

i = 0

#while n > 0.01 and i < 5:

for m in range(4):

    gradeavg,newpop = evolve(pop,retain,random_select,mutate)

    old = gradeavg

    print("""old""")

    print(old)

    i += 1

    new = grade(newpop)

    print("""new""")

    print(new)

    n = ((new-old)/old)**2

    print (n)

    pop = newpop

    rec = old

    old = new

    graded  = [ (fitness(construction),construction) for construction in pop]
```

```
            graded = [ x[1] for x in sorted(graded)]
            best = graded[0]

best_total_cost,best_delay_cost,best_workzone_setupcost,best_workzone_buildcost,b
est_accidental_cost = total_cost(best,cap_init,tf_init,length,edges,ODtable)
            print(best_total_cost)

print(best_delay_cost,best_workzone_setupcost,best_workzone_buildcost,best_accide
ntal_cost)

result.append([rec,new,math.sqrt(n),best,best_total_cost,best_delay_cost,best_workzo
ne_setupcost,best_workzone_buildcost,best_accidental_cost])
        return result

    def write(outputfile,linknum,tf_init,count,retain,random_select,mutate,teams):
        """
        Write result in the text profile for analysis
        """
        out = open (outputfile, 'w')
        out.write("link number" + ';' + str(linknum) + '\n')
        out.write("population size" + ';' + str(count) + '\n')
        out.write("construction team number" + ';' + str(teams) + '\n')
```

```
    out.write("retain" + ';' + str(retain) + '\n')
    out.write("random select rate" + ';' + str(random_select) + '\n')
    out.write("mutation rate" + ';' + str(mutate) + '\n')
    result = geneticA(linknum,tf_init,count,retain,random_select,mutate,teams)
    out.write("iteration number" + ';' + "avg obj of old population" + ';'"avg obj of
new population" +';' + "ratio of change in obj" + ';' + "best construction plan" + ';' +
"total cost" + ';' + "construction finish time" + ';' + "user delay cost" + ';' + "workzone
setup cost" + ';' + "workzone build cost" + ';' + "accident cost" + ';')
    itr = 1
    for i in range(len(result)):
        out.write('\n')
        out.write(str(itr) + ';')
        itr += 1
        for j in range(len(result[0])-1):
            out.write(str(result[i][j]) + ';')
        out.write(str(result[i][len(result[0])-1]))
    out.close()



if __name__ == "__main__":
    path = "C:\\Users\\liu\\Desktop\\DATA24.xlsx"
    edges,tf_init,cap_init = readmethod.trans_excel_to_list_tuple(path)
```

```
path = "C:\\Users\\liu\\Desktop\\DATA24.xlsx"
edges,tf_init,cap_init = readmethod.trans_excel_to_list_tuple(path)



ODtable = odread.readod(path)
######
##important thing: should pay attention to FW decision variable to make sure
it won't change much in objective f######
######
lanes,length = readlanes.readlanes(path)
count = 500
teams = 6
user_cost_orig = 28085.2816301
# from testmin.py pay attention to odread for the od trips input data
linknum = [23,26,43,65,67]
retain = 0.2
random_select = 0.02
mutate = 0.3
outputfile = "C:\\Users\\LIu\\Desktop\\output11.txt"
write(outputfile,linknum,tf_init,count,retain,random_select,mutate,teams)
```

Appendix D: Read data methods.

```
import xlrd


def readlanes(path):
    book = xlrd.open_workbook(path)
    sheet = book.sheet_by_index(1)
    nrows_sheet = sheet.nrows
    length = sheet.col_values(1)[1:nrows_sheet]
    lanes = sheet.col_values(4)[1:nrows_sheet]
    return lanes,length


def readod(path):
    book = xlrd.open_workbook(path)
    sheet = book.sheet_by_index(3)
    ODtable = []
    for j in range(24):
        tmp = []
        for i in range(4):
            for m in range(5):
                cell = sheet.cell(1+j*7+i,m).value


                tmp.append(cell)
```

```
        tmp.append(sheet.cell(5+j*7,0).value)
        tmp.append(sheet.cell(5+j*7,1).value)
        tmp.append(sheet.cell(5+j*7,2).value)
        tmp.append(sheet.cell(5+j*7,3).value)
        for k in range(24):

            tpod = (j+1,k+1,tmp[k]/2*1.1)
            ODtable.append(tpod)

sheet1 = book.sheet_by_index(4)

return ODtable
```