Final Report

June 2016

SCHEDULING WORK ZONES IN MULTI-MODAL NETWORKS

Phase I: Scheduling Work Zones in Transportation Service Networks

SOLARIS Consortium, Tier 1 University Transportation Center
Center for Advanced Transportation Education and Research
Department of Civil and Environmental Engineering
University of Nevada, Reno
Reno, NV 89557

Pitu B. Mirchandani, Principal Investigator.
Dening Peng
School of Computing, Informatics, and Decision Systems Engineering
Arizona State University
Tempe, AZ 85287

DISCLAIMER:

The contents of this report reflect the views of the authors, who are responsible for the facts and accuracy of the information presented herein. This document is disseminated under the sponsorship of the U.S. Department of Transportation's University Transportation Centers Program, in the interest of information exchange. The U.S. Government assumes no liability for the contents or use thereof.

EXECUTIVE SUMMARY

The purpose of this project is to study the optimal scheduling of work zones so that they have minimum negative impact (e.g., travel delay, gas consumption, accidents, etc.) on transport service vehicle flows. In this project, a mixed integer linear programming model is developed to schedule work zones in transportation service (i.e., trucking service) networks. The model schedules lane closures of links that need maintenance in a transportation network. When some lanes of a link are closed, the available capacity of that link is reduced. In the assumed scenarios, based on the available capacities on the links, given origin-destination (OD) flow demands are provided *system optimal* routing through the network to achieve total minimum flow cost for all the OD pairs. The link flow cost function is piece-wise linear such that regular flow cost is incurred for all the units flowing through the link at free flow while extra congestion cost is incurred for the units exceeding the link's nominal capacity. The goal is to schedule the work zones, that is, the corresponding lane closures, so that all maintenance work can be completed before a given completion date while the total flow cost over the project period is minimized. An innovative randomized fix-and-optimize (RFO) heuristic is developed to solve the problem efficiently. Various networks are tested for the performance comparison between CPLEX and RFO. It is concluded that the RFO heuristic is able to obtain optimal or near-optimal solutions with much less time than CPLEX. In Phase 2 of the project planned in the next period, *Scheduling Work Zones in Commuter Transportation Networks*, commuters from their origins to their destinations are included in the work zone scheduling problem.

Keywords: Transportation Service Network, Work Zone Scheduling; Mixed Integer Linear Programming (MIP); Randomized Fix-and-optimize Heuristic (RFO)

Acknowledgements: The research reported was partially supported by SOLARIS, a U.S. Department of Transportation's Tier 1 University Transportation Center led by the University of Nevada at Reno, and partially by the Arizona State University, Tempe, Arizona.

TABLE OF CONTENTS

TABLE OF CONTENTS(CONT'D)

Appendix

LIST OF TABLES

LIST OF FIGURES

1. Introduction

The repair and maintenance of road network results in "work zones", where some lane segments of a link are out of commission for a predicted period of time until the work is completed. Temporary link capacity reductions caused by lane closure can result in significant delays of commuters and transport services. Americans lose 3.7 billion hours and 2.3 billion gallons of fuel every year sitting in traffic jams. Work zones are estimated to cause about 10% of overall congestion which translates into annual fuel loss of over 700 million US dollars (FHWA, 2013).

Road construction companies and transportation management agencies do a reasonable job of coordinating work zone activities after the work zone is initiated through appropriate scheduling and staging of day-to-day and week-to-week operations so that the overall cost is contained, while safety and traffic congestion is not overly affected during peak periods. While a single, or few widely scattered concurrent work zones, will not have a large effect upon travel patterns, several work zones that are spatially and temporally close together, and which affect large flows of traffic, may result in traffic patterns that are both costly to the travelers and vehicle-based services, resulting in significant negative environmental and safety consequences. In 2010, there were 87,606 crashes in work zones and 526 of these crashes were fatal (FHWA, 2013). While large trucks accounted for only 4% of all registered vehicles in the United States, 27% of work zone fatal crashes involved at least one large truck (FHWA, 2013). Through proper scheduling of work zones with respect to the spatial locations in the network and the time periods of the work zones, a reduction of negative impacts is expected.

In this project, we consider a network that provides transport logistics services (e.g., freight and parcel delivery) from origins to destinations, where the overall objective is to minimize system transportation costs. The traveling cost of a link is treated as the cost in general sense, which can be interpreted as combinations or functions of travel time, monetary cost, and road unsafety. Origin-Destination (OD) flows of vehicles are reactive to the network topology changes resulted from temporary lane closures, which means the routes chosen by each vehicle could change in response to work zone activities to minimize their flow costs. Since this project focuses particularly on trucks and other delivery vehicles, we use the term "trucks" instead of the generic "vehicles" term. The model developed finds the optimal lane closure schedule that has the minimum negative impact (e.g. gas usage, congestion) on the trucks' flows in the network.

2. Related Literature Review

Work zone planning is a challenging task since there are multiple parties involved and more than many factors need to be taken into consideration. Bayraktar and Hastak (2009) reviewed the factors impacting the success of work zone projects. They modeled the relationships between the goals of the project

stakeholders and public satisfaction of the project using Bayesian belief networks. The model was aimed to assist highway agencies in developing suitable contracting strategies considering 52 interrelated factors impacting the success of work zone projects, which were grouped into four categories (contract characteristics, motorist issues, public issues, and resource issues). Despite the comprehensive list of factors taken into account, the model can only help prepare bids and not help to actually schedule the work zones.

Most of the literature related to the problem of this project can be grouped into three categories. The first category includes research that investigates the long term network rehabilitation planning problem with the objective of maintaining the roads in good condition with least cost in different aspects. For example, Chu and Chen (2012) developed a bi-level hybrid dynamic model in which the upper level problem decides the optimal threshold for each road that triggers maintenance action and the lower level problem solves the user equilibrium problem. These two levels of problems are connected by the road deterioration function which models the effects of traffic loads on a road and the impacts of road roughness on users' traveling cost. This type of research considers network-wide maintenance planning over a relatively long period of time (a year or longer). By assuming the project period is much shorter than the planning horizon, they omitted the impact of temporary link capacity reductions on traffic flow caused by the maintenance work. However, this assumption is not always reasonable especially for the maintenance work like resurfacing sets of links which would take months or longer. When the length of project period is comparable to the planning horizon, it is necessary to consider the effect of temporary link capacity reductions and to schedule the work zones in the way that minimizes the negative impacts on traffic flows.

Research in the second category focuses on developing operational strategies for work zone scheduling on a highway segment or a local arterial. Some research in this category has studied the short term work zone scheduling with time horizons less than a day. This research focuses on optimizing the workzone planning of a single link but does not consider the impact of diverting traffic and possible resulted from workzones to other links that are connected to or close to the focusing link; see e.g., works of Meng and Weng (2013), Tang and Chien (2008), and Jiang and Adeli (2003). However, in reality, as long as traffic congestion exists and there are alternative routes available, some portion of the traffic will divert to other routes which will affect the traffic on those alternative routes. Chien and Tang (2014) proposed a genetic algorithm to optimize the work zone length and start time in a day of the maintenance work on a highway stretch. The optimal schedule minimizes the total cost to the agencies conducting the maintenance plus the cost to the road users. Even though the temporary link capacity reductions, and resulting increased road user cost, and possible traffic diversion, were modeled, only one alternative route for the diverted traffic was considered. Often there are more than two lanes for some segments of highway, but Chien and Tang (2014) did not explicitly explore different lane closing scenarios. Schroeder and Rouphail (2010) compared different lane closure scenarios and discussed the operational impacts of freeway work zones on traffic.

Their approach can only compare every limited number of scenarios since each scenario requires extensive analysis. Summarizing, the research in this category focuses on scheduling work zones on single links and has very limited or no consideration on the impact of traffic diversion resulting from multiple link capacity reductions.

The third category consists research that studied the scheduling of network expansion projects. These research specifically considered the flow pattern changes caused by the increase of link capacities or the addition of new links over the planning time horizon. This research topic is closely related to the network design problem, which selects among a set of candidate links to be added to a network with budget constraints, so as to achieve lowest total cost at users' equilibrium state or system optimum. It is an extension of the network design problem since the addition of the chosen links need to be scheduled, and possible traffic flow pattern changes need to be evaluated after the addition of each link. Fontaine and Minner (2014) developed a mixed-integer programming model to select and schedule network expansion projects with minimum total project cost and system optimum flow cost, and solved it using Bender's decomposition. Bagloee and Asadi (2015) presumed the set of network expansion projects were given and only one of these projects could be worked on at a time, and studied the network expansion scheduling problem as a traveling sales man problem to determine the optimal sequence of the expansion projects. The inter-dependency of the expansion projects was evaluated using the artificial neural network model, so that the "cost" of "moving" from one expansion project to another could be computed. Gao et al. (2011) combined the problems of road maintenance and road expansion planning, and developed a mixed-integer, nonlinear, bi-level model that scheduled the repair or expansion of every road with budget constraints. In the model proposed, the road capacity increase after maintenance and expansion were considered, and the road degradation process was modeled. General Bender's decomposition method was applied to obtain the optimal maintenance and expansion schedule that gave the minimum total users' cost at equilibrium state. Although literature reviewed in this category modeled the capacity increase after the maintenance or expansion, they did not consider the link capacity reductions during the time period when these activities were being performed.

Only a handful of works considered the impact on traffic over the network due to multiple work zones and they comprised the fourth category. Orabi and El-Rayes (2012) developed a complex model with three genetic algorithm based modules – scheduling, network performance, and user savings, to select and prioritize rehabilitation projects, subject to budget constraints. Lee (2009) proposed a work zone scheduling model which considered the routing-changing behavior of road users. The schedule was optimized with an ant colony algorithm, where the users' equilibrium under each schedule scenario was obtained through simulations using VISSIM software. Hosseininasab and Shetab-Boushehri (2015) studied the work zone scheduling problem as a time-dependent network design problem. They formulated the problem as bi-level

programming models, and used genetic algorithm to obtain the link maintenance schedule that gave the minimum total traveling cost at equilibria over the planning time horizon. All the three of Orabi and El-Rayes (2012), Lee (2009) and Hosseininasab and Shetab-Boushehri (2015) did not explicitly discuss partial link capacity reductions resulting from work zones. Zheng et al. (2014) assumed the link capacity would reduce by 50% in their decision model developed. However, a link might have more than two lanes and it is not always true or optimal to close half of the lanes at a time for maintenance. Ma et al. (2004) developed a hybrid simulation methodology with genetic algorithm to schedule multiple lane closures with minimum total traffic delay of the network. However, the flexible lane-level maintenance scheduling required high computation effort for the solution approach proposed in Ma et al. (2004). For a problem instance of scheduling the maintenance of 20 lanes, it took more than 120 hours.

In this project, a mixed integer linear programming model is developed to schedule work zones in the perspective of networks that are used by trucks fulfilling transportation services. A randomized fix-and-optimize (RFO) heuristic is developed to solve the problem efficiently. In Section 3 provides a complete description of the work zone scheduling model for transportation service networks (WZS-TS) and its computational implementation. Section 4 describes the details of the heuristic developed and illustrates its performance by comparing it with an exact CPLEX-based approach on various test cases. Section 5 provides conclusions and gives some directions of possible future research.

3. Work Zone Scheduling in Transportation Service Networks (WZS-TS) Model

3.1 Piecewise Linear Cost Structure

In transportation service networks, linear flow cost structure is commonly used for minimum flow cost problems, where we set the cost of travelling on a link linear with respect to the total flow on that link when the total flow is smaller than or equal to the available nominal capacity of the link. In applications where the demand on a link is more than the available capacity, the excess flow is either detoured or given a very high cost for using the link thereby softening the hard capacity constraint. In this project we will use the latter approach by modeling the cost as a piece-wise linear cost to approximate the traffic condition aggravation effects in transportation networks. With the piece-wise linear cost functions the work zone scheduling model, developed later, can be solved by commercial solvers like CPLEX, the performance of which can be used to compare with the new heuristic developed later in the project.

In the workzone scheduling model, it is assumed that there are Origin-Destination (OD) flow demands of trucks every time period (e.g., peak period of a day). Each truck can choose its own route to minimize its travel cost and is treated as a unit of flow. When a link is under maintenance, one or more lanes are closed, inducing the temporary link capacity reductions, and thus the link cannot fully serve the flows

satisfactorily. That is likely to cause the current flow on the link to exceed the available nominal capacity, incurring the expensive extra flow cost. Suppose a link has three lanes and all three lanes have the same "flow capacity" u, Figure 1 below illustrates the relation between the flow units and flow cost for different lane closure situations:

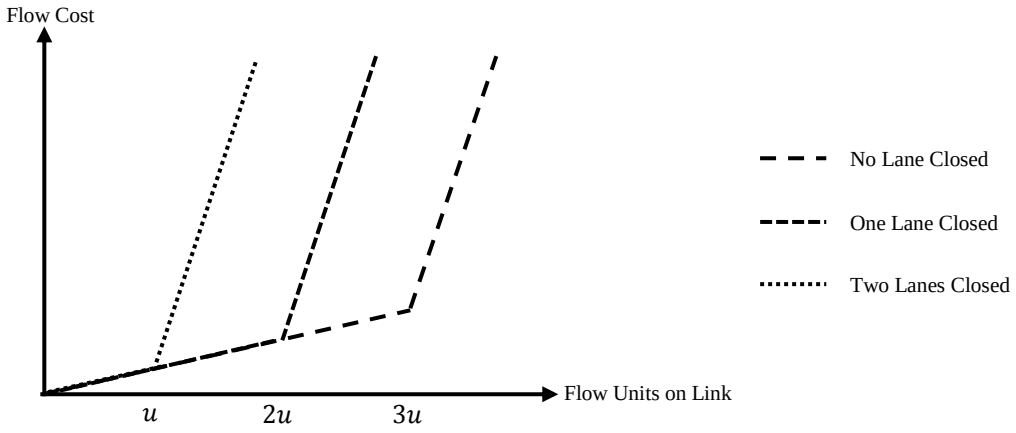


FIGURE 1: Three-Lane Link Flow Cost Curve.

When two lanes are closed for maintenance, the available capacity of the link is u. If the units of flows on the link is more than u, then the extra flow cost will be incurred. This is why the slope of the cost curve is much steeper when the flow units is more than u for the case of two-lane closure. Same cost curve pattern can be seen in the cases of no-lane closure and one-lane closure. When some of the lanes in a link is closed for maintenance, some of the flow that is originally on this link could divert to other paths and links to reach the destination with lower total, that is, the network flows are *reactive* to the maintenance schedules.

3.2 Model Formulation

The objective of the model is to schedule the lane closures so that all links that need maintenance are repaired before a given completion date for the whole network, while the total flow cost for all the OD pairs, which includes regular flow cost and extra congestion flow cost, is minimized over the project period. Denote c_i as the regular unit flow cost of link i, y_{ikt} as the flow units of OD pair k that flow through link i on day t, and z_{it} as the difference between flow units of all the OD pairs that flow through link i and the available capacity of link i on day t, the objective function (1) is formulated as $min \sum_{i \in E}\{\sum_{t=1}^{t=T}[c_i * (\sum_{k \in OD} y_{ikt}) + z_{it}\rho c_i]\}$, where E is the set of links, OD is the set of OD demand, and T is the common completion date of all the maintenance work. ρ is the congestion flow cost multiplier which makes the extra unit flow cost ρc_i much larger than the regular unit flow cost c_i. Note that z_{it} is non-negative in the sense

that it will have positive value only when the total flow units on link i exceed the available capacity and it will be zero otherwise.

Binary variables s_{imt} are introduced to indicate whether the repair of the m^{th} lane of link i starts on day t, and $s_{imt} = 1$ if it is. The WZS-TS model assumes once a lane is closed for repair, it cannot open to serve the flows until its repair is completed. Hence we have the constraints (2): $\sum_{t=1}^{t=T} s_{imt} = 1$ for $\forall i \in R$ and $\forall m \in [1, n_i]$, where R is the set of links that need repair and n_i is the number of lanes in link i. This set of constraints force every lane of all the links that need repair to have one and only one repair start date.

To indicate whether m^{th} lane of link i is closed for maintenance on day t, binary variables x_{imt} are added to the model. x_{imt} equal to 1 if the m^{th} lane of link i is closed for maintenance on day t. Let p_i be the number of days needed to repair a lane of link i, we formulate the constraints (3) $\sum_{t=1}^{t=T} x_{imt} = p_i$ for $\forall i \in R$ and $\forall m \in [1, n_i]$ to ensure the repair on all the links be completed by the common completion date T. Since each lane of the links needing maintenance have one and only one repair start date and the number of days needed to repair a lane is given, whether a lane is closed or not on a day is determined once the repair start date of that lane is determined. And thus, we develop the set of constraints (4) $x_{imt} = \sum_{a=max(t-p_i+1,1)}^{a=t} s_{ima}$ for $\forall i \in R, \forall t \in T$ and $\forall m \in [1, n_i]$ to make sure that once a lane is closed for repair, it will not open to serve the flows until the repair work on this lane is finished and that it will be open on other dates. Constraints (5) $\sum_{t=1}^{t=T} s_{imt} = 0$ for $\forall i \notin R, \forall m \in [1, n_i]$ and (6) $\sum_{t=1}^{t=T} x_{imt} = 0$ for $\forall i \notin R$ and $\forall m \in [1, n_i]$ are added to the model so that all the lanes of links that do not need repair will not have maintenance start date and will be open to serve the flows throughout the project period.

For each OD pair on each day, flow conservation constraints, consisting of three groups, are needed. The first group of constraints makes sure the total incoming flow units minus the total outgoing flow units equal to the OD demand for the origin node of the OD pair. Let D_k be the demand of OD pair k, the first part is formulated as (7) $D_k = \sum_{\{i:E_i^- = OD_k^-, i \in E\}} y_{ikt} - \sum_{\{j:E_j^+ = OD_k^-, j \in E\}} y_{jkt}$ for $\forall k \in OD, \forall t \in [1,\ T]$, where OD_K^- is the origin node of OD pair k, E_i^- is the head node of link i and E_j^+ is the tail node of link j. The second group ensures the total outgoing flow units minus the total incoming flow units equal to the demand of OD pair k for its destination node and is formulated as (8) $D_k = \sum_{\{i:E_i^+ = OD_k^+, i \in E\}} y_{ikt} - \sum_{\{j:E_j^- = OD_k^+, j \in E\}} y_{jkt}$ for $\forall k \in OD, \forall t \in [1,\ T]$, where OD_K^+ is the destination node of OD pair k, E_i^+ is the tail node of link i and E_j^- is the head node of link j. For the rest of the nodes, other than origin and destination nodes of OD pair k, the total incoming flows on the node from the origin of OD pair k should equal to the total outgoing flows from the node to the destination of the OD pair k. This is the third group

of the flow conservation constraints and it is formulated as (9) $\sum_{\{i:E_i^-=l,i\in E\}} y_{ikt} = \sum_{\{j:E_j^+=l,j\in E\}} y_{jkt}$ for $\forall l \in N, \forall t \in [1,T], \forall k \in \{k: OD_k^- \neq l\} \cap \{k: OD_k^+ \neq l\}$, where N is the set of nodes in the network.

In addition, binary variables v_{imt} are introduced to calculate the increased lane capacities and v_{imt} equals to 1 if lane m of link i is repaired before day t, since it is obvious that when a segment of road is repaired, the road condition should be improved and the capacity should increase. Constraints (10) $v_{imt} = \sum_{a=1}^{a=t-p_i} s_{ima}$, for $\forall i \in R, \forall m \in [1,n_i]$ and $\forall t \in [p_i+1, T]$ determine the values of v_{imt} by values of s_{imt}. In the constraints, the date ranges from $p_i + 1$ to T since the lane will be repaired and open to serve the flows on day $p_i + 1$ the earliest, because even if the maintenance starts on day 1, it would take p_i days to complete the repair work for this lane. Constraints (11) $v_{imt} = 0$, for $\forall i \in R, \forall m \in [1,n_i]$ and $\forall t \in [1,p_i]$ make sure each lane of the links that need maintenance stay in the status of not repaired in the first p_i days. And constraints (12) $v_{imt} = 0$, for $\forall i \notin R, \forall m \in [1,n_i]$ and $\forall t \in [1,T]$ force lanes of links that do not need repair stay in the status of not repaired throughout the project period.

Let θ be the percentage increase in lane capacity after the lane is repaired, and let u_i be the capacity of a lane of link i, the available capacity of link i on day t is $\left(n_i - \sum_{m=1}^{n_i} x_{imt} + \sum_{m=1}^{n_i} \theta v_{imt}\right) u_i$. Hence the values of z_{it} are determined by constraints (13) $\sum_{k\in OD} y_{ikt} - \left(n_i - \sum_{m=1}^{n_i} x_{imt} + \sum_{m=1}^{n_i} \theta v_{imt}\right) u_i \leq z_{it}$ and $z_{it} \geq 0$ for $\forall i \in E$ and $\forall t \in [1,\ T]$, where $\sum_{k\in OD} y_{ikt}$ are the total flow units from all OD pairs on link i on day t.

Because of the introduction of z_{it}, flows can exceed the available capacity. Hence it is needed to make sure there won't be flows on links with all lanes closed for maintenance, that is, entirely closed links cannot serve any flow. For this reason, the set of variables w_{it} are added into the model, the values of which equal to 1 if all the lanes of link i are closed on day t. Constraints (14) $1 - w_{it} \leq n_i - \sum_{m=1}^{n_i} x_{imt}$ for $\forall i \in R$ and $\forall t \in [1,\ T]$ make sure w_{it} equal to 1 when all the lanes of link i are closed on day t, and constraints (15) $n_i - \sum_{m=1}^{n_i} x_{imt} \leq n_i(1 - w_{it})$ for $\forall i \in R$ and $\forall t \in [1,\ T]$ force w_{it} to be 0 if at least one lane of link i is open on day t. Finally, constraints (16) $\sum_{k\in OD} y_{ikt} \leq \sum_{k\in OD} D_k\,(1 - w_{it})$ for $\forall i \in R$ and $\forall t \in [1,\ T]$ prevent links with all lanes closed from serving flows.

The sets, parameters, and variables mentioned in the model description above are summarized in Table 1 below:

TABLE 1: Notations

Term	Definition
Sets	
N	Node set of the network
E	The set of existing links in the network

$\boldsymbol{R}$	The set of existing links that need to be repaired in the network, $R \subseteq E$
$\boldsymbol{OD}$	The set of Origin-Destination pairs of flows
Parameters	
$\boldsymbol{T}$	Completion date for all the maintenance work (the earliest start date of a work zone is Day 1)
$\boldsymbol{n_i}$	Number of lanes of link i
$\boldsymbol{u_i}$	Capacity of a lane of link i
$\boldsymbol{c_i}$	The regular flow cost incurred by one-unit flow on link i per day
$\boldsymbol{p_i}$	The number of days needed to repair a lane of link i
$\boldsymbol{\rho}$	Extra flow cost multiplier, ρc_i is the extra flow cost incurred by the available link capacity being one unit less than the flow demand on link i
$\boldsymbol{\theta}$	Percentage of lane capacity increased after maintenance
$\boldsymbol{D_k}$	Flow demand of OD pair k
Variables	
$\boldsymbol{s_{imt}}$	Binary variable indicating whether to repair on the m^{th} lane of link i starts on day t. If repair work starts on day t, $s_{imt} = 1$; otherwise, $s_{imt} = 0$
$\boldsymbol{x_{imt}}$	Binary variable indicating whether the m^{th} lane of link i is closed for maintenance on day t, if it is closed, $x_{imt} = 1$; otherwise $x_{imt} = 0$
$\boldsymbol{y_{ikt}}$	The flow units incurred by the Origin-Destination (OD) flow of OD pair k on link i on day t
$\boldsymbol{z_{it}}$	Flow units on link i exceeding the available capacity of the link on day t. If the available capacity of link i on day t is less than the total flow units on link i, z_{it} equals to the difference between the available capacity and total flow on link i; otherwise $z_{it} = 0$
$\boldsymbol{w_{it}}$	Binary variable indicating whether all the lanes of link i on day t are closed, if it is, $w_{it} = 1$; otherwise $w_{it} = 0$
$\boldsymbol{v_{imt}}$	Binary variable indicating whether the m^{th} lane of link i is repaired before day t, if it is, $v_{imt} = 1$, otherwise 0; for all the links that don't need maintenance, $v_{imt} = 0$ all the time

The complete model of work zone scheduling in transportation service networks (WZS-TS) is presented in Figure 2, where the number of each expression matches the bracketed numbers in the model description earlier in this subsection.

$$min \sum_{i\in E}\{\sum_{t=1}^{t=T}[c_i * (\sum_{k\in OD} y_{ikt}) + z_{it}\rho c_i]\} \quad (1)$$

$$\sum_{t=1}^{t=T} s_{imt} = 1, \qquad \forall i \in R, \forall m \in [1, n_i] \quad (2)$$

$$\sum_{t=1}^{t=T} x_{imt} = p_i, \qquad \forall i \in R, \forall m \in [1, n_i] \quad (3)$$

$$x_{imt} = \sum_{a=max(t-p_i+1,1)}^{a=t} s_{ima}, \qquad \forall i \in R, \forall t \in T, \forall m \in [1, n_i] \quad (4)$$

$$\sum_{t=1}^{t=T} s_{imt} = 0, \qquad \forall i \notin R, \forall m \in [1, n_i] \quad (5)$$

$$\sum_{t=1}^{t=T} x_{imt} = 0, \qquad \forall i \notin R, \forall m \in [1, n_i] \quad (6)$$

$$D_k = \sum_{\{i:E_i^- = OD_k^-, i\in E\}} y_{ikt} - \sum_{\{j:E_j^+ = OD_k^-, j\in E\}} y_{jkt}, \qquad \forall k \in OD, \forall t \in [1,\ T] \quad (7)$$

$$D_k = \sum_{\{i:E_i^+ = OD_k^+, i\in E\}} y_{ikt} - \sum_{\{j:E_j^- = OD_k^+, j\in E\}} y_{jkt}, \qquad \forall k \in OD, \forall t \in [1,\ T] \quad (8)$$

$$\sum_{\{i:E_i^- = l, i\in E\}} y_{ikt} = \sum_{\{j:E_j^+ = l, j\in E\}} y_{jkt}, \quad \forall l \in N, \forall t \in [1,T], \forall k \in \{k: OD_k^- \neq l\} \cap \{k: OD_k^+ \neq l\} \quad (9)$$

$$v_{imt} = \sum_{a=1}^{a=t-p_i} s_{ima}, \quad \forall i \in R, \forall m \in [1, n_i], \forall t \in [p_i + 1, T] \quad (10)$$

$$v_{imt} = 0, \quad \forall i \in R, \forall m \in [1, n_i], \forall t \in [1, p_i] \quad (11)$$

$$v_{imt} = 0, \quad \forall i \notin R, \forall m \in [1, n_i], \forall t \in [1, T] \quad (12)$$

$$\sum_{k \in OD} y_{ikt} - \left(n_i - \sum_{m=1}^{n_i} x_{imt} + \sum_{m=1}^{n_i} \theta v_{imt}\right) u_i \leq z_{it}, \quad \forall i \in E, \forall t \in [1,\ T] \quad (13)$$

$$1 - w_{it} \leq n_i - \sum_{m=1}^{n_i} x_{imt}, \quad \forall i \in R, \forall t \in [1,\ T] \quad (14)$$

$$n_i - \sum_{m=1}^{n_i} x_{imt} \leq n_i (1 - w_{it}), \quad \forall i \in R, \forall t \in [1,\ T] \quad (15)$$

$$\sum_{k \in OD} y_{ikt} \leq \sum_{k \in OD} D_k (1 - w_{it}), \quad \forall i \in R, \forall t \in [1,\ T] \quad (16)$$

$$w_{it} = 0, \quad \forall i \notin R, \forall t \in [1, T] \quad (17)$$

$$w_{it} \in \{0, 1\}, \quad \forall i \in E, \forall t \in [1, T] \quad (18)$$

$$s_{imt}, x_{imt}, v_{imt} \in \{0, 1\}, \quad \forall i \in E, \forall m \in [1, n_i], \forall t \in [1,\ T] \quad (19)$$

$$z_{it} \geq 0, \quad \forall i \in E, \forall t \in [1, T] \quad (20)$$

$$y_{ikt} \geq 0, \quad \forall i \in E, \forall k \in OD, \forall t \in [1, T] \quad (21)$$

FIGURE 2: Work Zone Scheduling in Transportation Service Networks (WZS-TS) Model.

The WZS-TS model possesses the features of both scheduling models and multi-commodity flows models. For each feasible schedule on each day, there is a multi-commodity flows problem over the network based on links' available capacities after the scheduled lane closures. As shown in Figure 3, with lane closure schedules fixed, variables x_{imt}, v_{imt}, and w_{it} become parameters x^o_{imt}, v^o_{imt}, and w^o_{it} that define the links' available capacities for each day, which is calculated as $\left(n_i - \sum_{m=1}^{n_i} x^o_{imt} + \sum_{m=1}^{n_i} \theta v^o_{imt}\right) u_i$ in Constraint (13'). And the remaining problem is a multi-commodity flows problem for each day over the planning time horizon. In the multi-commodity flows subproblem of WZS-TS, the available link capacity can be exceeded with a very high extra flow cost. This is different from the traditional multi-commodity flows problem, where link capacity constraints are hard constraints and the total amount of flows on the link has to be less than or equal to the link's capacity.

$$min \sum_{i \in E} \{\sum_{t=1}^{t=T} [c_i * (\sum_{k \in OD} y_{ikt}) + z_{it} \rho c_i]\} \quad (1)$$

$$D_k = \sum_{\{i: E_i^- = OD_k^-, i \in E\}} y_{ikt} - \sum_{\{j: E_j^+ = OD_k^-, j \in E\}} y_{jkt}, \quad \forall k \in OD, \forall t \in [1,\ T] \quad (7')$$

$$D_k = \sum_{\{i: E_i^+ = OD_k^+, i \in E\}} y_{ikt} - \sum_{\{j: E_j^- = OD_k^+, j \in E\}} y_{jkt}, \quad \forall k \in OD, \forall t \in [1,\ T] \quad (8')$$

$$\sum_{\{i: E_i^- = l, i \in E\}} y_{ikt} = \sum_{\{j: E_j^+ = l, j \in E\}} y_{jkt}, \quad \forall l \in N, \forall t \in [1, T], \forall k \in \{k: OD_k^- \neq l\} \cap \{k: OD_k^+ \neq l\} \quad (9')$$

$$\sum_{k \in OD} y_{ikt} - \left(n_i - \sum_{m=1}^{n_i} x^o_{imt} + \sum_{m=1}^{n_i} \theta v^o_{imt}\right) u_i \leq z_{it}, \quad \forall i \in E, \forall t \in [1,\ T] \quad (13')$$

$$\sum_{k \in OD} y_{ikt} \leq \sum_{k \in OD} D_k (1 - w^o_{it}), \quad \forall i \in R, \forall t \in [1,\ T] \quad (16')$$

$$z_{it} \geq 0, \quad \forall i \in E, \forall t \in [1, T] \quad (20')$$

$$y_{ikt} \geq 0, \quad \forall i \in E, \forall k \in OD, \forall t \in [1, T] \quad (21')$$

FIGURE 3: WZS-TS Model with a Feasible Schedule.

3.3 Computational Implementation

Even et al. (1975) proved that the decision version of the multi-commodity flow problem is NP-Complete even for only two commodities and unit capacities. By setting $T = 1$ and $R = \emptyset$, the problem discussed in this project is converted to a multi-commodity flow problem, which proves that this problem can be reduced from multi-commodity flow problem in polynomial time. Thus the WZS-TS problem is strongly NP-Hard.

The WZS-TS model is programmed in C++ with IBM® ILOG® CPLEX® Concert Technology. Some preliminary experiments are conducted and the results indicate the non-polynomial nature of the problem. To give an example, with a computer of 3.7 GHz quad-core CPU and 24.0 GB memory, it takes 0.51 second to solve a problem instance of 4 nodes, 12 links, 30 lanes, 12 OD pairs, 25 days of project completion period with all links needing repair; while to solve a problem instance of 16 nodes, 48 links, 108 lanes, 16 OD pairs, 27 days of project period with 50% of the links needing maintenance, CPLEX still has a 32% optimality gap after 14 hours of computation. Therefore, it is clear an efficient heuristic to solve the problem quickly with satisfactory accuracy is needed.

4. Solution Approach

4.1 Randomized Fix-and-Optimize (RFO) Heuristic

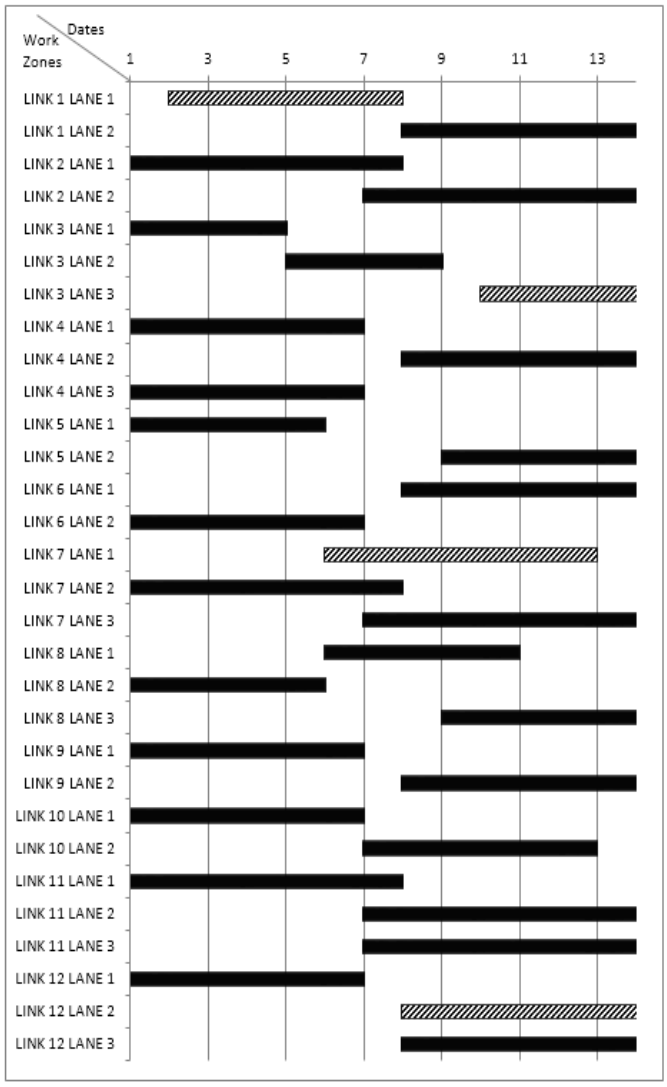
There are two layers of problems that constitute the problem of work zone scheduling in transportation service networks. The upper layer is the scheduling problem which decides the repair start date for each lane of the links that need maintenance. The lower layer is a series of multi-commodity flow problems based on the available capacities of links on each day, which is determined by the current lane closures. Once the schedule is set, solving the multi-commodity flow problems for each day is a relatively easy problem since the flow variables are all continuous variables. And thus the solution approach proposed in this project focuses on the upper layer of obtaining good work zone schedules.

To motivate the heuristic, suppose at a point in the algorithmic process we obtain a feasible schedule that has some aspects similar to the optimal schedule. For example, Figure 4 on the next page gives a comparison between the Gantt charts of the optimal schedule and one of the feasible schedules obtained for a small test network of 4 nodes, 12 links and 12 OD pairs. The vertical axis shows the lanes of links that need maintenance and the horizontal axis shows the date during the project period. Each bar represents the time period when a lane is closed for maintenance and cannot be used to serve the OD flows. For example, in the optimal schedule, Lane 1 of Link 2 is closed on Day 1 and will be reopen on Day 8, and Lane 2 of Link 2 will be closed from Day 7 to Day 13. Hence this two-lane link will have one lane available from

Day 1 to Day 6 and from Day 8 to Day 13. On Day 7 Link 2 is not available to serve any flows since both of the two lanes are closed.

From the Gantt charts we can see that the feasible schedule has lane closures of Link 1, 3, 7, and 12 different from the optimal schedule. If we only optimize the lane closure schedules of these four links and fix the schedules of all the other links, the problem size will be much smaller and the time needed to solve the problem instance will reduce dramatically since there are much fewer integer variables to go through in the branch-and-bound process performed by solvers like CPLEX. This observation leads to the adoption of the *fix-and-optimize* heuristic as the core of the solution approach

Optimal Schedule



Some Feasible Schedule

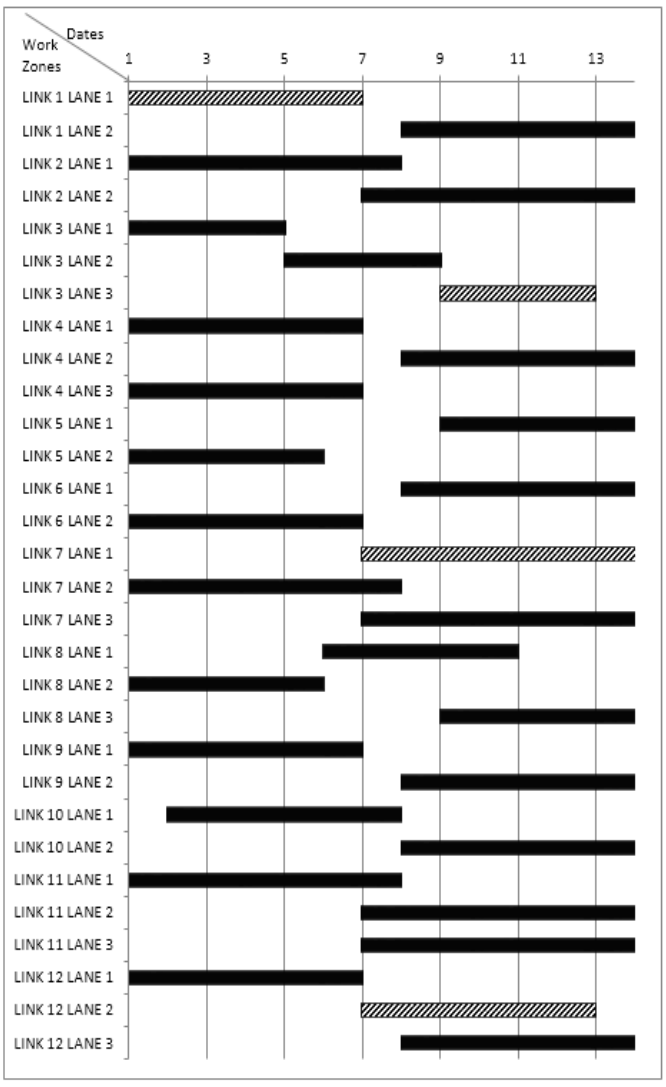


FIGURE 4: Schedule Comparison.

The fix-and-optimize heuristic was first introduced by Helber et al. (2010). It is an iterative optimization-based heuristic developed to solve the multi-level capacitated lot sizing problem which is a mixed integer program. The basic process of the fix-and-optimize heuristic is to partition the integer variables into subsets, based on an initial solution, and then optimize the values of a subset of integer variables together with all continuous variables while the values of the other integer variables in other subsets are fixed (this is called a subproblem of the fix-and-optimize procedure). If the new objective function value is better than current best objective value, then the current candidate optimal values are updated; iterate this process for other subsets of variables until a specified stopping criteria is met. The integer variables were decomposed into subsets based on the descending order on cost of each product in the lot-sizing problem, since usually a quite reasonable schedule was found after the first round of the product-oriented decomposition.

In the problem of scheduling work zones in transportation service networks, the relation among work zones is more complex than that among products in the capacitated lot-sizing problem. Products just compete with each other for resources (machine hours) in the capacitated lot-sizing problem. On the other hand, in the WZS-TS problem there are no resource constraints that work zones compete for, but instead the work zones affect the capacity of the network to serve the OD demands. Therefore, only the schedules that consider all or many work zones will have the lowest increase in total flow cost, because OD demands happen over the whole network and each OD pair has network-wide minimum cost routing. This means applying fix-and-optimize heuristic with small subsets of work zones (one or two links) will hardly find satisfactory schedules since it only considers the maintenance of a few links at a time.

However, if the size of the work zone subsets is large, the size of each fix-and-optimize subproblem will also be large and it would take long time to solve. To mitigate the conflict between solution quality and solving time length, we developed the fix-and-optimize procedure with varying subset sizes and used a truncated branch-and-bound method.

Initially, CPLEX tries to solve the entire problem within a given time limit (e.g. 60 seconds). If the problem is solved optimally, then the optimal schedule will be output and the program will terminate. If the problem is not solved optimally, the best feasible schedule obtained so far will be stored and used as the initial feasible solution for the fix-and-optimize procedure. A feasible schedule should be able to both complete all the maintenance work before the specified completion date and make sure each OD demands can be met.

The randomized fix-and-optimize (RFO) iteration starts with randomly dividing links that need maintenance into two subsets and solving each fix-and-optimize subproblem (FO subproblem) with a specified time limit. A RFO iteration is finished when the schedules of all the generated subsets of links are

optimized. The RFO will be performed for a preset number of iterations and if any of the FO subproblems is not solved within the time limit in the last iteration, the RFO will enter a new stage where the number of subsets which the links to repair are randomly divide into is three. The RFO proceeds similarly in stages with more subsets of links and each RFO iteration is performed the same way as it is in the initial stage when there are only two subsets.

The reason of randomly grouping links that need maintenance into subsets is because we do not know the set of links with schedules that are different from the optimal schedule since we do not have the optimal schedule. Also, consideration of various OD demand patterns, and flows being reactive to network capacity changes, makes it formidable to pin-point the links that can have better schedule through classical network flows optimization models. Hence random grouping is applied to explore various combinations of links for better schedules. Both the decomposition of the links based on the required number of days to repair and decomposition based on links' unit flow cost were tested, but both of them had inferior performance compared to the random grouping approach. Through the iterative randomized fix-and-optimize process, the work zone schedule changes gradually towards the optimal schedule.

4.2 Parameters Affecting the Performance of RFO

The randomized fix-and-optimize heuristic has two layers of computation procedures. The first layer randomly decomposes the links that need maintenance into a specific number of subsets and the second layer optimizes the repair schedules of each link subset with the schedules of links in other subsets fixed (FO subproblem) within a specified time limit. Hence the efficiency of RFO heuristic is mostly determined by two parameters: the number of iterations RFO performs for a specific number of groups which the links to repair are randomly partitioned, and the time limits for the initial attempt on solving the entire problem and for the attempts on each FO subproblem.

More RFO iterations means that the heuristic can solve FO subproblems for more combinations of links to repair for a specific subset size and is more likely to obtain better feasible solutions with objectives that are closer to the optimal solution. However, after a considerable amount of experimentation, we found that increasing the number of iterations does not effectively improve the solution quality. This is because there are too many possible combinations of links to repair for any specific subset size, and the chance is little that the links, which have schedules different from the optimal schedule, are in the same subset through random decomposition. Fewer subsets with more links in each subset can increase the chance of grouping together the links with repair schedules different from the optimal schedule. However, the time needed to find better schedules for each FO subproblem will be longer since now the FO subproblem has large number of integer variables. Thus, performing large number of iterations with fewer subsets with many links in one group will either result in poor solution quality with low time limit for each FO subproblem, or result in

very long solving time with high time limit for each FO subproblems. As default values, we set the number of iterations the same as the specified number of link groups (e.g. perform 2 RFO iterations when the number of groups is 2), and the numerical results in next section will show the RFO gives good feasible solutions within reasonable amount of time.

We also need the time limits for the initial attempt on solving the entire problem and for attempts on each FO subproblem. Problem instances with a few work zones have less integer variables, and is more likely to obtain a feasible solution that is close to the optimal solution (solution with less than 5% relative optimality gap) in a short time during the initial attempt to solve the entire problem. For each FO subproblem, if there is a feasible schedule that is better than the current best feasible schedule, the solver should be able to find it very quickly since the FO subproblem has even less integer variables. As long as a feasible schedule is found that is better than the current best feasible schedule, it can be used as the initial schedule for the next RFO iteration. Increasing the time limit in this case is pointless since a better schedule is already found and increased time will be wasted on improving the lower bound to prove the solution is optimal for the FO subproblem or the entire problem.

As the number of work zones increases, the dramatic increases in the number of combinations of integer variables complicates the branch-and-bound process substantially. This makes it nearly impossible to quickly obtain a feasible solution that is close to the optimal solution in the initial attempt on the entire problem. Improving the quality of initial feasible solution through increasing the time limit is not wise since it is very likely that the relative optimality gap is still larger than 5% after hours of calculation. With an initial feasible solution which is not close to the optimal solution to start the RFO process, it would also be challenging for the solver to find feasible solutions that are much better than the current best feasible solution found in a short time in the FO subproblem. Therefore, increasing the time limit on solving the FO subproblem will be much more effective in finding better solutions since the FO subproblem has much fewer integer variables. And thus, both the time limits on the initial attempt on the entire problem and on the attempts on each FO subproblem should be relatively higher to allow the solver to spend more time on searching for better feasible solutions.

The C++ code of the randomized fix-and-optimize heuristic (RFO) is attached in the appendix, and the detailed procedure of the RFO is shown below:

Randomized Fix-and-optimize Heuristic

---

1. Solve the entire problem with time limit $timeLimitSV$
   If optimal solution obtained, proceed to 4.
   Otherwise store the best feasible schedule and objective, and go to 2.
2. Set number of subsets $N = 2$

3. Randomly divide links to repair into N groups

3.1. Fix (v, s, x, w) for links in $N-1$ groups, set $LonSolTime = 0$, set iteration number $iter\_num = 1$

3.2. Solve the FO subproblem with time limit $timeLimitFO$ for the subset (n) of links the (v, s, x, w) of which are not fixed

If optimal solution is not obtained in $timeLimitFO$ proceed to 3.2.1.

3.2.1. Store the current best feasible schedule and objective, and set $LonSolTime = 1$

Otherwise directly proceed to 3.3.

3.3. If the objective obtained in current FO subproblem is lower than the best objective of the FO subproblems obtained so far ($TotalCostFO$), update the $TotalCostFO$ and the schedule of links in subset n

Otherwise directly proceed to 3.4.

3.4. Check whether there are subsets of links of which the FO subproblems are not solved

If there are, proceed to 3.4.1.

3.4.1. Choose one of the subsets to be subset n and go back to 3.1

Otherwise proceed to 3.4.2.

3.4.2. If $TotalCostFO < TotalCost$ (best objective overall), proceed to 3.4.2.1.

3.4.2.1. Update the value of $TotalCost$ with the value of $TotalCostFO$, increase $iter\_num$ by 1, go back to 3.

Otherwise proceed to 3.4.2.2.

3.4.2.2. If $iter\_num < N$, proceed to 3.4.2.2.1.

3.4.2.2.1. Increase $iter\_num$ by 1, go back to 3.

Otherwise proceed to 3.4.2.2.2.

3.4.2.2.2. If $LonSolTime = 1$, proceed to 3.4.2.2.2.1.

3.4.2.2.2.1. If $numLinpBat > 3$, increase subsets number N by 1, set iteration number 1, go back to 3.

Otherwise proceed to 4.

4. Output the best schedule and flows obtained

4.3 Numerical Results

The randomized fix-and-optimize heuristic is tested on three representative networks: a radial network, a grid network, and the Sioux Falls network. For each network, the links that need maintenance are randomly selected based on the preset percentage of links to repair. Test cases of a network vary by the parameter T, which is the completion date for all the maintenance work. The extra flow cost multiplier ρ is set to 10000 and the percentage of lane capacity increase after repair θ is set to 20% for all the test cases. The computer used to run these tests cases is the same computer mentioned in Section 3.3.

We begin the test on the heuristic designed with a radial network. Radial transportation network structure is commonly found in large cities with long history like London and Paris. The radial network tested is a small network with 6 nodes, 20 links and 20 OD pairs (shown in Figure 5) with 10 randomly selected links that need maintenance. This network has a total number of 30 work zones to be scheduled (since a link has multiple lanes and each lane is an independent work zone). Details of this radial network (i.e., number of lanes of each link, number of days required to repair a lane, lane capacity, unit flow cost,

and whether maintenance is required) and the OD demand are listed in the appendix. The time limits for solving the entire problem initially and for each FO subproblem are both 60 seconds. The performance comparison between solving the test cases by randomized fix-and-optimize heuristic (RFO) and solely by CPLEX is shown in Table 2.

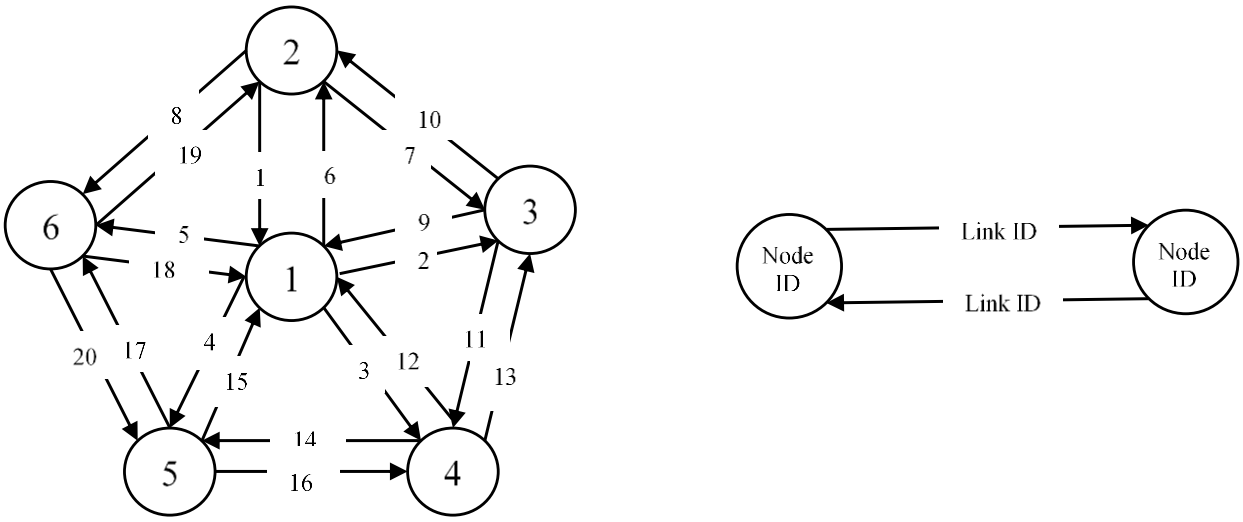


FIGURE 5: Radial Network

TABLE 2: RFO VS CPLEX in Radial Network

Completion Date *(T)*	Solving Time		Objective Value		Optimality Gap
	RFO	CPLEX	RFO	CPLEX	
12	1.89 sec	1.89 sec	489892	489892	0.00%
13	4.37 sec	4.37 sec	404316	404316	0.00%
14	10.70 sec	10.70 sec	318741	318741	0.00%
15	1.52 min	29.75 min	233166	233166	0.00%
16	3.25 min	>14.87 hr.	170591	170591 (UB) 167322 (LB)	0.00%(UB Gap)
17	5.8 min	>40.82 hr.	101516	101516 (UB) 92039 (LB)	0.00%(UB Gap)
18	6.5 min	>2.69 hr.	25645	25645 (UB) 573 (LB)	0.00%(UB Gap)
19	6.52 min	>2.54 hr.	19067	19264 (UB) 6762 (LB)	-1.02%(UB Gap)
20	4.07 min	> 15.73 hr.	9888.26	9790 (UB) 3320 (LB)	1.00%(UB Gap)
26	4.62 sec	4.62 sec	623.34	623.34	0.00%
36	49.79 sec	49.79 sec	856.62	856.62	0.00%
46	1.88 min	1.07 hr.	1090.17	1090.17	0.00%

For the solving time of CPLEX that has ">", it means CPLEX is not able to solve the test case optimally after a long time and the solving process is terminated manually with the best upper bound and lower bound obtained recorded. The upper bound is the objective value of the best feasible solution obtained at the time of terminating the solving process. The optimality gap is calculated as the objective obtained by RFO minus the objective (or upper bound if solving process is terminated manually) obtained by CPLEX and divide the difference by the objective (or upper bound) obtained by CPLEX. These result display formats are the same for the illustration on the experiments on the grid network and Sioux Fall network later.

Since the grouping of links that need maintenance is random for each RFO iteration, the time needed to solve the same test case for each run will be different and the best solution obtained in each run may also be different from each other. To obtain the representative solving time and objective value for each test case that are not solved optimally by CPLEX in 60 seconds, we run RFO to solve each test case for five times, choose the pair of the objective value and solving time that can represent the average performance of RFO, and compare them with the objective and solving time of CPLEX. The objective values and solving

times of five runs of each test case are listed in the appendix. For the test cases of other networks presented later in this report, we use the same approach to obtain the representative objective and solving time, and append their objectives and solving times in the appendix as well. The solving time of RFO and CPLEX for some test cases are the same because CPLEX was able to solve the entire problem in 60 seconds and the randomized fix-and-optimize procedure did not start.

From Table 2 we can see that even for a 20-link radial network with 50% of the links need maintenance, CPLEX is not able to solve some of the test cases in tolerable amount of time. Also, the RFO heuristic is able to obtain optimal or near-optimal solutions within little amount of time compared to CPLEX. Notice that for the test case when $T = 19$, the objective value from RFO is better than the best feasible solution obtained by CPLEX. To obtain the best feasible solution of this test case, RFO takes less than 7 minutes and the solution dominates the best feasible solution from CPLEX after nearly 3 hours of computation.

A larger network tested is a grid network with 16 nodes, 48 links and 24 OD pairs (network is shown in Figure 6). Grid transportation network structure is frequently found in large modern cities like Phoenix and Vancouver, and their central business districts. Like the radial network tested, the grid network tested also has 50% of links randomly selected as the links to be repaired and the total number of work zones to be scheduled is 52. Details of this grid network (i.e., number of lanes of each link, number of days required to repair a lane, lane capacity, unit flow cost, and whether maintenance is required) and the OD demand are listed in the appendix. The time limits set for solving the entire problem initially and for the FO subproblems are both 60 seconds. RFO is used to solve each test case for five times and representative result is chosen to compare with CPLEX as well. The comparison between the average performance of RFO and the performance of CPLEX is displayed in Table 3 on the next page.

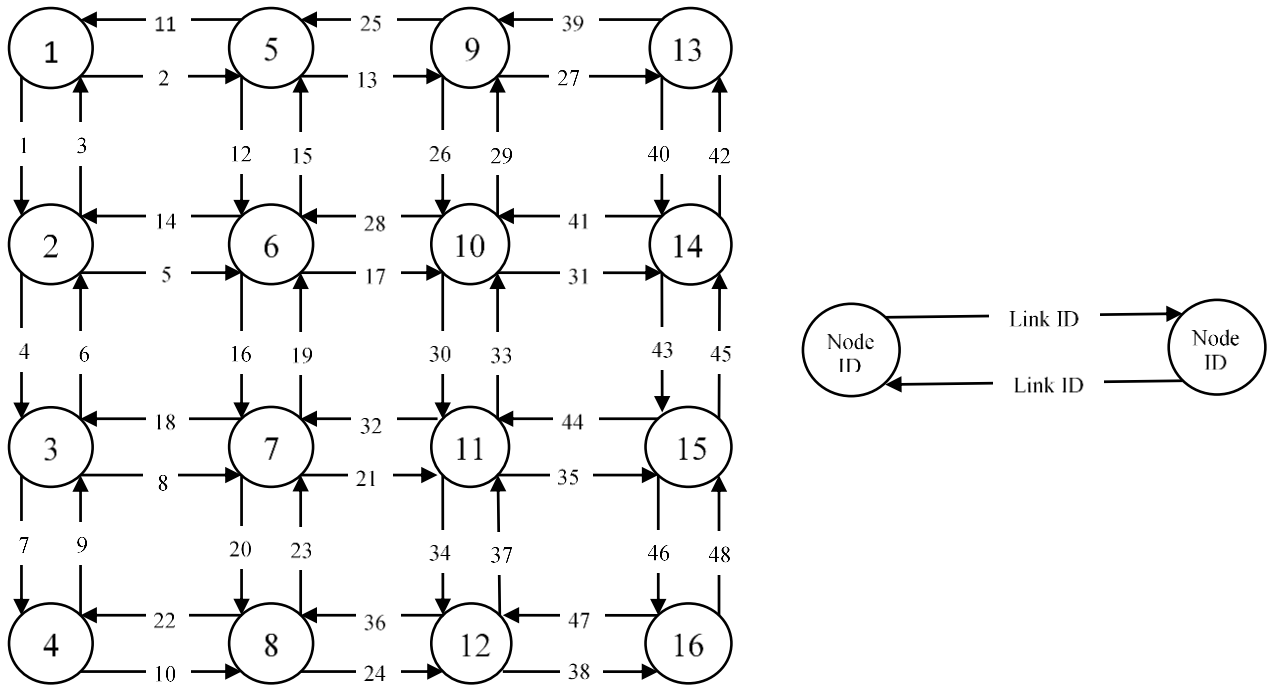


FIGURE 6: Grid Network.

Table 3 below shows that RFO is much more efficient than CPLEX on solving the test cases of the grid network, especially when the test case is difficult to solve. And the solution quality of RFO is also quite good. Usually the percentage of links that need maintenance in a network won't be as much as 50%. The reason we set the percentage of links to repair 50% for the radial network and grid network tested is because we would like to show how difficult the WZS-TS problem can be and how efficient the RFO is compared to solving the test cases solely by CPLEX.

TABLE 3: RFO VS CPLEX in Grid Network

Completion Date *(T)*	Solving Time		Objective Value		Optimality Gap
	RFO	CPLEX	RFO	CPLEX	
12	52.21 sec	52.21 sec	255576	255576	0.00%
13	2.35 min	1.24 min	186740	186740	0.00%
14	5.33 min	25.78 min	143429	142525	0.63%
15	3 min	37.03 min	105997	105502	0.47%
16	4.02 min	27.36 min	67711.7	66209.3	2.27%
17	7.92 min	>14.28 hr.	51773.7	51771(UB) 37692(LB)	0.00%(UB Gap)

18	9.95 min	>1.23 hr.	37350	37344.6(UB) 25990.6(LB)	0.01%(UB Gap)
19	7.32 min	>13.39 hr.	26672.5	26666.25(UB) 19660.61(LB)	0.02%(UB Gap)
20	5.22 min	>3.98 hr.	15988.9	15988.21(UB) 12611.43(LB)	0.00%(UB Gap)
21	4 min	>2.98 hr.	7810.32	7807.98(UB) 5806.61(LB)	0.03%(UB Gap)
22	48.72 sec	48.72 sec	1630.4	1630.4	0.00%
23	57.45 sec	57.45 sec	1701.99	1701.99	0.00%
26	1.97 min	2.75 min	1915.75	1915.66	0.00%
36	2.6 min	2.61 min	2631.65	2630.46	0.05%
46	57.73 sec	57.73 sec	3347.04	3347.04	0.00%
56	31.64 sec	31.64 sec	4066.15	4066.15	0.00%
66	1.38 min	1.75 min	4786.15	4782.58	0.07%

We also test the randomized fix-and-optimize heuristic on the Sioux Falls network which is a real network with 24 nodes, 76 links and 87 OD pairs. There are two sets of problem instances created for the Sioux Falls network, the first set of test cases are based on the scenario that 10% of the links are randomly selected as the links that need maintenance which results in a total number of 16 work zones need to be scheduled. The percentage of links to repair in the second set of test cases is 20% and the total number of work zones to be scheduled is 25. Details of the Sioux-Falls network (i.e., number of lanes of each link, number of days required to repair a lane, lane capacity, unit flow cost, and whether maintenance is required) and the OD demand are listed in the appendix. The time limits on solving the entire problem initially and on solving each FO subproblem are both 40 seconds for first set of test cases, and both are 120 seconds for the second set of test cases. Table 4 and Table 5 give the performance comparison between RFO and CPLEX on the first and second set of test cases respectively. Again, RFO solves each test case five times and the data shown in the table is the one we think represents the average performance of RFO.

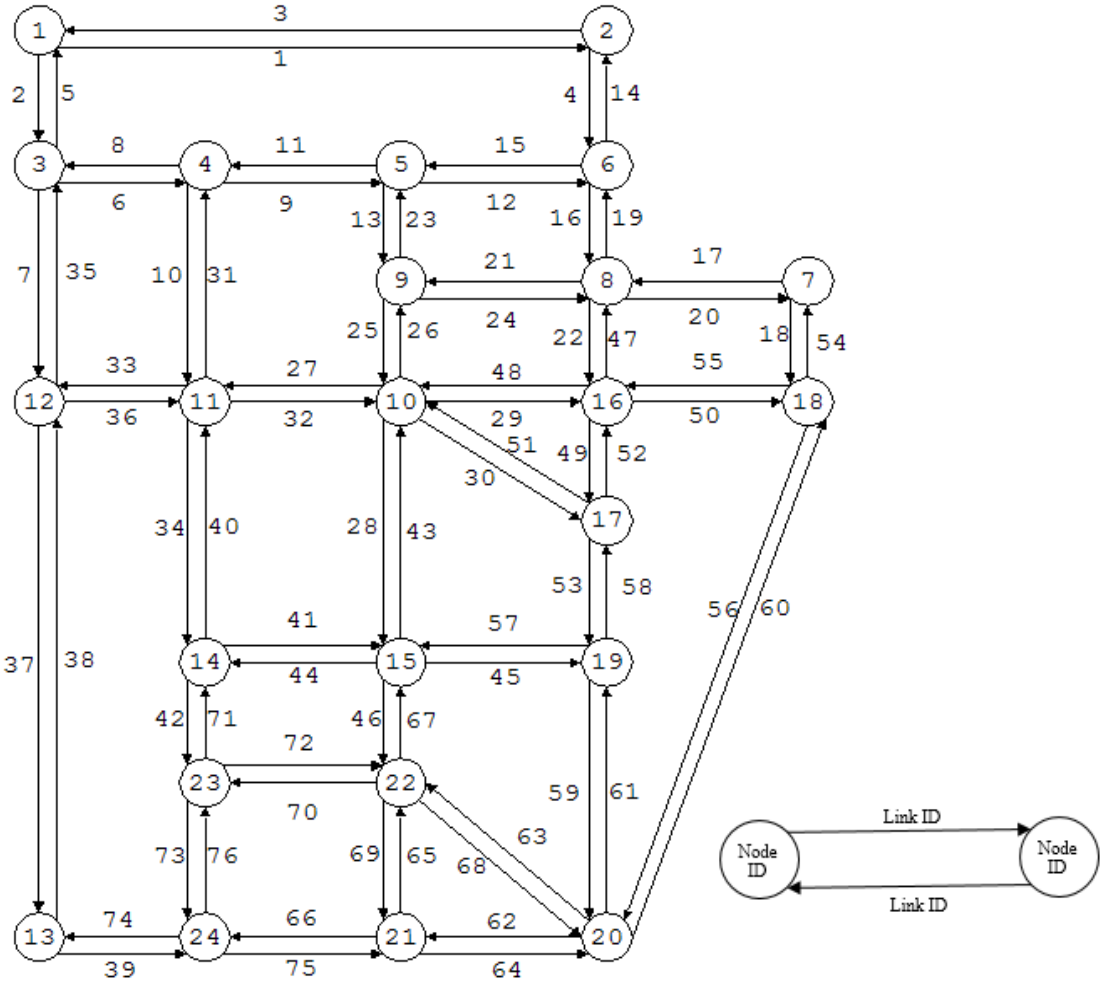


FIGURE 7: Sioux Falls Network.

From Table 4 we see that when the completion date is small the RFO takes more time to give the final solution than CPLEX does. This is because the problem instance of Sioux Falls network with 10% of links to repair is relatively easy to solve especially when the completion date is small, since the number of integer variables are not large. As the completion date gets larger, the problem instance has more integer variables and gets harder to solve. As a result, the solving times of test cases with larger completion dates are much longer for CPLEX. As a comparison, the solving times for RFO on these test cases increase slightly and the objectives obtained are close to the optimal objectives given by CPLEX. Data in Table 5 also shows that RFO generally has better performance compared to CPLEX when 20% of links of the Sioux Falls network need repair.

TABLE 4: RFO VS CPLEX in Sioux Falls Network with 10% of Links to Repair

Completion Date *(T)*	Solving Time		Objective Value		Optimality Gap
	RFO	CPLEX	RFO	CPLEX	
18	33 sec	33 sec	232233.88	232233.88	0.00%
19	2.05 min	1.92 min	237542	237458.8	0.04%
20	3 min	2.22 min	242532	242531.8	0.00%
21	2.47 min	1.1 min	247325	247342.39	-0.01%
22	2.6 min	57.67 sec	252201	252203.14	0.00%
23	3.05 min	1.12 min	260302	260489.83	-0.07%
24	2.97 min	2.11 min	268570	268666.57	-0.04%
25	6.87 min	3.85 min	277223	277160.17	0.02%
26	4.12 min	6.21 min	285841	285930.8	-0.03%
27	6.72 min	3.16 min	294744	294426.42	0.11%
28	8.27 min	16.39 min	303279	302816.79	0.15%
29	8.82 min	12.8 min	311643	311690.72	-0.02%
30	8.7 min	13.27 min	320798	320326.55	0.15%
31	10.78 min	18.85 min	329476	329038.46	0.13%
32	9.35 min	11.79 min	338665	338241.54	0.13%
33	6.48 min	12.82 min	349015	347560.79	0.42%
34	9.1 min	17.7 min	357045	356870.16	0.05%
35	9.47 min	23.64 min	366280	366126.06	0.04%
36	9.38 min	16.4 min	375585	375420.78	0.04%
37	10.95 min	16.12 min	385649	385436.84	0.06%
38	10.53 min	24.18 min	395879	395675.45	0.05%

TABLE 5: RFO VS CPLEX in Sioux Falls Network with 20% of Links to Repair

Completion Date (T)	Solving Time		Objective Value		Optimality Gap
	RFO	Original	RFO	Original	
26	20.63 min	1.02 hr.	429644	429621.64	0.01%
27	27.45 min	1.18 hr.	441998	436397.52	1.28%
28	38.73 min	2.15 hr.	446316	443226.27	0.70%
29	44.5 min	2.37 hr.	451943	451307.57	0.14%
30	53 min	3.22 hr.	462005	459098.29	0.63%
31	53.72 min	2.39 hr.	468724	466737.54	0.43%
32	1.2 hr.	3.68 hr.	474658	474657.98	0.00%
33	47.63 min	3.14 hr.	486761	483550.96	0.66%
34	1.29 hr.	4.37 hr.	495502	492508.96	0.61%
35	1.43 hr.	>1.29 hr.	502656	445782.53 (LB) 502912.96 (UB)	-0.05%
36	1.24 hr.	>1.37 hr.	512588	463690.32 (LB) 511092.08 (UB)	0.29%
37	1.28 hr.	>1.38 hr.	523197	459461.32 (LB) 521498.54 (UB)	0.33%
38	30.32 min	>1.4 hr.	547711	464731.92 (LB) 529503.64 (UB)	3.44%
39	40.72 min	10.4 hr.	544046	537251.06	1.26%
40	38.82 min	>1.42 hr.	563160	469568.44 (LB) 547592.55 (UB)	2.84%
41	43.07 min	>1.4 hr.	563869	52061.60 (LB) 555430.09 (UB)	1.52%
42	47.62 min	>1.43 hr.	585013	482454.42 (LB) 566841.84 (UB)	3.21%

Notice that in Table 4 and 5 the objective obtained by RFO for some test cases is better than the optimal objective obtained by CPLEX. For example, in Table 4 for the test case when $T = 23$, the objective obtained by RFO is 260302, which is less than the optimal objective 260489.83 from CPLEX. This is because the relative MIP gap tolerance is set to 0.5% for the CPLEX and FO subproblems. CPLEX stops solving process as soon as the relative optimality gap (which is calculated as upper bound minus lower bound and then divide the difference by the upper bound) is under 0.5% and uses the best feasible solution obtained as the optimal solution, which is same for FO subproblems. But because of the randomized grouping of links that need repair, it is possible for a FO subproblem start with a branching node that leads to a better upper bound when the 0.5% relative optimality gap is reached, and this node is not selected or reached by CPLEX in the regular branch-and-bound process. So when the 0.5% relative optimality gap is

reached, the upper bound obtained by CPLEX is not as good as the one obtained by RFO. If we reduce the relative MIP gap tolerance to 0.1% or smaller for CPLEX, CPLEX should be able to obtain the same final solution but certainly with much more time spent on the branch-and-bound process.

5. Conclusion

In this project a mixed integer linear programming model is developed to schedule work zones in transportation service networks (WZS-TS). The model coordinates scheduling of work zones with network-wide perspective to achieve minimum total flow cost of all OD demands throughout the entire project period. The WZS-TS problem is very challenging and CPLEX is not able to solve many test cases, even for a small network with 20 links, after hours of computation on a personal computer. The randomized fix-and-optimize heuristic (RFO) is developed to solve the WZS-TS problem efficiently, which can obtain optimal or near-optimal solutions with much less time compared to solving the WZS-TS problem solely with CPLEX. To illustrate the advantage RFO has over CPLEX, the performance comparison between RFO and CPLEX is made on various tests cases created based on three different networks. The authors note that optimal repair and maintenance of other flow carrying service networks, such as power networks, water distribution networks, and communication systems, may benefit from the model and solution method developed in this project.

This project completes the first phase of the research project "Scheduling Work Zones in Multi-modal Networks", whose ultimate goal is to study the optimal work zone scheduling that has minimum negative impacts on both transport service vehicle flows and commute vehicle flows. The next phase of the research project is to consider the work zone scheduling in networks serving OD flows of personal cars (commuter vehicles). The flow cost of commuter vehicles is nonlinear and depends on the relation between the link capacity and the total volume of the traffic that is already using the link. The problem of scheduling work zones in networks of commuter vehicles (CV) is similar to the WZS-SV problem discussed in this project on the scheduling part. But for the OD flow routing part, the CV problem is different in the way that each OD pair will choose its own route or set of routes to minimize its own flow cost and all the OD pairs will reach a user equilibrium that every OD pair won't be able to reduce its flow cost by changing its flow routing unilaterally (Wardrop's first principle). The CV problem is an interesting but challenging mixed integer nonlinear programming problem that cannot be solved by commercial solvers like CPLEX. Study the CV problem is of great value since the majority users of city transportation networks are commuter vehicles. With the results from the first two phases of the research project, the third phase, which studies the work zone scheduling with considerations for both transport service vehicles and commuter vehicles, and the interactions between these two types of vehicle flows, will be carried out.

Reference

Bagloee, S. A., & Asadi, M. (2015). Prioritizing road extension projects with interdependent benefits under time constraint. Transportation Research Part a-Policy and Practice, 75, 196-216. doi:10.1016/j.tra.2015.03,016

Bayraktar, M. E., & Hastak, M. (2009). A decision support system for selecting the optimal contracting strategy in highway work zone projects. *Automation in Construction, 18*(6), 834-843. doi:10.1016/j.autcon.2009.03.007

Chien, S. I. J., & Tang, Y. (2014). Scheduling highway work zones with genetic algorithm considering the impact of traffic diversion. *Journal of Advanced Transportation, 48*(4), 287-303. doi:10.1002/atr.213

Chu, J. C., & Chen, Y. J. (2012). Optimal threshold-based network-level transportation infrastructure life-cycle management with heterogeneous maintenance actions. *Transportation Research Part B-Methodological, 46*(9), 1123-1143. doi:10.1016/j.trb.2012.05.002

Even, S., Itai, A., Shamir, A. (Oct. 1975). *On the complexity of time table and multi-commodity flow problems.* Paper presented at the 16th Annual Symposium on Foundations of Computer Science, Washington, DC, USA.

Fontaine, P., & Minner, S. (2014). Benders Decomposition for Discrete-Continuous Linear Bilevel Problems with application to traffic network design. Transportation Research Part B-Methodological, 70, 163-172. doi:10.1016/j.trb.2014.09.007

FWHA. (September, 2013). Facts and Statistics - Work Zone Delay. *Facts and Statistics.* Retrieved fromhttp://www.ops.fhwa.dot.gov/wz/resources/facts_stats/delay.htm#fn1

Gao, L., Xie, C., Zhang, Z. M., & Waller, S. T. (2011). Integrated Maintenance and Expansion Planning for Transportation Network Infrastructure. Transportation Research Record(2225), 56-64. doi:10.3141/2225-07

Helber, S., & Sahling, F. (2010). A fix-and-optimize approach for the multi-level capacitated lot sizing problem.*International Journal of Production Economics, 123*(2), 247-256. doi:10.1016/j.ijpe.2009.08.022

Hosseininasab, S.-M., & Shetab-Boushehri, S.-N. (2015). Integration of selecting and scheduling urban road construction projects as a time-dependent discrete network design problem. European Journal of Operational Research, 246(3), 762-771. doi:10.1016/j.ejor.2015.05.039

Jiang, X. M., & Adeli, H. (2003). Freeway work zone traffic delay and cost optimization model. *Journal of Transportation Engineering-Asce, 129*(3), 230-241. doi:10.1061/(asce)0733-947x(2003)129:3(230)

Lee, H. Y. (2009). Optimizing schedule for improving the traffic impact of work zone on roads. *Automation in Construction, 18*(8), 1034-1044. doi:10.1016/j.autcon.2009.05.004

Ma, W. T., Cheu, R. L., & Lee, D. H. (2004). Scheduling of lane closures using genetic algorithms with traffic assignments and distributed simulations. Journal of Transportation Engineering-Asce, 130(3), 322-329. doi:10.1061/(asce)0733-947x(2004)130:3(322)

Meng, Q., & Weng, J. X. (2013). Optimal subwork zone operational strategy for short-term work zone projects in four-lane two-way freeways. *Journal of Advanced Transportation, 47*(2), 151-169. doi:10.1002/atr.153

Orabi, W., & El-Rayes, K. (2012). Optimizing the Rehabilitation Efforts of Aging Transportation Networks. *Journal of Construction Engineering and Management-Asce, 138*(4), 529-539. doi:10.1061/(asce)co.1943-7862.0000445

Schroeder, B. J., & Rouphail, N. M. (2010). Estimating Operational Impacts of Freeway Work Zones on Extended Facilities.*Transportation Research Record*(2169), 70-80. doi:10.3141/2169-08

Tang, Y. M., & Chien, S. I. J. (2008). Scheduling Work Zones for Highway Maintenance Projects Considering a Discrete Time-Cost Relation. *Transportation Research Record*(2055), 21-30. doi:10.3141/2055-03

Zheng, H., Nava, E., & Chiu, Y.-C. (2014). Measuring Networkwide Traffic Delay in Schedule Optimization for Work-Zone Planning in Urban Networks. *Ieee Transactions on Intelligent Transportation Systems, 15*(6), 2595-2604. doi:10.1109/tits.2014.2318299

Appendix

Radial Network Link Information

Link ID	Head Node	Tail Node	Number of Lanes	Capacity per Lane	Regular Unit Flow Cost	Number of Days to Repair a Lane	Need to Repair?
1	1	2	3	50	0.01	6	1
2	1	3	3	50	0.01	6	1
3	1	4	3	50	0.01	6	1
4	1	5	3	50	0.01	6	0
5	1	6	3	50	0.01	6	1
6	2	1	3	50	0.01	6	0
7	2	3	3	50	0.015	9	1
8	2	6	3	50	0.015	9	1
9	3	1	3	50	0.01	6	0
10	3	2	3	50	0.015	9	0
11	3	4	3	50	0.015	9	0
12	4	1	3	50	0.01	6	0
13	4	3	3	50	0.015	9	1
14	4	5	3	50	0.015	9	0
15	5	1	3	50	0.01	6	0
16	5	4	3	50	0.015	9	1
17	5	6	3	50	0.015	9	1
18	6	1	3	50	0.01	6	0
19	6	2	3	50	0.015	9	0
20	6	5	3	50	0.015	9	1

Radial Network OD Demand Information

OD ID	Origin Node	Destination Node	Demand Units
1	1	2	64
2	1	4	80
3	2	3	50
4	3	1	72
5	3	4	74
6	4	5	63
7	4	1	54
8	5	6	56
9	6	5	119
10	6	2	39
11	2	4	70
12	3	5	64
13	4	6	92

OD ID	Origin Node	Destination Node	Demand Units
14	5	2	73
15	6	3	53
16	5	3	67
17	2	5	55
18	3	6	92
19	4	2	56
20	5	4	60

Objective and Time Consumption of Five Runs by RFO for Each Test Case of the Radial Network

Completion Date *(T)*	Run 1		Run 2		Run 3		Run 4		Run 5	
	Objective Value	Solving Time	Objective Value	Solving Time	Objective Value	Solving Time	Objective Value	Solving Time	Objective Value	Solving Time
15	233166	1.6 min	233166	1.45 min	233166	1.58 min	233166	1.52 min	233166	1.5 min
16	170591	3.25 min	170591	2.85 min	170591	3.02 min	170591	4.4 min	170591	4.95 min
17	101516	8.35 min	101516	4.93 min	101516	6.12 min	101516	5.45 min	101516	5.8 min
18	25644.7	7.1 min	25644.7	6.5 min	26547.7	5.92 min	25677.7	6.1 min	25647.7	4.97 min
19	19668.1	4.92 min	19067.3	12.87 min	19067.3	6.52 min	19067.4	6.52 min	19067.4	4.32 min
20	10889.6	6.15 min	10389.2	6.56 min	9888.26	7.4 min	9888.26	9.42 min	9888.07	7.42 min

Grid Network Link Information

Link ID	Head Node	Tail Node	Number of Lanes	Capacity per Lane	Regular Unit Flow Cost	Number of Days to Repair a Lane	Need to Repair?
1	1	2	2	50	0.08	16	1
2	1	5	2	50	0.07	14	0
3	2	1	2	50	0.055	11	1
4	2	3	2	50	0.04	8	1
5	2	6	2	50	0.045	9	0
6	3	2	2	50	0.055	11	1
7	3	4	2	50	0.07	14	1
8	3	7	3	50	0.04	8	1
9	4	3	2	50	0.09	18	0
10	4	8	2	50	0.035	7	0
11	5	1	2	50	0.06	12	0
12	5	6	2	50	0.015	3	0
13	5	9	2	50	0.06	12	0
14	6	2	2	50	0.04	8	1
15	6	5	2	50	0.04	8	1
16	6	7	2	50	0.07	14	0
Link ID	Head Node	Tail Node	Number of Lanes	Capacity per Lane	Regular Unit Flow Cost	Number of Days to Repair a Lane	Need to Repair?

17	6	10	2	50	0.02	4	1
18	7	3	3	50	0.09	18	0
19	7	6	2	50	0.02	4	1
20	7	8	2	50	0.11	22	0
21	7	11	3	50	0.075	15	0
22	8	4	2	50	0.05	10	1
23	8	7	2	50	0.09	18	1
24	8	12	2	50	0.065	13	0
25	9	5	2	50	0.055	11	0
26	9	10	3	50	0.035	7	0
27	9	13	2	50	0.03	6	1
28	10	6	2	50	0.04	8	0
29	10	9	3	50	0.045	9	0
30	10	11	3	50	0.04	8	1
31	10	14	2	50	0.085	17	1
32	11	7	3	50	0.045	9	0
33	11	10	3	50	0.095	19	0
34	11	12	3	50	0.06	12	0
35	11	15	3	50	0.06	12	1
36	12	8	2	50	0.065	13	0
37	12	11	3	50	0.08	16	0
38	12	16	2	50	0.04	8	1
39	13	9	2	50	0.05	10	0
40	13	14	2	50	0.04	8	1
41	14	10	2	50	0.09	18	0
42	14	13	2	50	0.03	6	1
43	14	15	2	50	0.02	4	1
44	15	11	3	50	0.1	20	1
45	15	14	2	50	0.015	3	0
46	15	16	2	50	0.055	11	1
47	16	12	2	50	0.11	22	1
48	16	15	2	50	0.015	3	1

Grid Network OD Demand Information

OD ID	Origin Node	Destination Node	Demand Units
1	1	8	39
2	1	16	11
3	1	11	26
4	1	3	20
OD ID	Origin Node	Destination Node	Demand Units
5	4	9	23

6	4	13	39
7	4	10	32
8	4	2	24
9	13	12	18
10	13	4	37
11	13	7	25
12	13	15	25
13	16	9	47
14	16	1	29
15	16	6	33
16	16	14	28
17	6	12	39
18	6	15	51
19	7	9	14
20	7	14	38
21	10	3	19
22	10	8	48
23	11	2	29
24	11	5	19

Objective and Time Consumption of Five Runs by RFO for Each Test Case of the Grid Network

Completion Date *(T)*	Run 1		Run 2		Run 3		Run 4		Run 5	
	Objective Value	Solving Time	Objective Value	Solving Time	Objective Value	Solving Time	Objective Value	Solving Time	Objective Value	Solving Time
14	143.23	4.9 min	144071	4.2 min	143630	2.78 min	143033	5.23 min	143429	5.33 min
15	105997	3 min	105997	6.5 min	105991	5.71 min	105989	3.75 min	106000	3.5 min
16	67527.2	3.05 min	68704.6	5.43 min	67711.7	2.23 min	66211.3	3.25 min	67711.7	4.02 min
17	51773.6	5.87 min	51773.7	4.1 min	51773.7	7.92 min	51772	7.28 min	51772	8.18 min
18	37350	9.95 min	37350.5	4.53 min	37348.3	3.95 min	37350.3	13 min	38602	7.98 min
19	26921.2	7.53 min	26671.2	6.56 min	26921.4	6.58 min	26672.5	7.32 min	26671.4	6.25 min
20	15989.2	6.68 min	15988.9	5.22 min	15989.5	3.62 min	15989.2	3.75 min	15989.3	6.5 min
21	7809.33	4.48 min	7810.41	7.67 min	7809.9	3.65 min	7810.32	4 min	7809.11	6.7 min
26	1915.95	2.13 min	1915.51	2.17 min	1914.49	2.15 min	1915.75	1.97 min	1915.97	2.02 min
36	2631.65	2.6 min	2630.52	2.6 min	2631.65	2.57 min	2628.9	2.93 min	2631.65	2.65 min

Sioux-Falls Network Link Information

Link ID	Head Node	Tail Node	Number of Lanes	Capacity per Lane	Regular Unit Flow Cost	Number of Days to Repair a Lane	Need to Repair?	
							10% Case	20% Case
1	1	2	4	259	0.0006	18	0	0

2	1	3	4	234	0.0004	15	0	0
3	2	1	4	259	0.0006	18	1	0
4	2	6	1	198	0.0005	14	0	0
5	3	1	4	234	0.0004	15	0	0
6	3	4	3	228	0.0004	13	0	1
7	3	12	4	234	0.0004	15	0	0
8	4	3	3	228	0.0004	13	0	0
9	4	5	4	178	0.0002	10	0	0
10	4	11	1	196	0.0006	12	0	1
11	5	4	4	178	0.0002	10	0	0
12	5	6	1	198	0.0004	12	0	1
13	5	9	2	200	0.0005	10	0	0
14	6	2	1	198	0.0005	14	1	0
15	6	5	1	198	0.0004	12	0	0
16	6	8	1	196	0.0002	9	0	1
17	7	8	2	157	0.0003	10	0	0
18	7	18	4	234	0.0002	11	1	0
19	8	6	1	196	0.0002	9	0	0
20	8	7	2	157	0.0003	10	0	0
21	8	9	1	202	0.001	14	0	1
22	8	16	1	202	0.0005	11	0	0
23	9	5	2	200	0.0005	10	0	0
24	9	8	1	202	0.001	14	0	0
25	9	10	3	186	0.0003	6	0	0
26	10	9	3	186	0.0003	6	0	0
27	10	11	2	200	0.0005	9	0	0
28	10	15	3	180	0.0006	10	0	0
29	10	16	1	194	0.0004	10	0	1
30	10	17	1	200	0.0008	13	0	0
31	11	4	1	196	0.0006	12	0	1
32	11	10	2	200	0.0005	9	0	0
33	11	12	1	196	0.0006	14	0	0
34	11	14	1	195	0.0004	9	0	0
35	12	3	4	234	0.0004	15	0	0
36	12	11	1	196	0.0006	14	0	0
37	12	13	4	259	0.0003	14	0	0
38	13	12	4	259	0.0003	14	0	1
39	13	24	1	204	0.0004	12	0	1
40	14	11	1	195	0.0004	9	0	0
41	14	15	1	205	0.0005	10	0	0
42	14	23	1	197	0.0004	9	0	0

Link ID	Head Node	Tail Node	Number of Lanes	Capacity per Lane	Regular Unit Flow Cost	Number of Days to Repair a Lane	Need to Repair?	
							10% Case	20% Case

43	15	10	3	180	0.0006	10	0	0
44	15	14	1	205	0.0005	10	0	0
45	15	19	3	194	0.0003	10	0	0
46	15	22	2	192	0.0003	11	0	0
47	16	8	1	202	0.0005	11	0	1
48	16	10	1	194	0.0004	10	0	1
49	16	17	1	209	0.0002	6	1	1
50	16	18	4	197	0.0003	9	0	0
51	17	10	1	200	0.0008	13	1	0
52	17	16	1	209	0.0002	6	0	0
53	17	19	1	193	0.0002	8	0	0
54	18	7	4	234	0.0002	11	0	0
55	18	16	4	197	0.0003	9	0	0
56	18	20	4	234	0.0004	16	0	0
57	19	15	3	194	0.0003	10	1	0
58	19	17	1	193	0.0002	8	0	0
59	19	20	1	200	0.0004	10	0	0
60	20	18	4	234	0.0004	16	0	1
61	20	19	1	200	0.0004	10	0	0
62	20	21	1	202	0.0006	11	0	0
63	20	22	1	203	0.0005	12	0	0
64	21	20	1	202	0.0006	11	0	0
65	21	22	1	209	0.0002	8	0	0
66	21	24	1	195	0.0003	10	0	0
67	22	15	2	192	0.0003	11	0	1
68	22	20	1	203	0.0005	12	0	1
69	22	21	1	209	0.0002	8	0	0
70	22	23	1	200	0.0004	10	0	0
71	23	14	1	197	0.0004	9	0	0
72	23	22	1	200	0.0004	10	1	1
73	23	24	1	203	0.0002	7	1	0
74	24	13	1	204	0.0004	12	0	0
75	24	21	1	195	0.0003	10	0	0
76	24	23	1	203	0.0002	7	0	0

Sioux-Falls Network OD Demand Information

OD ID	Origin Node	Destination Node	Demand Units
1	1	4	92
2	1	8	100

3	1	10	108
4	1	14	88
5	1	16	112
6	1	19	108
7	1	22	120
8	1	24	88
9	2	4	104
10	2	8	96
11	2	10	100
12	2	14	104
13	2	16	84
14	2	19	96
15	2	22	92
16	7	4	80
17	7	8	104
18	7	10	96
19	7	14	100
20	7	16	92
21	7	19	112
22	7	22	92
23	7	24	112
24	12	4	96
25	12	8	108
26	12	10	104
27	12	14	96
28	12	16	120
29	12	19	92
30	12	22	100
31	12	24	120
32	13	4	92
33	13	8	96
34	13	10	108
35	13	14	104
36	13	16	92
37	13	19	120
38	13	22	108
39	13	24	112

OD ID	Origin Node	Destination Node	Demand Units
40	20	4	80
41	20	8	100
42	20	10	96
43	20	14	104

44	20	16	108
45	20	19	104
46	20	22	80
47	20	24	104
48	4	1	5
49	4	2	5
50	4	7	4
51	4	12	5
52	4	20	5
53	4	13	4
54	8	1	5
55	8	2	5
56	8	7	5
57	8	12	5
58	8	20	5
59	8	13	5
60	10	1	5
61	10	2	5
62	10	7	5
63	10	12	5
64	10	20	5
65	10	13	5
66	14	1	4
67	14	2	5
68	14	7	5
69	14	12	5
70	14	20	5
71	14	13	5
72	16	1	6
73	16	2	4
74	16	7	5
75	16	12	6
76	16	20	5
77	16	13	5
78	19	1	5
79	19	2	5
80	19	7	6
OD ID	Origin Node	Destination Node	Demand Units
81	19	12	5
82	19	20	6
83	19	13	5
84	22	1	6

85	22	2	5
86	22	7	5
87	22	12	5
88	22	20	5
89	22	13	4
90	24	1	4
91	24	7	6
92	24	12	6
93	24	20	6
94	24	13	5

Objective and Time Consumption of Five Runs by RFO for Each Test Case of the Sioux-Falls Network with 10% of Links to Repair

Completion Date *(T)*	Run 1		Run 2		Run 3		Run 4		Run 5	
	Objective Value	Solving Time	Objective Value	Solving Time	Objective Value	Solving Time	Objective Value	Solving Time	Objective Value	Solving Time
19	237459	2.68 min	237494	4.63 min	237515	5.12 min	237487	2.17 min	237542	2.05 min
20	242532	4.67 min	242542	2.2 min	242518	4.47 min	242532	3 min	242542	2.58 min
21	247317	2.17 min	247325	2.63 min	247325	2.47 min	247325	2.58 min	247325	2.35 min
22	252140	2.57 min	252201	2.38 min	252201	2.6 min	252203	3.62 min	252140	2.77 min
23	260386	3.05 min	260302	2.87 min	260302	5.6 min	260318	2.75 min	260302	3.05 min
24	268500	2.87 min	268570	2.97 min	268498	6.53 min	268603	3.25 min	268631	3.03 min
25	277223	6.87 min	277258	3.35 min	277170	7.15 min	277339	3.07 min	277216	7.28 min
26	285841	7.65 min	285791	3.13 min	285841	4.12 min	285841	7.63 min	285843	3.35 min
27	294744	7.07 min	294744	4.75 min	294744	6.72 min	294573	5.3 min	294591	7 min
28	302933	7.92 min	303279	8.27 min	303278	8.92 min	303529	3.93 min	303396	7.5 min
29	311643	7.37 min	311643	8.82 min	311691	4.77 min	311447	8.37 min	311723	8.23 min
30	320849	5.45 min	320798	8.7 min	320522	8.27 min	320756	8.28 min	320798	8.75 min
31	329556	9.12 min	329368	10.77 min	329476	10.78 min	329430	8.18 min	329436	8.93 min
32	338608	10.93 min	338334	10.35 min	338860	9.08 min	338829	10.62 min	338665	9.35 min
33	349090	8.18 min	349265	2.67 min	347910	6.83 min	347897	3.83 min	349015	6.48 min
34	357086	8.43 min	357090	8.63 min	357045	9.5 min	357064	9.1 min	256866	10.38 min
35	366209	11.55 min	366280	9.47 min	366242	9.22 min	366256	11.11 min	366335	9.8 min
36	375665	9.62 min	375585	9.38 min	375633	9.57 min	375407	9.52 min	375516	10.18 min
37	385649	10.08 min	385649	11.1 min	385649	12.07 min	385649	10.95 min	358649	9.5 min
38	395879	10.53 min	395879	11.17 min	395879	10.75 min	395743	10.05 min	395861	12.45 min

Objective and Time Consumption of Five Runs by RFO for Each Test Case of the Sioux-Falls Network with 20% of Links to Repair

Completion Date (*T*)	Run 1		Run 2		Run 3		Run 4		Run 5	
	Objective Value	Solving Time	Objective Value	Solving Time	Objective Value	Solving Time	Objective Value	Solving Time	Objective Value	Solving Time
26	429644	20.63 min	419612	19.38 min	429673	23.15 min	430182	18.85 min	430182	18.33 min
27	441983	24.33 min	439772	28.43 min	440444	34.22 min	441998	27.45 min	441053	55.08 min
28	446741	1.03 hr.	446946	25.7 min	446316	38.73 min	444821	1.13 hr.	447709	26.4 min
29	451943	44.5 min	451798	1.06 hr.	452415	41.88 min	451972	1.17 hr.	451404	23.28 min
30	461263	37.18 min	462005	53 min	465685	42.33 min	463749	22.77 min	460272	49.8 min
31	468504	1.17 hr.	468968	52.93 min	468724	53.72 min	468724	1.33 hr.	467886	1.18 hr.
32	474658	1.2 hr.	475698	33.17 min	475486	1.14 hr.	474658	1.34 hr.	474848	48.28 min
33	486761	47.63 min	485102	1.11 hr.	485800	1.23 hr.	487476	24.58 min	486772	44.37 min
34	495069	1.3 hr.	496514	1.25 hr.	496356	1.24 hr.	495278	1.21 hr.	495502	1.29 hr.
35	502656	1.43 hr.	502775	1.48 hr.	502704	1.31 hr.	502616	1.47 hr.	502656	1.3 hr.
36	513889	1.5 hr.	513445	1.04 hr.	512527	1.25 hr.	512588	1.24 hr.	514482	1.14 hr.
37	523158	1.41 hr.	521056	1.27 hr.	523197	1.28 hr.	521984	1.14 hr.	523766	1.35 hr.
38	547711	30.32 min	550335	38.5 min	535769	17.73 min	554759	52.37 min	558797	44.03 min
39	543698	26.23 min	544046	40.72 min	553827	46.6 min	560203	33.07 min	542010	38.35 min
40	568072	56.52 min	563160	38.82 min	575092	56.82 min	556089	39.6 min	551897	19.53 min
41	563869	43.07 min	562826	31.43 min	561137	30.55 min	581717	1.16 hr.	573486	40.75 min
42	571838	49.55 min	594626	1.02 hr.	567858	30.08 min	585013	47.62 min	588380	1.07 hr.

C++ Code of the Randomized Fix-and-Optimize Heuristic

```
//RFO of Workzone Scheduling for Transportation Service Networks -- Two Layer Structure
with Network Connectivity Check Ensuring the Feasibility of a Schedule
//Dening Peng
//%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%
#include <ilcplex/ilocplex.h>
#include <vector>
#include <time.h>
#include <math.h>
#include <iostream>
#include <fstream>
#include <sstream>
#include <cmath>
#include <random>

ILOSTLBEGIN  // macro that allows the use of standard libraries

//%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%
```

```
//=============================================================================================
==========
//Declare MIP callbacks
//=============================================================================================
==========
ILOMIPINFOCALLBACK1(AbortCriteria, IloNum, TotalCostFO){
	if(getBestObjValue() > TotalCostFO){
		abort();		//abort current fix and optimize subproblem if the
lower bound is larger than the current best FO objective
		cout << endl;
		cout << "The optimization of current FO subproblm is aborted." << endl;
		//cin.get();
	}
}

//%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%
//=============================================================================================
==========
//Declare functions
//=============================================================================================
==========
void InputData();
void Initialization();

//%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%
//=============================================================================================
==========
//Declare struct for data
//=============================================================================================
==========
struct generalInfo{
	// from text file "general"
	int numNodes;							//number of
nodes
	int numLinks;							//number of
links
	int numODs;
	//number of Origin-Destination (OD) pairs
	int maxnumLanes;						//max number
of lanes among all links
	float rho;
	//ratio of unit extra flow cost to regular flow cost
	int T;
	//required maintenance completion date (RMCD)
	float theta;							//percentage
of capacity increase once the lane is repaired, 0<theta<0.5
	//int batchMethod;						//the method
of batching the links for sequential fix and optimize procedure, 0 is random batching, 1
is batching based on pindex
	float numLinpBat;
	//number of links per batch
};

struct linkInfo{
	// from MS Excel file worksheet "link"
```

```
	int ID;
	//link ID
	int FN;
	//from node
	int TN;
	//to node
	int n;										//# of
lanes of the link
	int u;
	//capacity upper bound of a lane
	float c;									//cost
inccured by one unit flow on link when the link is regularly open to serve traffic
	int p;										//the
number of days needed to repair a lane of the link
	bool R;
	//indicator of whether the lane needs repair at the begining of the repair project
	int obindex;								//obindex is
the sequence of different batches of links to have their maintenance start dates
optimized
};

struct ODInfo{
	// from MS Excel file worksheet "od"
	int ID;
	//Origin-Destination (OD) pair ID
	int ON;
	//origin node
	int DN;
	//destination node
	int D;
	//traffic flow demand of OD pair each day
};

//%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%
//=========================================================================================
==========
// parameters
//=========================================================================================
==========
//special structs
	generalInfo general;			// general parameters
	vectorlinkInfo> link;				// link information
	vector<ODInfo> OD;					// OD information

//parameters
	float timeLimitSA,					// time stopping criteria for SA
problem
		timeLimitFO,						// time stopping criteria
for FO subproblems
		mipGap,							// optimality gap stopping
criteria
		cplexLB;						// lower bound on solution
	float cplexUB;						// upper bound on solution

	float TotalCostFO;					//store the obejctive value of
current iteration of the Fix and Optimize problem
```

```
	float TotalCost;				//store the objective value of the best feasible solution in FO relaxation iteration
	int sumD;					//the summation of all the OD demand. This number is used to construct the constraint that ensures entirely closed links can not serve any flows
	float numBatches;				//total number of batches of links to go through fix and optimize procedure
	bool aborted;		//flag variable indicating that at least one FO subproblem of current batching has aborted the optimization process because of LB is greater than TotalCost

	vector<vector<vector<float> > >y_o;		//variables used for final solution output
	vector<vector<float> >z_o;

//parameters used for lower layer multi-commodity flow problem
	vector<vector<vector<bool> > > s_o, x_o, v_o;		//store the schedule with the best objective so far of upper layer scheduling problem
	vector<vector<bool> >w_o;

//%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
//==================================================================================================
// the class of the entire workzone scheduling problem for networks of service agents (SA)
//==================================================================================================
class SA{
public:
	IloEnv env;
	IloModel SA_Opt;			//model for the SA problem
    IloCplex SA_Solver;			//solver for the SA problem
    IloObjective obj;
    IloExpr expr;
    IloInt k, a, t, m, i, j;
	IloNumVar dummy, dummy1;
	IloRangeArray con_Nonpremp, con_StartDate, con_EntireClose, con_RepairClose, con_ODDemand, con_FlowConserve, con_V, con_W, con_Z;
//desicion variables
	IloArray<IloArray<IloArray<IloNumVar> > > s, x, v, y;
	IloArray<IloArray<IloNumVar> > w, z;

public:
/********************************Function that builds the model FO and solves the problem instance*******************************/
	void BuildSolveSA(){
		IloModel SA_Opt(env);
		IloCplex SA_Solver(SA_Opt);
		IloExpr expr(env);		//expression used while building model
		IloExpr expr1(env);		//expression used while building mode
		IloNumVar dummy(env, 0, 0, ILOFLOAT);
		IloNumVar dummy1(env, 0, 0, ILOFLOAT);

//Define Variables
		IloArray<IloArray<IloArray<IloNumVar> > > s(env, general.numLinks);
	//variable for the repair start date of each lane of each link
```

```
		IloArray<IloArray<IloArray<IloNumVar> > > x(env, general.numLinks);
	//variable indicating whether the lane of a link is open or closed
		IloArray<IloArray<IloArray<IloNumVar> > > v(env, general.numLinks);
	//if the mth lane of link (i,j) is repaired before day t, v_{ijmt}=1, otherwise 0
		IloArray<IloArray<IloArray<IloNumVar> > > y(env, general.numLinks);
	//flow variable used to ensure the schedule generated is feasible

		for (i = 0; i < general.numLinks; i++){
				//link index
			s[i] = IloArray<IloArray<IloNumVar> >(env, general.maxnumLanes);
			x[i] = IloArray<IloArray<IloNumVar> >(env, general.maxnumLanes);
			v[i] = IloArray<IloArray<IloNumVar> >(env, general.maxnumLanes);
			for (m = 0; m < link[i].n; m++){
		//lane index
				s[i][m] = IloArray<IloNumVar>(env, general.T);
				x[i][m] = IloArray<IloNumVar>(env, general.T);
				v[i][m] = IloArray<IloNumVar>(env, general.T);
				for(t = 0; t < general.T; t++){
				//date index
					s[i][m][t] = IloNumVar(env, 0, 1, ILOBOOL);
					x[i][m][t] = IloNumVar(env, 0, 1, ILOBOOL);
					v[i][m][t] = IloNumVar(env, 0, 1, ILOBOOL);
				}
			}
		}

		for (i = 0; i < general.numLinks; i++){
	//link index
			y[i] = IloArray<IloArray<IloNumVar> >(env, general.numODs);   //OD
index
			for(k = 0; k < general.numODs; k++){
				y[i][k] = IloArray<IloNumVar>(env, general.T);
	//date index
				for(t = 0; t < general.T; t++){
					y[i][k][t] = IloNumVar(env, 0, IloInfinity, ILOFLOAT);
				}
			}
		}

		w = IloArray<IloArray<IloNumVar> >(env, general.numLinks);
	//whether all the lanes of link (i,j) is closed on day t, if it is, w_{ijt}=1;
otherwise 0
		z = IloArray<IloArray<IloNumVar> >(env, general.numLinks);
		for (i = 0; i < general.numLinks; i++) {
			w[i] = IloArray<IloNumVar>(env, general.T);
			z[i] = IloArray<IloNumVar>(env, general.T);
			for (t = 0; t < general.T; t++){
				w[i][t] = IloNumVar(env, 0, 1, ILOBOOL);
				z[i][t] = IloNumVar(env, 0, IloInfinity, ILOFLOAT);
			}
		}
//Define Constraints
//Constraints ensuring once a lane is closed for repair, it won't open to serve the flows
until it is fully repaired
		IloRangeArray con_Nonpremp(env);
		for(i = 0; i < general.numLinks; i++) {
			if(link[i].R > 0.5){
				for(m = 0; m < link[i].n; m++){
```

```
                                    for(t = 0; t < general.T; t++){
                                          expr = dummy;
                                          if((t - link[i].p + 1) > 0){
                                                for(a = t - link[i].p + 1; a < t + 1;
a++){
                                                      expr += s[i][m][a];
                                                }
                                          }
                                          else{
                                                for(a = 0; a < t + 1; a++){
                                                      expr += s[i][m][a];
                                                }
                                          }
                                          expr = x[i][m][t] - expr;
                                          con_Nonpremp.add(IloRange(env, 0, expr, 0, 0));
                                          expr.end();
                                    }
                              }
                        }
                  }
                  SA_Opt.add(con_Nonpremp);
//Constraints ensuring each lane of the links to be repaired has a repair start date, and
those do not need repair does not have a repair start date
                  IloRangeArray con_StartDate(env);
                  for(i = 0; i < general.numLinks; i++) {
                        for(m = 0; m < link[i].n; m++){
                              expr = dummy;
                              for(t = 0; t < general.T; t++){
                                    expr += s[i][m][t];
                              }
                              if(link[i].R > 0.5){
                                    expr = expr - 1;
                              }
                              con_StartDate.add(IloRange(env, 0, expr, 0, 0));
                              expr.end();
                        }
                  }
                  SA_Opt.add(con_StartDate);
//All links that need maintenance have to be repaired before the end of the time horizon,
all links that don't need repair can't close any lanes at any time
                  IloRangeArray con_RepairClose(env);
                  for(i = 0; i < general.numLinks; i++) {
                        for(m = 0; m < link[i].n; m++){
                              expr = dummy;
                              for(t = 0; t < general.T; t++){
                                    expr += x[i][m][t];
                              }
                              if(link[i].R == 1){
                                    expr = expr - link[i].p;
                              }
                              con_RepairClose.add(IloRange(env, 0, expr, 0, 0));
                              expr.end();
                        }
                  }
                  SA_Opt.add(con_RepairClose);
//Constraint ensuring entirely closed link won't serve flows
                  IloRangeArray con_EntireClose(env);
                  for(i = 0; i < general.numLinks; i++){
```

```
                        if(link[i].R > 0.5){
                                for(t = 0; t < general.T; t++){
                                        expr = dummy;
                                        m = 0;
                                        for(k = 0; k < general.numODs; k++){
                                                expr += y[i][k][t];
                                                m += OD[k].D;
                                        }
                                        expr = (1 - w[i][t]) * m - expr;
                                        IloRange EntireClose(env, 0, expr, IloInfinity, 0);
                                        con_EntireClose.add(EntireClose);
                                        expr.end();
                                        expr1.end();
                                }
                        }
                }
                SA_Opt.add(con_EntireClose);
//definition of w[i][t]
                IloRangeArray con_W(env);
                for(i = 0; i < general.numLinks; i++){
                        if(link[i].R > 0.5){
                                for(t = 0; t < general.T; t++){
                                        expr = dummy;
                                        expr1 = dummy1;
                                        for(m = 0; m < link[i].n; m++){
                                                expr1 += x[i][m][t];
                                        }
                                        expr = link[i].n - expr1;
                                        expr = expr - (1 - w[i][t]);
                                        con_W.add(IloRange(env, 0, expr, IloInfinity, 0));
                                        expr.end();
                                        expr = dummy;
                                        expr = link[i].n * (1 - w[i][t]);
                                        expr = expr - (link[i].n - expr1);
                                        con_W.add(IloRange(env, 0, expr, IloInfinity, 0));
                                        expr.end();
                                        expr1.end();
                                }
                        }
                        else{
                                for(t = 0; t < general.T; t++){
                                        expr = w[i][t];
                                        con_W.add(IloRange(env, 0, expr, 0, 0));
                                        expr.end();
                                }
                        }
                }
                SA_Opt.add(con_W);
//definition of v[i][m][t]
                IloRangeArray con_V(env);
                for(i = 0; i < general.numLinks; i++){
                        for(m = 0; m < link[i].n; m++){
                                if(link[i].R > 0.5){
                                        for(t = 0; t < link[i].p; t++){
                                                expr = v[i][m][t];
                                                con_V.add(IloRange(env, 0, expr, 0, 0));
                                                expr.end();
                                        }
```

```
                                for(t = link[i].p; t < general.T; t++){
                                        expr = dummy;
                                        for(a = 0; a < t - link[i].p + 1; a++){
                                                expr += s[i][m][a];
                                        }
                                        expr = expr - v[i][m][t];
                                        con_V.add(IloRange(env, 0, expr, 0, 0));
                                        expr.end();
                                }
                        }
                        else{
                                for(t = 0; t < general.T; t++){
                                        expr = v[i][m][t];
                                        con_V.add(IloRange(env, 0, expr, 0, 0));
                                        expr.end();
                                }
                        }
                }
        }
        SA_Opt.add(con_V);
//OD demand constraint
        IloRangeArray con_ODDemand(env);
        for(k = 0; k < general.numODs; k++){
                for(t = 0; t < general.T; t++){
                        expr = dummy;
                        expr1 = dummy;
                        for(i = 0; i < general.numLinks; i++){
                                if(link[i].FN == OD[k].ON){
                                        expr += y[i][k][t];
                                }
                                else if(link[i].TN == OD[k].ON){
                                        expr1 += y[i][k][t];
                                }
                        }
                        expr = expr - expr1;
                        con_ODDemand.add(IloRange(env, OD[k].D, expr, OD[k].D, 0));
                        expr.end();
                        expr1.end();
                        expr = dummy;
                        expr1 = dummy;
                        for(i = 0; i < general.numLinks; i++){
                                if(link[i].TN == OD[k].DN){
                                        expr += y[i][k][t];
                                }
                                else if(link[i].FN == OD[k].DN){
                                        expr1 += y[i][k][t];
                                }
                        }
                        expr = expr - expr1;
                        con_ODDemand.add(IloRange(env, OD[k].D, expr, OD[k].D, 0));
                        expr.end();
                }
        }
        SA_Opt.add(con_ODDemand);
//Flow Conservation Constraint
        IloRangeArray con_FlowConserve(env);
        for(j = 1; j < general.numNodes + 1; j++){
                for(k = 0; k < general.numODs; k++){
```

```
                        if(OD[k].ON != j && OD[k].DN != j){
                                for(t = 0; t < general.T; t++){
                                        expr = dummy;
                                        expr1 = dummy1;
                                        for(i = 0; i < general.numLinks; i++){
                                                if(link[i].FN == j){
        //OD flow out
                                                        expr += y[i][k][t];
                                                }
                                                else if(link[i].TN == j){
        //OD flow in
                                                        expr1 += y[i][k][t];
                                                }
                                        }
                                        expr = expr - expr1;
                                        con_FlowConserve.add(IloRange(env, 0, expr, 0,
0));
                                        expr.end();
                                        expr1.end();
                                }
                        }
                }
        }
        SA_Opt.add(con_FlowConserve);
//definition of z[i][t]
        IloRangeArray con_Z(env);
        for(i = 0; i < general.numLinks; i++){
                for(t = 0; t < general.T; t++){
                        expr = z[i][t];
                        IloRange Z(env, 0, expr, IloInfinity, 0);
                        con_Z.add(Z);
                        expr.end();
                        expr = dummy;
                        expr1 = dummy1;
                        for(k = 0; k < general.numODs; k++){
                                expr += y[i][k][t];
                        }
                        for(m = 0; m < link[i].n; m++){
                                expr1 += (-x[i][m][t] + general.theta*v[i][m][t]);
                        }
                        expr1 = (link[i].n + expr1) * link[i].u;
                        expr = expr - expr1;
                        expr = z[i][t] - expr;
                        IloRange Z1(env, 0, expr, IloInfinity, 0);
                        con_Z.add(Z1);
                        expr.end();
                        expr1.end();
                }
        }
        SA_Opt.add(con_Z);
//Define Objective
        expr = dummy;
        for(i = 0; i < general.numLinks; i++) {
                for(t = 0; t < general.T; t++){
                        expr1 = dummy1;
                        for(k = 0; k < general.numODs; k++){
                                expr1 += y[i][k][t];
                        }
```

```
                        expr += link[i].c * expr1 + z[i][t] * general.rho * link[i].c;
                        expr1.end();
                }
        }
        IloObjective obj(env, expr, IloObjective::Minimize);
        SA_Opt.add(obj);
        expr.end();
        expr1.end();
//Set Sch_Solver parameter
        SA_Solver.setParam(IloCplex::TiLim, timeLimitSA);             //restrict
the solving time for the entire SA problem to be 60 sec
        SA_Solver.setParam(IloCplex::EpGap, mipGap);
        SA_Solver.setParam(IloCplex::CutUp, cplexUB);
        SA_Solver.setParam(IloCplex::CutLo, cplexLB);
        SA_Solver.setParam(IloCplex::Threads, 0);
        SA_Solver.setParam(IloCplex::FPHeur, 2);  //use feasibility pump heuristic
        SA_Solver.setParam(IloCplex::MIPEmphasis, 1);    //let cplex put more effort
on finding a better feasible solution

        cout<<"\n";
        cout<<"Constraints of the SA problem are constructed\n";

        if (SA_Solver.solve()){
                TotalCost = SA_Solver.getObjValue();
                for(i = 0; i < general.numLinks; i++){
                        for(t = 0; t < general.T; t++){
                                for(m = 0; m < link[i].n; m++){
                                        if(SA_Solver.getValue(s[i][m][t]) > 0.5){
                                                s_o[i][m][t] =
SA_Solver.getValue(s[i][m][t]);
                                        }
                                        else if(SA_Solver.getValue(s[i][m][t]) < 0.5){
                                                s_o[i][m][t] = 0;
                                        }
                                        if(SA_Solver.getValue(x[i][m][t]) > 0.5){
                                                x_o[i][m][t] =
SA_Solver.getValue(x[i][m][t]);
                                        }
                                        else if(SA_Solver.getValue(x[i][m][t]) < 0.5){
                                                x_o[i][m][t] = 0;
                                        }
                                        if(SA_Solver.getValue(v[i][m][t]) > 0.5){
                                                v_o[i][m][t] =
SA_Solver.getValue(v[i][m][t]);
                                        }
                                        else if(SA_Solver.getValue(v[i][m][t]) < 0.5){
                                                v_o[i][m][t] = 0;
                                        }
                                }
                                if(SA_Solver.getValue(w[i][t]) > 0.5){
                                        w_o[i][t] = SA_Solver.getValue(w[i][t]);
                                }
                                else{
                                        w_o[i][t] = 0;
                                }
                        }
                }
                for(i = 0; i < general.numLinks; i++){
```

```
                    for(t = 0; t < general.T; t++){
                        for(k = 0; k < general.numODs; k++){
                            y_o[i][k][t] = SA_Solver.getValue(y[i][k][t]);
                        }
                        z_o[i][t] = SA_Solver.getValue(z[i][t]);
                    }
                }
                env.end();
            }
            else{
                cout<<endl;
                cout << "The original SA problem is not solved" << endl;
                cout << endl;
                //cin.get();
                env.end();
            }
        }
};

//%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%
//=========================================================================================
==========
// the class of FO Relaxation problem for variable v,s,x,w,y,z
//=========================================================================================
==========
class FO
{
public:
        IloEnv env;
        IloModel FO_Opt;                    //model for the FO problem
    IloCplex FO_Solver;                     //solver for the FO problem
    IloObjective obj;
    IloExpr expr;
    IloInt k, a, t, m, i, j;
        IloNumVar dummy, dummy1;
        IloRangeArray con_Nonpremp, con_StartDate, con_EntireClose, con_RepairClose,
con_ODDemand, con_FlowConserve, con_V, con_W, con_Z, con_Fix;
//desicion variables
        IloArray<IloArray<IloArray<IloNumVar> > > s, x, v, y;
        IloArray<IloArray<IloNumVar> > w, z;

public:
/*********************************Function that builds the model
FO*******************************/
        void Build(){
            FO_Opt = IloModel(env);
            FO_Solver = IloCplex(FO_Opt);
            IloExpr expr(env);                  //expression used while building model
            IloExpr expr1(env);                 //expression used while building mode
            IloNumVar dummy(env, 0, 0, ILOFLOAT);
            IloNumVar dummy1(env, 0, 0, ILOFLOAT);

//Define Variables
            s = IloArray<IloArray<IloArray<IloNumVar> > >(env, general.numLinks);
            //variable for the repair start date of each lane of each link
            x = IloArray<IloArray<IloArray<IloNumVar> > >(env, general.numLinks);
            //variable indicating whether the lane of a link is open or closed
```

```
		v = IloArray<IloArray<IloArray<IloNumVar> > >(env, general.numLinks);
		//if the mth lane of link (i,j) is repaired before day t, v_{ijmt}=1,
otherwise 0
		y = IloArray<IloArray<IloArray<IloNumVar> > >(env, general.numLinks);
		//flow variable used to ensure the schedule generated is feasible

		for (i = 0; i < general.numLinks; i++){
				//link index
			s[i] = IloArray<IloArray<IloNumVar> >(env, general.maxnumLanes);
			x[i] = IloArray<IloArray<IloNumVar> >(env, general.maxnumLanes);
			v[i] = IloArray<IloArray<IloNumVar> >(env, general.maxnumLanes);
			for (m = 0; m < link[i].n; m++){
		//lane index
				s[i][m] = IloArray<IloNumVar>(env, general.T);
				x[i][m] = IloArray<IloNumVar>(env, general.T);
				v[i][m] = IloArray<IloNumVar>(env, general.T);
				for(t = 0; t < general.T; t++){
				//date index
					s[i][m][t] = IloNumVar(env, 0, 1, ILOBOOL);
					x[i][m][t] = IloNumVar(env, 0, 1, ILOBOOL);
					v[i][m][t] = IloNumVar(env, 0, 1, ILOBOOL);
				}
			}
		}

		for (i = 0; i < general.numLinks; i++){
	//link index
			y[i] = IloArray<IloArray<IloNumVar> >(env, general.numODs);   //OD
index
			for(k = 0; k < general.numODs; k++){
				y[i][k] = IloArray<IloNumVar>(env, general.T);
	//date index
				for(t = 0; t < general.T; t++){
					y[i][k][t] = IloNumVar(env, 0, IloInfinity, ILOFLOAT);
				}
			}
		}

		w = IloArray<IloArray<IloNumVar> >(env, general.numLinks);
	//whether all the lanes of link (i,j) is closed on day t, if it is, w_{ijt}=1;
otherwise 0
		z = IloArray<IloArray<IloNumVar> >(env, general.numLinks);
		for (i = 0; i < general.numLinks; i++) {
			w[i] = IloArray<IloNumVar>(env, general.T);
			z[i] = IloArray<IloNumVar>(env, general.T);
			for (t = 0; t < general.T; t++){
				w[i][t] = IloNumVar(env, 0, 1, ILOBOOL);
				z[i][t] = IloNumVar(env, 0, IloInfinity, ILOFLOAT);
			}
		}
//Define Constraints
//Constraints ensuring once a lane is closed for repair, it won't open to serve the flows
until it is fully repaired
		IloRangeArray con_Nonpremp(env);
		for(i = 0; i < general.numLinks; i++) {
			if(link[i].R > 0.5){
				for(m = 0; m < link[i].n; m++){
					for(t = 0; t < general.T; t++){
```

```
                                        expr = dummy;
                                        if((t - link[i].p + 1) > 0){
                                               for(a = t - link[i].p + 1; a < t + 1;
a++){
                                                      expr += s[i][m][a];
                                               }
                                        }
                                        else{
                                               for(a = 0; a < t + 1; a++){
                                                      expr += s[i][m][a];
                                               }
                                        }
                                        expr = x[i][m][t] - expr;
                                        con_Nonpremp.add(IloRange(env, 0, expr, 0, 0));
                                        expr.end();
                                 }
                          }
                   }
            }
            FO_Opt.add(con_Nonpremp);
//Constraints ensuring each lane of the links to be repaired has a repair start date, and
those do not need repair does not have a repair start date
            IloRangeArray con_StartDate(env);
            for(i = 0; i < general.numLinks; i++) {
                   for(m = 0; m < link[i].n; m++){
                          expr = dummy;
                          for(t = 0; t < general.T; t++){
                                 expr += s[i][m][t];
                          }
                          if(link[i].R > 0.5){
                                 expr = expr - 1;
                          }
                          con_StartDate.add(IloRange(env, 0, expr, 0, 0));
                          expr.end();
                   }
            }
            FO_Opt.add(con_StartDate);
//All links that need maintenance have to be repaired before the end of the time horizon,
all links that don't need repair can't close any lanes at any time
            IloRangeArray con_RepairClose(env);
            for(i = 0; i < general.numLinks; i++) {
                   for(m = 0; m < link[i].n; m++){
                          expr = dummy;
                          for(t = 0; t < general.T; t++){
                                 expr += x[i][m][t];
                          }
                          if(link[i].R == 1){
                                 expr = expr - link[i].p;
                          }
                          con_RepairClose.add(IloRange(env, 0, expr, 0, 0));
                          expr.end();
                   }
            }
            FO_Opt.add(con_RepairClose);
//Constraint ensuring entirely closed link won't serve flows
            IloRangeArray con_EntireClose(env);
            for(i = 0; i < general.numLinks; i++){
                   if(link[i].R > 0.5){
```

```
                        for(t = 0; t < general.T; t++){
                                expr = dummy;
                                m = 0;
                                for(k = 0; k < general.numODs; k++){
                                        expr += y[i][k][t];
                                        m += OD[k].D;
                                }
                                expr = (1 - w[i][t]) * m - expr;
                                IloRange EntireClose(env, 0, expr, IloInfinity, 0);
                                con_EntireClose.add(EntireClose);
                                expr.end();
                                expr1.end();
                        }
                }
        }
        FO_Opt.add(con_EntireClose);
//definition of w[i][t]
        IloRangeArray con_W(env);
        for(i = 0; i < general.numLinks; i++){
                if(link[i].R > 0.5){
                        for(t = 0; t < general.T; t++){
                                expr = dummy;
                                expr1 = dummy1;
                                for(m = 0; m < link[i].n; m++){
                                        expr1 += x[i][m][t];
                                }
                                expr = link[i].n - expr1;
                                expr = expr - (1 - w[i][t]);
                                con_W.add(IloRange(env, 0, expr, IloInfinity, 0));
                                expr.end();
                                expr = dummy;
                                expr = link[i].n * (1 - w[i][t]);
                                expr = expr - (link[i].n - expr1);
                                con_W.add(IloRange(env, 0, expr, IloInfinity, 0));
                                expr.end();
                                expr1.end();
                        }
                }
                else{
                        for(t = 0; t < general.T; t++){
                                expr = w[i][t];
                                con_W.add(IloRange(env, 0, expr, 0, 0));
                                expr.end();
                        }
                }
        }
        FO_Opt.add(con_W);
//definition of v[i][m][t]
        IloRangeArray con_V(env);
        for(i = 0; i < general.numLinks; i++){
                for(m = 0; m < link[i].n; m++){
                        if(link[i].R > 0.5){
                                for(t = 0; t < link[i].p; t++){
                                        expr = v[i][m][t];
                                        con_V.add(IloRange(env, 0, expr, 0, 0));
                                        expr.end();
                                }
                                for(t = link[i].p; t < general.T; t++){
```

```
                                        expr = dummy;
                                        for(a = 0; a < t - link[i].p + 1; a++){
                                                expr += s[i][m][a];
                                        }
                                        expr = expr - v[i][m][t];
                                        con_V.add(IloRange(env, 0, expr, 0, 0));
                                        expr.end();
                                }
                        }
                        else{
                                for(t = 0; t < general.T; t++){
                                        expr = v[i][m][t];
                                        con_V.add(IloRange(env, 0, expr, 0, 0));
                                        expr.end();
                                }
                        }
                }
        }
        FO_Opt.add(con_V);
//OD demand constraint
        IloRangeArray con_ODDemand(env);
        for(k = 0; k < general.numODs; k++){
                for(t = 0; t < general.T; t++){
                        expr = dummy;
                        expr1 = dummy;
                        for(i = 0; i < general.numLinks; i++){
                                if(link[i].FN == OD[k].ON){
                                        expr += y[i][k][t];
                                }
                                else if(link[i].TN == OD[k].ON){
                                        expr1 += y[i][k][t];
                                }
                        }
                        expr = expr - expr1;
                        con_ODDemand.add(IloRange(env, OD[k].D, expr, OD[k].D, 0));
                        expr.end();
                        expr1.end();
                        expr = dummy;
                        expr1 = dummy;
                        for(i = 0; i < general.numLinks; i++){
                                if(link[i].TN == OD[k].DN){
                                        expr += y[i][k][t];
                                }
                                else if(link[i].FN == OD[k].DN){
                                        expr1 += y[i][k][t];
                                }
                        }
                        expr = expr - expr1;
                        con_ODDemand.add(IloRange(env, OD[k].D, expr, OD[k].D, 0));
                        expr.end();
                }
        }
        FO_Opt.add(con_ODDemand);
//Flow Conservation Constraint
        IloRangeArray con_FlowConserve(env);
        for(j = 1; j < general.numNodes + 1; j++){
                for(k = 0; k < general.numODs; k++){
                        if(OD[k].ON != j && OD[k].DN != j){
```

```
                                    for(t = 0; t < general.T; t++){
                                          expr = dummy;
                                          expr1 = dummy1;
                                          for(i = 0; i < general.numLinks; i++){
                                                if(link[i].FN == j){
      //OD flow out
                                                      expr += y[i][k][t];
                                                }
                                                else if(link[i].TN == j){
      //OD flow in
                                                      expr1 += y[i][k][t];
                                                }
                                          }
                                          expr = expr - expr1;
                                          con_FlowConserve.add(IloRange(env, 0, expr, 0,
0));
                                          expr.end();
                                          expr1.end();
                                    }
                              }
                        }
                  }
                  FO_Opt.add(con_FlowConserve);
//definition of z[i][t]
                  IloRangeArray con_Z(env);
                  for(i = 0; i < general.numLinks; i++){
                        for(t = 0; t < general.T; t++){
                              expr = z[i][t];
                              IloRange Z(env, 0, expr, IloInfinity, 0);
                              con_Z.add(Z);
                              expr.end();
                              expr = dummy;
                              expr1 = dummy1;
                              for(k = 0; k < general.numODs; k++){
                                    expr += y[i][k][t];
                              }
                              for(m = 0; m < link[i].n; m++){
                                    expr1 += (-x[i][m][t] + general.theta*v[i][m][t]);
                              }
                              expr1 = (link[i].n + expr1) * link[i].u;
                              expr = expr - expr1;
                              expr = z[i][t] - expr;
                              IloRange Z1(env, 0, expr, IloInfinity, 0);
                              con_Z.add(Z1);
                              expr.end();
                              expr1.end();
                        }
                  }
                  FO_Opt.add(con_Z);
//Define Objective
                  expr = dummy;
                  for(i = 0; i < general.numLinks; i++) {
                        for(t = 0; t < general.T; t++){
                              expr1 = dummy1;
                              for(k = 0; k < general.numODs; k++){
                                    expr1 += y[i][k][t];
                              }
                              expr += link[i].c * expr1 + z[i][t] * general.rho * link[i].c;
```

```
                        expr1.end();
                }
            }
            IloObjective obj(env, expr, IloObjective::Minimize);
            FO_Opt.add(obj);
            expr.end();
            expr1.end();
//Set Sch_Solver parameter
            FO_Solver.setParam(IloCplex::TiLim, timeLimitFO);             //restrict
the solving time for each FO subproblem to be 30 sec
            FO_Solver.setParam(IloCplex::EpGap, mipGap);
            FO_Solver.setParam(IloCplex::CutUp, cplexUB);
            FO_Solver.setParam(IloCplex::CutLo, cplexLB);
            FO_Solver.setParam(IloCplex::Threads, 0);
            FO_Solver.setParam(IloCplex::FPHeur, 2);  //use feasibility pump heuristic
to find a good feasible point to start with
            FO_Solver.setParam(IloCplex::MIPEmphasis, 1);    //let cplex put more effort
on finding a better feasible solution

            cout<<"\n";
            cout<<"Constraints of the FO problem are constructed\n";
      }
/*********************function that adds the constraint fixing the values of scheduling
variables of links not in the currect FO batch ***************/
      void Fix(int FOBatchNum){
            con_Fix = IloRangeArray(env);
            for(i = 0; i < general.numLinks; i++){
                  if(link[i].obindex != FOBatchNum){
                        for(t = 0; t < general.T; t++){
                              /*IloRange fixw(env, w_o[i][t], w[i][t], w_o[i][t], 0);
      //fix all the variables of the link, when time horizon is long, removing the
con_Fix takes a very long time
                              con_Fix.add(fixw);
                              for(m = 0; m < link[i].n; m++){
                                    IloRange fixs(env, s_o[i][m][t], s[i][m][t],
s_o[i][m][t], 0);
                                    IloRange fixx(env, x_o[i][m][t], x[i][m][t],
x_o[i][m][t], 0);
                                    IloRange fixv(env, v_o[i][m][t], v[i][m][t],
v_o[i][m][t], 0);
                                    con_Fix.add(fixs);
                                    con_Fix.add(fixx);
                                    con_Fix.add(fixv);
                              }*/
                              for(m = 0; m < link[i].n; m++){
                                    if(s_o[i][m][t] == 1){
                                          IloRange fixs(env, 1, s[i][m][t], 1, 0);
      //just fix the s variable when s=1
                                          con_Fix.add(fixs);
                                    }
                              }
                        }
                  }
            }
            FO_Opt.add(con_Fix);
            //cin.get();
      }
```

```
/*******************function that deletes the constraint fixing the value of scheduling
variables of links not in the currect FO batch **************/
	void Unfix(){
		FO_Opt.remove(con_Fix);
		con_Fix.end();
	}
/************************function that solves the Fix and Optimize problem for the
optimizatio batch***************/
	bool Solve(int FOBatchNum){
		IloCplex::Callback abortcriteria = FO_Solver.use(AbortCriteria(env,
TotalCostFO));
		if (FO_Solver.solve()){
			cout << endl;
			cout << "The best lower bound of current FO subproblem is: " <<
FO_Solver.getBestObjValue() << endl;
			//cin.get();
			if(FO_Solver.getBestObjValue()>TotalCostFO){
				aborted = 1;
			}
			if(TotalCostFO > FO_Solver.getObjValue()){
				TotalCostFO = FO_Solver.getObjValue();
				for(i = 0; i < general.numLinks; i++){
					if(link[i].obindex == FOBatchNum){
						for(t = 0; t < general.T; t++){
							for(m = 0; m < link[i].n; m++){

	if(FO_Solver.getValue(s[i][m][t]) > 0.5){
									s_o[i][m][t] =
FO_Solver.getValue(s[i][m][t]);
							}
							else
if(FO_Solver.getValue(s[i][m][t]) < 0.5){
									s_o[i][m][t] = 0;
							}

	if(FO_Solver.getValue(x[i][m][t]) > 0.5){
									x_o[i][m][t] =
FO_Solver.getValue(x[i][m][t]);
							}
							else
if(FO_Solver.getValue(x[i][m][t]) < 0.5){
									x_o[i][m][t] = 0;
							}

	if(FO_Solver.getValue(v[i][m][t]) > 0.5){
									v_o[i][m][t] =
FO_Solver.getValue(v[i][m][t]);
							}
							else
if(FO_Solver.getValue(v[i][m][t]) < 0.5){
									v_o[i][m][t] = 0;
							}
						}
						if(FO_Solver.getValue(w[i][t]) > 0.5){
							w_o[i][t] =
FO_Solver.getValue(w[i][t]);
						}
						else{
```

```
                                                w_o[i][t] = 0;
                                        }
                                }
                        }
                        for(i = 0; i < general.numLinks; i++){
                                for(t = 0; t < general.T; t++){
                                        for(k = 0; k < general.numODs; k++){
                                                y_o[i][k][t] =
FO_Solver.getValue(y[i][k][t]);
                                        }
                                        z_o[i][t] = FO_Solver.getValue(z[i][t]);
                                }
                        }
                        //cout << endl;
                        //cout << "Best schedule for the links in FO batch is
updated" << endl;
                        //cin.get();
                    }
                }
                cout<<"\n";
                cout<<"The Fix and Optimize problem is solved, the objective is
"<<TotalCostFO<<"\n";
                cout << endl;
                stringstream ss8;
                stringstream ss9;
                stringstream ss10;
                string fileName8;
                string fileName9;
                string fileName10;
                ofstream ofileSol8;
                ofstream ofileSol9;
                ofstream ofileSol10;
                ss8 << "FO_s_x.csv";
                fileName8 = ss8.str();
                ofileSol8.open(fileName8.c_str());
                ss9 << "FO_y.csv";
                fileName9 = ss9.str();
                ofileSol9.open(fileName9.c_str());
                ss10 << "FO_z_w_mu.csv";
                fileName10 = ss10.str();
                ofileSol10.open(fileName10.c_str());
                ofileSol8 << "Schedule when time horizon = " << general.T << endl;
                ofileSol8 << "OBJECTIVE: " << TotalCostFO << endl;
                ofileSol8 << "Values of the s and x matrice: " << endl;
                ofileSol8 << "Link ID ," << "From Node ," << "To Node ," << "Lane
No. ," << "Date ," << "s_value ," << "x_value ," << "v_value" << endl;
                for(i = 0; i < general.numLinks; i++){
                for(m = 0; m < link[i].n; m++){
                        for(t = 0;t < general.T; t++){
                                ofileSol8 << link[i].ID << "," << link[i].FN << "," <<
link[i].TN << "," << m + 1 << "," << t + 1 << "," << s_o[i][m][t] << "," << x_o[i][m][t]
<< "," << v_o[i][m][t] << endl;
                        }
                }
                 }
                ofileSol8.close();
                ofileSol9 << "Schedule when time horizon = " << general.T << endl;
                ofileSol9 << "OBJECTIVE: " << TotalCostFO << endl;
```

```
				ofileSol9 << "Values of the y matrix: " << endl;
				ofileSol9 << "Link ID ," << "From Node ," << "To Node ," << "OD 
ID ," << "Origin Node ," << "Destination Node ," << "Date ," << "y_value ," << endl;
				for(i = 0; i < general.numLinks; i++){
				for(k = 0; k < general.numODs; k++){
					for(t = 0;t < general.T; t++){
						ofileSol9 << link[i].ID << "," << link[i].FN << "," << 
link[i].TN << "," << OD[k].ID << "," << OD[k].ON << "," << OD[k].DN << "," << t + 1 << 
"," << y_o[i][k][t] << endl;
					}
				}
				 }
				ofileSol9.close();
				ofileSol10 << "Schedule when time horizon = " << general.T << endl;
				ofileSol10 << "OBJECTIVE: " << TotalCostFO << endl;
				ofileSol10 << "Values of the z and w matrice: " << endl;
				ofileSol10 << "Link ID ," << "From Node ," << "To Node ," << 
"Date ," << "z_value ," << "w_value " << endl;
				for(i = 0; i < general.numLinks; i++){
				for(t = 0;t < general.T; t++){
					ofileSol10 << link[i].ID << "," << link[i].FN << "," << 
link[i].TN << ","  << t + 1 << "," << z_o[i][t] << "," << w_o[i][t] << endl;
				}
				}
				ofileSol10.close();
				abortcriteria.end();
				return true;
			}
			else{
				cout<<endl;
				cout << "The FO relaxation problem is not solved" << endl;
				cout << endl;
				//cin.get();
				return false;
			}
		}
/***********************function that set the solver spend more effort on finding better 
lower bounds***********************/
		void BLB(){
			FO_Solver.setParam(IloCplex::MIPEmphasis, 2);
		}
/***********************function that set the solver spend more effort on finding better 
feasible solutions***********************/
		void BFS(){
			FO_Solver.setParam(IloCplex::MIPEmphasis, 1);
		}
};

//%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%
//==========================================================================================
==========
//main function
//==========================================================================================
==========
int main(int argc, char** argv) {

		time_t startTime, endTime, startTimeC, endTimeC;
```

```
	double runTime;
	int i, j, k, m, t;
	int FOBatchNum;
	int iter_num;		//record the number of iterations performed

	/*************************************************************************
*****************/
	InputData();  //obtain network information
	/*************************************************************************
*****************/

	/********************************** Initialization
**********************************************/
	Initialization();

	/******************************** Construct the models of the entire SA problem
***************************/
	/*************************************************************************
***************************/
	startTime = time(0); //time stamp of the starting point of the whole solving
process
	SA modelSA;
	startTimeC = time(0);	//time stamp of cples starts solving the entire SA
problem
	modelSA.BuildSolveSA();
	endTimeC = time(0);		//time stamp of cplex stops solving the entire SA
problem

	FO modelFO;
	modelFO.Build();

	if(difftime(endTimeC, startTimeC)<timeLimitSA){
		cout << endl;
		//cin.get();
		goto stop;
	}
	else{
		cout << endl;
		cout << "Optimality is not reached in " << timeLimitSA << " seconds.
Continue to fix and optimize procedure." << endl;
		//cin.get();
		goto FOStart;
	}

	/*************************************************************************
*************************************************************************
*********************************/
	/*************************************************************************
*************** FO Iteration
*************************************************************************
*****/
	FOStart:
	iter_num = 1;
	numBatches = 2.0;		//start with 2 batches
	int numLinpBat;			//number of links per batch
	bool LonSolTime;	//flag variable indicating that at least one FO subproblem of
current batching has solving time longer than 30 sec
	cout << endl;
```

```
	cout << "Start of FO iteration." << endl;
	//cin.get();
	while(iter_num < 10000){
		vector<int> LintoRep;		//the set of links that need repair and haven't been assigned to a batch
		for(i = 0; i < general.numLinks; i++){
			if(link[i].R == 1){
				LintoRep.push_back(i);		//construct the set with the IDs of links that need repair
			}
		}
		numLinpBat = floor(LintoRep.size()/numBatches+0.5);		//calculate the number of links in a batch
		cout << endl;
		cout << "Number of links per batch: " << numLinpBat << endl;
		//cin.get();
		cout << endl;
		cout << "Total number of links that need repair is " << LintoRep.size() << endl;
		for(j = 1; j < numBatches+1; j++){
			for(k = 1; k < numLinpBat+1; k++){
				int index;
				int ID;
				if(k == 1 && LintoRep.size() == 1){
					ID = LintoRep.at(0);		//if there is one link to be batched in a new batch, put the link in the batch formed in last iteration
					link[ID].obindex = j - 1;
					LintoRep.erase(LintoRep.begin());
					numBatches = numBatches - 1;	//when the only link left is put in the batch in last iteration, the total number of batches will be 1 less
				}
				else{
					srand( time(0) );
					index = rand() % LintoRep.size();  //randomly pick a link from the set, assign it to the current batch
					ID = LintoRep.at(index);
					link[ID].obindex = j;
					LintoRep.erase(LintoRep.begin()+index);
		//after the assignment, delete the link ID from the set
				}
				if(LintoRep.empty()){
					goto endofbatchingFO;
				}
			}
		}
		endofbatchingFO:
		cout << endl;
		cout << "Number of batches to go through the fix and optimize process: " << numBatches << endl;
		cout << endl;
		//cin.get();
		cout << "All links that need repair are grouped into batches." << endl;
		cout << endl;
		for(j = 1; j < numBatches+1; j++){
			cout << "Batch " << j << ":" << endl;
			for(i = 0; i < general.numLinks; i++){
				if(link[i].obindex == j){
					cout << "Link[" << i+1 << "]" << endl;
```

```
                        }
                }
                cout << endl;
        }
        //cin.get();
        LonSolTime = 0;             //before the solve the FO subproblems of current
batch set the LonSolTime 0
        aborted = 0;
        for(FOBatchNum = 1; FOBatchNum < numBatches+1; FOBatchNum++){
                modelFO.Fix(FOBatchNum);
                cout << endl;
                cout << "Schedules of links other than the Fix and Optimize Batch "
<< FOBatchNum << " are fixed." << endl;
                startTimeC = time(0);             //record the start time of the
solving process of current FO subproblem
                modelFO.Solve(FOBatchNum);
                endTimeC = time(0);               //record the end time of the
solving process of current FO subproblem
                //cout << endl;
                //cout << difftime(endTimeC, startTimeC) << endl;
                //cin.get();
                if(difftime(endTimeC, startTimeC)>timeLimitFO-1){
                        LonSolTime = 1;                   //if one of the subproblems
has solving time longer than 30 sec, set LonSolTime 1
                        //cin.get();
                }
                modelFO.Unfix();
                cout << endl;
                cout << "Schedules of links other than the Fix and Optimize Batch "
<< FOBatchNum << " are unfixed." << endl;
        }
        cout << endl;
        cout << "iter_num = " << iter_num << endl;
        cout << "LonSolTime = " << LonSolTime << endl;
        cout << "TotalCostFO = " << TotalCostFO << endl;
        cout << "TotalCost = " << TotalCost << endl;
        cout << "aborted  = " << aborted << endl;
        //cin.get();
        if((TotalCost-TotalCostFO)/TotalCost > 0.000001){
                TotalCost = TotalCostFO;    //if the best objective of current
batching is better than the best objective in record, re-batch the links again and do the
fix and optimize iteration
                iter_num = iter_num + 1;
        }
        else{
                if(iter_num < numBatches+1){
                        iter_num = iter_num + 1;    //if no better schedule if found
in current batching and the number of iterations performed is less than 3, random batch
again with the same batch number and resolve the FO subproblems
                }
                else{
                        if(LonSolTime == 1){
                                if(numLinpBat > 3){
                                        numBatches = numBatches + 1;       //increase
the number of batches by 1 only if the number of links in per batch is greater than 2
                                        iter_num = 1;                              //if
no better schedule is found and the solving time of at least one FO subproblem is longer
than 30 sec, reset the iter_num
```

```
                                }
                                else{
                                        goto stop;                      //if the
numLinpBat=3, and no better schedule is found, stop
                                }
                        }
                        else{
                                goto stop;                              //if the
LonSolTime=0 and no better schedule is found, stop
                        }
                }
        }
        if(LonSolTime == 0){
                modelFO.BLB();      //if no FO subproblem takes long solving time in
current batching, set the solver on finding better lower bounds
        }
        else if(LonSolTime == 1){
                modelFO.BFS();      //if there is FO subproblem taking long solving
time in current batching, set the solver on finding better feasible solutions
        }
    }
    /*************************************************************************************
*************************************************************************************
**********************************/
    stop:
    stringstream ss1;
    stringstream ss2;
    stringstream ss3;
    string fileName1;
    string fileName2;
    string fileName3;
    ofstream ofileSol1;
    ofstream ofileSol2;
    ofstream ofileSol3;
    ss1 << "c_network_repair_s_x.csv";
    fileName1 = ss1.str();
    ofileSol1.open(fileName1.c_str());
    ss2 << "c_network_repair_y.csv";
    fileName2 = ss2.str();
    ofileSol2.open(fileName2.c_str());
    ss3 << "c_network_repair_z_w.csv";
    fileName3 = ss3.str();
    ofileSol3.open(fileName3.c_str());

    cout << endl;
    cout << "Problem instance is solved." << endl;
    cout << "The objective value is :" << TotalCost << "." << endl;
    ofileSol1 << "Schedule when time horizon = " << general.T << endl;
    ofileSol1 << "OBJECTIVE: " << TotalCost << endl;
    ofileSol1 << "Values of the s and x matrice: " << endl;
    ofileSol1 << "Link ID ," << "From Node ," << "To Node ," << "Lane No. ," <<
"Date ," << "s_value ," << "x_value ," << "v_value" << endl;
    for(i = 0; i < general.numLinks; i++){
    for(m = 0; m < link[i].n; m++){
            for(t = 0;t < general.T; t++){
                    ofileSol1 << link[i].ID << "," << link[i].FN << "," << link[i].TN <<
"," << m + 1 << "," << t + 1 << "," << s_o[i][m][t] << "," << x_o[i][m][t] << "," <<
v_o[i][m][t] << endl;
```

```
			}
		}
		 }
		ofileSol1.close();
		ofileSol2 << "Schedule when time horizon = " << general.T << endl;
		ofileSol2 << "OBJECTIVE: " << TotalCost << endl;
		ofileSol2 << "Values of the y matrix: " << endl;
		ofileSol2 << "Link ID ," << "From Node ," << "To Node ," << "OD ID ," << "Origin
Node ," << "Destination Node ," << "Date ," << "y_value ," << endl;
		for(i = 0; i < general.numLinks; i++){
		for(k = 0; k < general.numODs; k++){
			for(t = 0;t < general.T; t++){
				ofileSol2 << link[i].ID << "," << link[i].FN << "," << link[i].TN <<
"," << OD[k].ID << "," << OD[k].ON << "," << OD[k].DN << "," << t + 1 << "," <<
y_o[i][k][t] << endl;
			}
		}
		}
		ofileSol2.close();
		ofileSol3 << "Schedule when time horizon = " << general.T << endl;
		ofileSol3 << "OBJECTIVE: " << TotalCost << endl;
		ofileSol3 << "Values of the z and w matrice: " << endl;
		ofileSol3 << "Link ID ," << "From Node ," << "To Node ," << "Date ," <<
"z_value ," << "w_value" << endl;
		for(i = 0; i < general.numLinks; i++){
		for(t = 0;t < general.T; t++){
			ofileSol3 << link[i].ID << "," << link[i].FN << "," << link[i].TN << ","
<< t + 1 << "," << z_o[i][t] << "," << w_o[i][t] << endl;
		}
		}
		ofileSol3.close();

		endTime = time(0);		//time stamp of the time point when the entire program
ends
		runTime = difftime(endTime, startTime) / 60;
		cout << endl;
		cout << "Total Solving Time: \t" << runTime << " min" << endl;
		cin.get();
		return 0;
}


//=============================================================================================
===========
//InputData function
//=============================================================================================
===========
void InputData(){

//if stream
		ifstream modelParams,
				linkData,
				ODData;

//read general data
		modelParams.open("Input_modelParameters.txt");
		modelParams >> timeLimitSA;
		modelParams >> timeLimitFO;
```

```
	modelParams >> mipGap;
	modelParams >> cplexUB;
	modelParams >> cplexLB;
	modelParams >> general.numNodes;
	modelParams >> general.numLinks;
	modelParams >> general.numODs;
	modelParams >> general.maxnumLanes;
	modelParams >> general.rho;
	modelParams >> general.T;
	modelParams >> general.theta;
	modelParams.close();

// read link data
	linkData.open("Input_links.txt");
	int j = 1;
	int i = 0;
	for(i = 0; i < general.numLinks; i++){
		link.push_back(linkInfo());
		linkData >> link[i].ID;
		linkData >> link[i].FN;
		linkData >> link[i].TN;
		linkData >> link[i].n;
		linkData >> link[i].u;
		linkData >> link[i].c;
		linkData >> link[i].p;
		linkData >> link[i].R;
		if(link[i].R == 1){
			link[i].obindex = 0;
		}
	}
	linkData.close();

// read generator parameters
	ODData.open("Input_ODs.txt");
	int D = 0;
	for(int k = 0; k < general.numODs; k++){
		OD.push_back(ODInfo());
		ODData >> OD[k].ID;
		ODData >> OD[k].ON;
		ODData >> OD[k].DN;
		ODData >> OD[k].D;
		D = D + OD[k].D;
	}
	sumD = D;
	ODData.close();

	cout << "Import Data complete." << endl;
}

//=============================================================================================
===========
//InitSolution function
//=============================================================================================
===========
void Initialization(){
	int i, j, m, k, t;
	TotalCost = 1e38;
	TotalCostFO = 1e38;
```

```
	for (i = 0; i < general.numLinks; i++){
		//link index
		s_o.push_back(vector<vector<bool> >());
		x_o.push_back(vector<vector<bool> >());
		y_o.push_back(vector<vector<float> >());
		v_o.push_back(vector<vector<bool> >());

		for (m = 0; m < link[i].n; m++){
		//lane index
			s_o[i].push_back(vector<bool>());
			x_o[i].push_back(vector<bool>());
			v_o[i].push_back(vector<bool>());

			for(t = 0; t < general.T; t++){
			//date index
				s_o[i][m].push_back(0);
				x_o[i][m].push_back(0);
				v_o[i][m].push_back(0);
			}
		}
		for (k = 0; k < general.numODs; k++){
			y_o[i].push_back(vector<float>());			//OD index
			for (t = 0; t < general.T; t++){
				y_o[i][k].push_back(0.0);
	//date index
			}
		}
	}
	for (i = 0; i < general.numLinks; i++) {
		z_o.push_back(vector<float>());				//link
index
		w_o.push_back(vector<bool>());
		for (t = 0; t < general.T; t++){
			z_o[i].push_back(0.0);
	//date index
			w_o[i].push_back(0);
		}
	}
}
```