2004

Efficient path search in intermodal transportation optimization

Daniel Y. Yankov

Rochester Institute of Technology

Efficient Path Search in Intermodal Transportation Optimization

A Thesis

Submitted in partial fulfillment of the requirements for the degree of

Master of Science in Industrial Engineering

In the

Department of Industrial and Systems Engineering

Kate Gleason College of Engineering

By

Daniel Y. Yankov

MS, IE candidate

DEPARTMENT OF INDUSTRIAL AND SYSTEMS ENGINEERING

KATE GLEASON COLLEGE OF ENGINEERING

ROCHESTER INSTITUTE OF TECHNOLOGY

ROCHESTER, NEW YORK

CERTIFICATE OF APPROVAL

M.S. DEGREE THESIS

The M.S. Degree Thesis of Daniel Y. Yankov

has been examined and approved by the thesis committee

as satisfactory for the thesis requirement

for the Master of Science degree.

Approved by:

Moses Sudit

Dr. Moses Sudit, Ph.D., Primary advisor

Sanjay Joshi

Dr. Sanjay Joshi, Ph.D. Advisor

Thesis/Dissertation Author Permission Statement

Title of thesis or dissertation: EFFICIENT PATH SEARCH IN INTERMODAL TRANSPORT OPTIMIZATION

Name of author: DANIEL Y. YANKOV
Degree: MS
Program: INDUSTRIAL ENGINEERING
College: COE KATE GLISON

I understand that I must submit a print copy of my thesis or dissertation to the RIT Archives, per current RIT guidelines for the completion of my degree. I hereby grant to the Rochester Institute of Technology and its agents the non-exclusive license to archive and make accessible my thesis or dissertation in whole or in part in all forms of media in perpetuity. I retain all other ownership rights to the copyright of the thesis or dissertation. I also retain the right to use in future works (such as articles or books) all or part of this thesis or dissertation.

Print Reproduction Permission Granted:

I, DANIEL YANKOV, hereby **grant permission** to the Rochester Institute Technology to reproduce my print thesis or dissertation in whole or in part. Any reproduction will not be for commercial use or profit.

Signature of Author: Daniel Yankov Date: 7/26/04

Print Reproduction Permission Denied:

I, _____, hereby **deny permission** to the RIT Library of the Rochester Institute of Technology to reproduce my print thesis or dissertation in whole or in part.

Signature of Author: _____ Date: _____

Inclusion in the RIT Digital Media Library Electronic Thesis & Dissertation (ETD) Archive

I, _____, additionally grant to the Rochester Institute of Technology Digital Media Library (RIT DML) the non-exclusive license to archive and provide electronic access to my thesis or dissertation in whole or in part in all forms of media in perpetuity.

I understand that my work, in addition to its bibliographic record and abstract, will be available to the world-wide community of scholars and researchers through the RIT DML. I retain all other ownership rights to the copyright of the thesis or dissertation. I also retain the right to use in future works (such as articles or books) all or part of this thesis or dissertation. I am aware that the Rochester Institute of Technology does not require registration of copyright for ETDs.

I hereby certify that, if appropriate, I have obtained and attached written permission statements from the owners of each third party copyrighted matter to be included in my thesis or dissertation. I certify that the version I submitted is the same as that approved by my committee.

Signature of Author: _____ Date: _____

Abstract

As the economies of the world become more interrelated and Supply Chains are globalizing, the need arises to create efficient transportation network. This reality in conjunction with conservation of fuel and environmental friendliness gives rise to the research of Efficient Intermodal Transportation System. In particular, the underutilization of railroads in the United States motivates us to research the development of optimal procedures in the transportation of containers in a rail network.

With this thesis we search for a cost, time and capacity effective algorithm for solving transportation problem in a graph of intermodal centers (IMC's). We consider discrete model of the real time dynamic situation when all the arcs of the input graph can be affected by changes in their costs, the transportation means have limited and different container capacities at each IMC, and all the nodes (IMC's) can be visited more than once either by different transport means or at different time. This is more general and real situation than the ones considered in the literature so far.

The resulting optimization problem is computational intractable (NP-hard), which creates the necessity to develop, implement and test efficient heuristic optimization techniques. We will use Shortest Path Problem (SPP) as the basis for the development of three heuristics. Because of the nature of the problem and application, shortest path procedures provide a very flexible and computationally efficient technique for our model.

We will compare the three heuristics with the optimal solution for small size problems for which we could find optimality. Furthermore, we will demonstrate that one of the heuristics perform very well when the fixed costs of running transportation modes is the dominant aspect of the cost structure.

1. Formal Problem Statement and Definitions

According to the researchers the intermodal transportation should be defined as: "the shipment of cargo and movement of people involving more than one mode of transportation during a single, seamless journey "[3].

In this paper we consider containerized freight transportation within a set of nodes (IMC's) where parallel arcs with different weights are allowed – i.e. a container can be transported by different modes on different costs within different transportation time.
The problem is quite general. To provide a practical solution – an easily computed algorithm for efficient path search considering cost, time and capacity constraints within a system of IMC's, we apply different techniques and strategies.

First we calculate the optimal solution of a series of simplified transportation examples. Then we compare the results of the same problems generated with three heuristic. Finally we analyze the results under statistical and optimization point of view.

Definitions:

Transportation Job (job): The amount of containers with the same start and destination IMC, which must be transferred by the same deadline time *T*.

Transportation mode: Used transportation mean like train, truck, ship, and airplane.

We define the intermodal effective path algorithm problem as follows:

On the static graph of IMC's, develop a practically applicable algorithm to compute the most cost efficient paths from given origin node(s) to their destination(s), transportation routes and modes considering the time constraints and the capacities of the modes.

Mathematical Modeling of the example problem

Notation:

$i = 1,\ldots, N$ – job index

I – job set, $I = \{i : i = 1, \ldots, N\}$

$j = 1,\ldots, O_i$ – container index, O_i = number of containers of job i

J_i – container set for job i, $J_i = \{j : j = 1,\ldots, O_i\}$

$r = 1,\ldots,R$ – transportation mean index

$k = 1,\ldots,K$ – IMC index

(k-1) , k , (k+1) = Consecutive (Middle) IMC's in the route

A_{rk} = Arrival time of transportation mean *r* at IMC *k*

D_{rk} = Depart time of transportation mean *r* from IMC *k*

T_i = Deadline time of job *i*

P_{irk} = Container capacity of transportation mean *r* at IMC *k* during job *i*

Input parameters:

a_r = fixed cost of operation of transportation mean *r*

b_{ijk} = cost of reloading of container *j* of job *i* at IMC *k*

$c_{ijrk,(k+1)}$ = cost of transportation of container *j* of job *i* by transportation mean *r* from IMC *k* to *k+1*.

Variables:

$x_{ijrk,(k+1)}$: 1, if container *j* of job *i* is transported by transportation mean *r* from IMC *k* to *(k+1)*;
0, otherwise

y_r : 1, if transportation mean *r* is used at all;
0, otherwise

z_{ijrk} : 1, if transportation mean *r* is changed for transportation of container *j* of job *i* at IMC *k*;
0, otherwise

Objective Function:

$$\text{Min} \sum_{i=1}^{N}\sum_{j=1}^{Oi}\sum_{r=1}^{R}\sum_{k=1}^{K}\left[x_{ijrk(k+1)} * c_{ijrk(k+1)}\right] + \sum_{r=1}^{R}\left(a_r * y_r\right) + \sum_{i=1}^{Ni}\sum_{j=1}^{Oi}\sum_{r=1}^{R}\sum_{k=1}^{K}\left(b_{ijr} * z_{ijrk}\right)$$

Minimizing the sum of transportation fixed cost, variable cost and changing cost

Subject to:

1. $\sum_{r=1}^{R} x_{ijr(k-1)k} - \sum_{r=1}^{R} x_{ijrk(k+1)} = \begin{cases} 0, \text{ if } k \text{ is a mid IMC in the route of } r; \\ 1, \text{ if } k \text{ is the destination IMC of } r; \\ -1, \text{ if } k \text{ is the starting IMC of } r, \quad \text{for } \forall \text{ i,j,k;} \end{cases}$

Continuity of flow

2. $\sum_{i=1}^{N} \sum_{j=1}^{Oi} \sum_{k=1}^{K} x_{ijrk} \le L\, y_r$, for $\forall\, r = 1 \ldots R$, and L –sufficiently large

If transportation ***r*** is used with its fixed cost

3. $x_{ijr(k-1),k} + \sum_{\substack{r'=1 \\ r' \neq r}}^{R} x_{ijr'k,(k+1)} \le z_{ijk+1} + 1$, for $\forall$ i,j,k

If transportation ***r*** is changed at ITC ***k***

4. $A_{rk} * x_{ijr(k-1),k} \le D_{rk} * x_{ijrk,(k+1)}$, for $\forall$ i,j,r,k

The arrival time at IMC ***k*** is less than the depart time from that ITC

5. $A_{rk} * x_{ijrk,(k+1)} \le T_i$, for $\forall$ i, r, k = 9,10

Jobs deadline limit

Bounds:

$x_{ijrk,(k+1)}$ – binary;
y_r – binary;
z_{ijrk} – binary.

2. Literature review

Because of their fundamental theoretical consideration and efficiency, first we present the *Shortest Path Tree* (SPT) methods that are the most interesting in the transportation field.

Let $G=(N,A)$ is a simple directed graph, where N is the set of the nodes, of cardinality n, and A is the set of the arcs, of cardinality m, $c: A \rightarrow R$ be a function which assigns a *cost* c_{ij} to each $(i,j) \in A$.

Given a *root* $r \in N$, the SPT problem consists in finding a directed tree T such that the (only) path from r to i in T is one of the shortest paths from r to i in G, for each $i \in N$ which is connected to r, i.e. a directed path from r to i exists. It is proved in the literature [4], [11] that a finite solution exists if and only if there is no direct cycle of negative cost in G.

Let, $FS_{(i)} = \{(i,j) \in A\}$ and $BS_{(i)} = \{(j,i) \in A\}$ denote the *forward star* and the *backward star* of i, respectively, while $b_r = 1 - n$, and $b_i = 1$, $\forall\, i \neq r$. Then the SPT is formulated as follows:

$$\min \sum_{(i,j)\in A} c_{ij} x_{ij}$$

$$\text{St} \quad \sum_{(i,j)\in BS(i)} x_{ij} - \sum_{(i,j)\in FS(i)} x_{ij} = b_i, \quad \forall i \in N$$

$$x_{ij} \geq 0, \quad \forall (i,j) \in A,$$

and the dual SPT (DSPT) with cost dual variable π_i of i:

$$\text{Max } (1-n)\pi_r + \sum_{j\neq r} \pi_j$$

$$\text{St} \quad \pi_j - \pi_i \leq c_{ij}, \qquad \forall (i,j) \in A.$$

2.1 Primal algorithms

The "primal" [1] algorithms basically adapt the primal network simplex approach. At each step they maintain a spanning arborescence T of root r, and check if the cost labels C (either the costs of the paths outgoing from r, or an upper bound for these costs) are a dual feasible solution, i.e. meet Bellman conditions:

$$C_j \leq C_i + c_{ij}, \qquad \forall (i.j) \in A. \tag{1}$$

These algorithms maintain a set of candidate nodes Q, whose incident arcs might violate (1). At each step a candidate node i is selected from Q, and condition (1) is checked for each arc of the forward star of i; if some arc (i, j) violates the condition, the cost label C_j is updated and j is inserted into Q.

The main difference in the algorithms of this family is in the selection rule of the candidate nodes in Q. The most famous algorithms of this type are [11]: Bellman-Ford's Algorithm, Floyd-Warshall's Algorithm, Dijkstra's Algorithm. Bellman and Ford's algorithm finds the shortest paths from one node s (the source) to all the others. The key insight is the repeated evaluation of the functional equation $v_{[k]}$ (i.e. the current shortest length from the source to node k). Each iteration evaluates the functional equation for each $v_{[k]}$ (2) using results from the previous iteration. The algorithm stops if no result changes.

The functional equations of the length of the shortest path from the source s to node i:

$$v[s] = 0$$
$$v[s] = \min\{v[i] + c_{i,k} : (i,k) - exists\} \qquad (2)$$

Bellman-Ford Algorithm:

Step 0: Initialization: $v^0_{[k]} = 0$ if $k = s$; $+\infty$ otherwise

Iteration counter t ←1.

Step 1: Evaluation: For each node k evaluate

$$v_k^t \leftarrow \min\{v_i^{t-1} + c_{i,k} : (i,k) exist\}$$

if $v_k^t < v_k^{t-1}$, set d[k]← the number of a neighboring node i achieving the min v_k^t

Step 2: Stopping: Terminate if $v_k^t = v_k^{t-1}$ for $\forall$ k or if t =the number of nodes in the graph.

Step 3: Advance: If some *v[k]* changed at $t <$ the number of nodes, increment t ← t + 1 and return to *Step 1*.

Here d[k] denotes the node preceding k in the best known path from s to k. This algorithm finds the optimal paths of t or fewer steps. [11]

The algorithm of Floyd and Warshall searches for the paths from all nodes to all others in a graph without negative cycles [11]. The main feather is in the sequence of

route calculations. Initialization sets the $V_{k,l}^{0} = c_{k,l}$, because the only pat from k to l without intermediate nodes is the arc (k,l) with cost $c_{k,l}$. With the subsequent iterations we concatenate the previous results (subpaths). At the termination, all the possibilities of the minimization in functional equations have been checked, and the final *v[k,l]* are optimal.

Dijkstra's algorithm is considered more efficient for searching for shortest paths from one node to all others, and all costs are nonnegative ($c_{i,j} \geq 0$).

Dijkstra's Algorithm:

Step 0: Initialization: $v^{0}_{[i]} = 0$ if $i = s$ (source); $+\infty$ otherwise

Mark all nodes temporary and choose $p \leftarrow s$ as the next permanently labeled node.

Step 1: Processing: Mark node p permanent and for every arc/edge (p,i) from p to a temporary node, update

$$V_i \leftarrow \min\{V_i, V_p + C_{p,i}\}$$

if V_i changed in value, set d[i]$\leftarrow p$.

Step 2: Stopping: If no temporary nodes remain, stop; values V_i now reflect the required shortest path lengths.

Step 3: Next Permanent: Choose as next permanently labeled node p a temporary node with least current value $V_i : V_i = \min\{V_i : i - temporary\}$

Return to *Step 1*.

The new approach with the Dijkstra's algorithm is the way of calculation of the functional equations. Bellman-Ford algorithm evaluates functional equations (2) for all nodes on all iterations. Dijkstra's method process outbound arcs and edges instead of inbounds. At each iteration it makes one new node p permanent and thus it processes each arc/edge only once. Once a node is classified permanent its *v[p]*and *d[p]* labels never change again. So Dijkstra's algorithm is the most efficient available for computing shortest paths from one node to all others in a graph with all arc costs nonnegative.

It runs in $O(n^2)$ time in the case of nonnegative costs [4].

All these SPT algorithms select a candidate node with different strategies, are known as label-correcting or list-search algorithms. For implementation of the set Q they use FIFO queue and are also named L-queue.

Further development of these algorithms is in the list, which combines the properties of the data structures queue and stack – deque, where addition and deletion are possible at either end of the list [1]. Although its worst-case complexity is $O(n2^n)$, this type proved to be efficient on sparse an planar graphs.

Next development of the deque algorithms is in the partition of the candidate nodes into two subsets – Q' and Q'' according to a suitable threshold value. Q' contains only nodes with label (cost) $\leq$ threshold. When Q' is empty, the threshold is updated and all the nodes in Q'' with label $\leq$ threshold are moved into Q'. The aim is to increase the probability of selecting the minimum label node, by keeping the operational simplicity of the list-search strategies.

Bertsekas proposed an innovative algorithm based on L-threshold in 1993. It selects the candidate nodes trying to scan the nodes with a small label as early as possible. If the label of the candidate node to be inserted into Q is less than the one of the node at the head of Q, the candidate node is inserted in the head; otherwise, it is inserted in the end of the data structure. This method can be combined with the threshold approach and has shown good practical behavior. [1]

The topological ordering algorithm of Goldberg and Radzik (1993) tries to propagate the label updating of the nodes in Q by means of a topological visit of a proper sub graph of G. It also runs in $O(mn)$ time.

2.2 Dual Algorithms for SPT

The base of this algorithm family is the classical dual simplex algorithm, which maintains dual feasible basic solutions. Handler and Zang (1980) proposed a pioneering dual algorithm to solve SPP with the additional constraint that feasible paths must have a "weight" less than a given value λ. [1]

Recently dual ascent algorithms have been described to solve SPT. They consider the dual Lagrangean problem and try to maximize the dual objective function by updating

the price vector π in such a way as to always maintain the complementary slackness conditions. Only one price is changed at a time:

$$\pi_j = \min\{\pi_i + c_{ij} : (i,j) \in BS_{(j)}\} \quad \forall j \in N \setminus \{r\}.$$

If all the cycles in G have positive cost and the initial price vector π is dual feasible, i.e.

$$\pi_j - \pi_i \leq c_{ij}, \quad \forall (i,j) \in A, \tag{3}$$

then the algorithm produces the optimal price vector.

Bertsekas introduced the innovative Auction Approach for the SPT [8]. With this algorithm the shortest path is searched using the "primal-dual" approach: at each step a candidate path P outgoing from root r and a feasible dual vector π are maintained, which is dual feasible, i.e. meet (3), and satisfy the complementary slackness conditions:

$$(i,j) \in P \Rightarrow \pi_j = \pi_i + c_{ij}. \tag{4}$$

If P reaches the destination t, i.e. a primal feasible solution has been found, P is one of the shortest paths from r to t. Otherwise extension contraction, and deletion operations are performed at the end node of P.

Recently, a new family of dual ascent algorithms, the Hanging family, was proposed, which is strictly related to Relaxation method. These algorithms maintain a partial shortest path tree rooted at r, T = (N_T, A_T) and a feasible dual vector π, which satisfies (3) and (4). At the beginning, $N_T = \{r\}$ and A_T = Ø. To augment T, each $j \notin T$ called an out of tree node, looks for an entering arc (i, j) that satisfies (3), and $i \in N_T$. If such an arc is found, node j is "hanged" to T through (i, j), since the shortest path from r to j is found. To guarantee that at least one node can be hanged to T at each step, the algorithm makes a partition of the set of the arcs coming to each $j \notin N_T$ into the sets of arcs outgoing from tree nodes, called border arcs ($BS_{B(j)}$), and external arcs ($BS_{E(j)}$) and compute the minimum of both dual prices:

$$\beta_i = \min\{\pi_i + c_{ij} : (i,j) \in BS_{b(j)}\}$$
$$\eta_i = \min\{\pi_i + c_{ij} : (i,j) \in BS_{e(j)}\} \text{ and}$$
$$\gamma_i = \min\{\beta_i, \eta_i\}$$

If $\gamma_i = \beta_i$ for some node j, j can be hanged to T. If $\beta_i > \gamma_i$ for each $j \notin N_T$, no hanging is performed. To avoid this situation a bunch of global dual updating operations (classical, local, global, selected, extended, and double reprises) is introduced. All they maintain the feasibility of the price vector and ensure the entry of at least one new node into T. They are based on the global threshold gap of the prices δ:

$$\delta = \min\{\beta_i - \pi_i : j \notin N_T\}$$

δ is the minimum increment to be added to the prices of the out of the tree nodes to guarantee that at least one of them can be hanged to T.

2.3 Reoptimization approaches in SPT

One of the first efficient reoptimzation strategies in SP problem considers the situation where a shortest path tree relative to a given origin r, say T_r is determined and we have to find a new shortest path tree when either the origin is changed, or exactly one arc is given a new cost [1]. These two cases are of interest because of their strong practical application.

In the case when exactly one arc is given a new cost, the proposed algorithms reoptimize the shortest path tree after the cost change, and are basically extensions of Dijkstra's approach.

When the origin is changed to a node $s \neq r$, Gallo's procedure recognizes that the subtree of T_r rooted at s certainly belongs to the new shortest path tree rooted at s, say T_s, and then it computes T_s by means of Dijkstra's or Dial's approach, by using the (optimum) reduced costs relative to T_r instead of the original arc costs:

$$c^*_{ij} = c_{ij} + \pi^r_i - \pi^r_j, \quad \forall (i.j) \in A. \tag{5}$$

Here, c^*_{ij} 's are called the updated costs and due to the dual feasibility of the price vector of T_r, $c^*_{ij} \geq 0$, $\forall (i.j) \in A$, and hence shortest –first search procedures can also be applied for negative costs. We will use a similar approach later in one of the heuristics we developed (FTU).

Further development of SPT leads to Gallo&Pallotino algorithms [2], which maintains a cost label C^*i, $\forall\, i \in N$, relative to the updated costs (5). Starting from the new

root s, at each step the algorithm extends the current shortest path tree, say T^*, by inserting new nodes and new arcs until T^* spans onto G. The algorithm considers the arcs in the cutset separating T^* from the remaining nodes, and adds to T^* a cutest arc, say (u, v), such that u belongs to T^* and v has a minimum cost label C^*v. When (u, v) and v are added to T^*, all the nodes in the subtree of Tr rooted at v are reachable from v through arcs having a zero updated cost, i.e. all these nodes have the same minimum label as v, the entire subtree is moved to T^* by suitably implementing the set of the candidate nodes Q.

Although presented as a primal approach, Gallo&Pallotino's algorithm can be interpreted as a dual method. Starting from the partial shortest path tree T^*, it maintains the dual feasible solution at each step by operators based on reprise functions.

2.4 Time Dependent Shortest Paths

The dynamic SP problems consider the factor "time" and represent close to real problems that arise in transportation. A travel time or delay $d_{ij}(t)$ is associated with each arc (i, j), so that if t is the leaving time from i, then $t + d_{ij}(t)$ is the arrival time at node j. Also a time dependent cost $c_{ij}(t)$ is associated with (i,j), and a waiting cost $w_i(t)$ represents the possibility and the price of waiting at node i at time t.

Different models have been defined and analyzed in the literature depending on the properties of the delay functions (e.g. continuous or discrete), on the possibility of waiting at the nodes (e.g. no waiting, waiting at each node, waiting only at the root node), and depending also on the choice of the leaving time from the root node (in particular, dynamic shortest paths for a fixed leaving time or for all the possible leaving times).

An interesting discrete model is proposed by Orda and Rom (1990-91) [1]. They assume that the time variable t can vary in the discrete set $T = \{t1, t2, tq\}$, and each delay function $d_{ij}(t)$ is such that $t + d_{ij}(t) \in T$, which is the general case in transportation. Here they introduce the Space-Time Network R, defined as follows:

$$V = \{i_h: i \in N, 1 \le h \le q\};$$
$$E = \{(i_h, j_k): (i,j) \in A, t_h + d_{ij}(t_h) = t_k, 1 \le h < k \le q\},$$

and waiting arcs (i_h, i_{h+1}) which represent the waiting at i from time t_h to time t_{h+1}, and it is given the unit time cost $w_i(t_h)$, $h = 1, \ldots q$-1. So R is a standard graph, with $|V| = nq$ and

$|A| \leq (m+n)q$, with size is pseudopolynomial with respect to the size of the original graph G. It is proved that the Minimum Cost Dynamic Path Problem can be solved in O(|A|) time whatever the delay, cost and waiting functions.

The procedure selects the nodes of R in chronological order and uses a bucket-list $B = \{B_1, B_2, ..., B_q\}$ to perform the selecting operations (B_h denotes the bucket content of nodes to be visited at time instant t_h). The initial step is $Bp = \{r\}$ if the given departure time is $t = t_p$, while the other buckets are empty. The stop condition is verified when all the buckets are empty (when this happens, the minimum of the labels associated with each node $i \neq r$ gives the optimum path cost from r to i).

Procedure Chrono-SPT
* typical iteration *
select i **from** B_h; $B_h := B_h \setminus \{i\}$;
for each $(i,j) \in FS_{(i)}$ **do**
    begin
    $t_k := t_h + d_{ij(th)}$;
        if $C_{i(th)} + c_{ij(th)} < C_{j(tk)}$ **then**
            begin
            $C_{j(tk)} := C_{i(th)} + c_{ij(th)}$;
            $P_{j(tk)} := i_h$;
                if $j \notin B_k$ **then** $B_k = B_k \cup \{j\}$
            end
    end;
* if waiting at i at time t_h is allowed *
    if $C_{i(th)} + w_{i(th)}(t_{h+1} - t_h) < C_{i(th+1)}$ **then**
        begin
        $C_{i(th+1)} := C_{i(th)} + w_{i(th)(th+1 - th)}$;
        $P_{i(th+1)} := i_h$;
            if $i \notin B_{h+1}$ **then** $B_{h+1} = B_{h+1} \cup \{i\}$
        end;

It is also proved that Chrono-SPT runs in O(q + $|A^*|$) time, where A^* is the set of non-redundant arcs.

An interesting approach is derived considering the FIFO property [1]:
an arc (i,j) is said to be a *FIFO arc* if leaving i earlier guarantees that one will arrive no later at j along (i,j). Or:

$$t_h + d_{ij(th)} \leq t_k + d_{ij(tk)}, \text{ for any } t_h < t_k. \qquad (6)$$

A dynamic graph G is said to be a *FIFO graph* when all its arcs are FIFO.

A similar property can be imposed on the arc costs: an arc (i,j) is said to be a *Cost Consistent (CC) arc* if leaving i earlier along (i,j) does not cost more than leaving later, and G is a *CC graph* when all its arcs are CC arcs. Or:

$$t_u = t_h + d_{ij(th)} \text{ and } t_v = t_k + d_{ij(tk)}, \text{ for any } t_h < t_k.$$

If G is both FIFO and CC, i.e. when leaving earlier along any arc allows one to arrive no later and to pay no more than leaving later, then the concept of "dominated labels" can be introduced and exploited in the algorithmic model Chrono-SPT. Suppose we have visited two different paths from a given origin node r to a certain node i, and suppose that the two paths arrive at i at time th and at time tk, respectively, with $th < tk$. Assume also that the costs of the two paths, $C_{i(th)}$ and $C_{i(tk)}$ are such that $C_{i(th)} \leq C_{i(tk)}$. In this case, since the graph is both FIFO and CC, it is not convenient to extend the second path through any arc (i,j) outgoing from i, since this extension will not produce any minimum cost path going through node i: the cost of the second path, i.e. $C_{i(tk)}$, is a so-called *dominated label* for node i, and it can be ignored. Chrono-SPT can thus be simplified in such a way to only maintain non-dominated labels.

It is clear that the proposed Chrono-SPT algorithm is a very close practical approximation in case the network is serviced by a set of means of same transportation type. For example, it will be very useful in modeling subway or railway transportation systems, where the transportation time and cost keep constant *dominancy*. However, the case we consider is broader with the assumptions of parallel arcs, i.e. the IMC's could be visited by different transportation means with different container capacities at different time. Hence the net of IMC's is neither Cost Consistent nor FIFO graph.

2.5 Minimum Cost Flow Solvers

Based on the above-considered algorithms, a number of single-commodity Min Cost Flow (MCF) codes have been proposed [10]. Practically, among the most efficient ones are:

- Relax Solver – developed by Dimitri Bertsekas and Paul Tseng. Relax implements a primal dual algorithm, where at each iteration it tries to construct a feasible path of arcs with zero reduced cost (4) – from a node

with positive surplus to a node with negative surplus. The cost reoptimization is easily performed.

- CS2 solver – developed by Andrew Goldberg and Boris Cherkassky. This is also a primal dual approach, with the difference that a cost-scaling phase allows to operate on arcs with nonzero reduced cost. The cost reoptimization is also very easy.
- MCF ZIB solver – developed by Andreas Lobel. This solver is a specialized version of the simplex algorithm implemented directly on the network. The primal simplex is more suited to cost reoptimization than the dual simplex, because it easily exploits the previous optimal base.
- MCF Cplex solver Netopt – developed by ILOG Co. Netopt is based on primal and dual network simplex implementation. It's primal is very efficient and offers full reoptimization capabilities.

2.6 Intermodal Path Search

Most of the algorithms and researches in the intermodal path search consider urban transportation network and passenger problems.

Ziliaskopoulos and Wardell proposed a practically applicable algorithm for intermodal time path search with variety of transportation modes. [14] It is very efficient for calculating routings of intelligent transportation systems operating in dynamically changing conditions like urban transportation networks. "It computes the least-time paths from every origin node, mode and departure time to the destination node, considering all available modes of transportation and accounting for mode and arc switching delays". The algorithm begins at the destination node and in a label correcting fashion iteratively solves the optimality equation by "scanning" all nodes that have the potential of improving at least one label of another node. "Scanning" a node *i* means examining all of its predecessors and checking if extending the path from the current node to its predecessor nodes provides a better path from these nodes to the destination node for all modes and time intervals. If such an extension provides a better path for a predecessor node *j* for at least one mode, time interval and arc combination, node *j* is considered to have the potential to improve the paths to its predecessor nodes and marked as eligible to

be scanned. All nodes eligible to be scanned are kept in a list called "scan eligible" (SE) list, a structure used in static label correcting implementations [1].

In the beginning, the SE list only contains the destination node *N*. The labels of node *N* are set equal to the cost of switching from mode *x* to the exit mode x_f and exit node N^2 ; all other node labels are set to infinity. In the first iteration, all nodes that can directly reach *N* are updated and are inserted in the SE list. Next, the first node *j* of the SE list is scanned by updating every predecessor node *i* to node *j*. If at least one of the labels of L_i is modified, then node *i* is inserted in the SE list. This step is repeated until the SE list is empty and the algorithm terminates.

Although the travel times on the optimum paths among the origin-destination pairs are different for different departure times and modes, most of them are topologically the same. So, the algorithm simultaneously updates all paths that are topologically similar. It is proved that the computational complexity of this algorithm is independent of the number of modes and fixed schedule routes.

Ahuja and Orlin also investigated the dynamic shortest paths minimizing travel times and costs with main application in urban transportation environment [9]. They assume that the cost of an arc depends linearly on the minimum possible travel time and the excess time for the arc i.e. $c_{i,j}(t) = \alpha d^*_{i,j} + \beta e_{i,j}(t)$.

In a graph with FIFO property they show that:

1. The min excess time walk problem (the min cost walk problem) is NP – complete.
2. The min cost walk problem can be solved in O(βe*/min(α,β)) time if $d_{i,j}(t) \geq 0$, and if there are no directed cycles with 0 travel time.
3. The running time depends on the maximum excess time on any arc e*, rather than on the travel times $d_{ij}(t)$.

In the transit passenger applications studied by Marcotte and Nguen [7] has been considered the combination of path probabilities and the route strategy. In addition to the ordinary cost c_{ij} a traversal cost w_j is associated with each node, and so the hyper path problem. The proposed algorithm for hyper short path is based on Bellman's equation for residual cost from any node to the destination.

In this hyper short problem, the arc travel costs are associated with the mean waiting time at the stop node and the route strategy. However, the computations of all-shortest-paths problems, such as Floyd-Warshall algorithm and the reoptimization techniques developed by Galo and Pallotino [2], [5], Gabini[6] cannot be adapted to the all-shortest-hyper paths problems. Instead, specific reoptimization techniques have been proposed based on reduced costs with label setting approach [7].

In the case of capacity optimizations they require that a path cannot be utilized if a shorter unsaturated path is available. Than the optimum solution is obtained by adding a suitably chosen queuing delay α_{ij} to the cost of each saturated arc (i,j). The delay along a path is equal to the sum of delays of the arcs forming that path.

They assume that the users of the transportation network select a travel strategy at their start node. It is associated with an ordered set of successor nodes. At each node, the user selects the first available outgoing arc in this ordered set, where the probability that the arc is available is proportional to its capacity. Optimal strategies take into account that an arc could be unavailable and must propose an alternative one. Although the strategies are deterministic, their realization depends on arc availability that is stochastic.

The cost function is defined with two processes that are closely related – *loading process* and *pricing process*. The first one generates the access probabilities, the hyper path flows and the arc flows. The second provides the expected costs of best strategies.

3. Solution Techniques

Our goal is developing an easily computed algorithm with practical application for calculation of the system's optimal transportation cost of container transportation within the borders of network of IMC's. The constraints that must be satisfied are limited container capacity of the transportation means at each IMC P_{irk} and the due time of delivery T_i for execution of each transportation job.

We use the obtained data for rail intermodal rail transportation from last year independent study. The rail container transportation cost is \$0.03/ton/mile, loading/unloading operations = \$20/container. The fixed costs of train operation are assumed about \$10,000.

We search for optimum costs a set of examples within a fixed network of 10 IMC's by 6 transportation means under a fixed time schedule and capacities (fig.1). Then we calculate the same transportation jobs with three heuristics. Finally, we compare the obtained results.

Here are the assumptions we will follow:

1. The cheapest route is not necessary the shortest one.
2. A container can be loaded on a transportation mean if it arrives at the current IMC, at least half an hour before the departure time of the transportation mean.
3. With the heuristic models a new job can start after the previous is already completed.
4. A transportation mean is considered used (i.e. assigned value $y_r = 1$) even if it has transferred one container within one segment (between two IMC's).

3.1 Optimal Cost Solution with Cplex

First we solve a set of examples consisting of four jobs transferring from 82 to 112 containers within a fixed network of 10 IMC's by 6 transportation means under a fixed time schedule and capacities with Cplex. The transportation schedules with the container capacities and prices are prepared on an Excel table sheets (see *Attachment 1*). Job data are assigned on separate Excel sheets (see *Attachment 2*), one per each test.

In this case we use the developed C/C++ code (see *Attachment 3*) based on Cplex Callable Library [15]. It extracts the data from corresponding transportation tables and

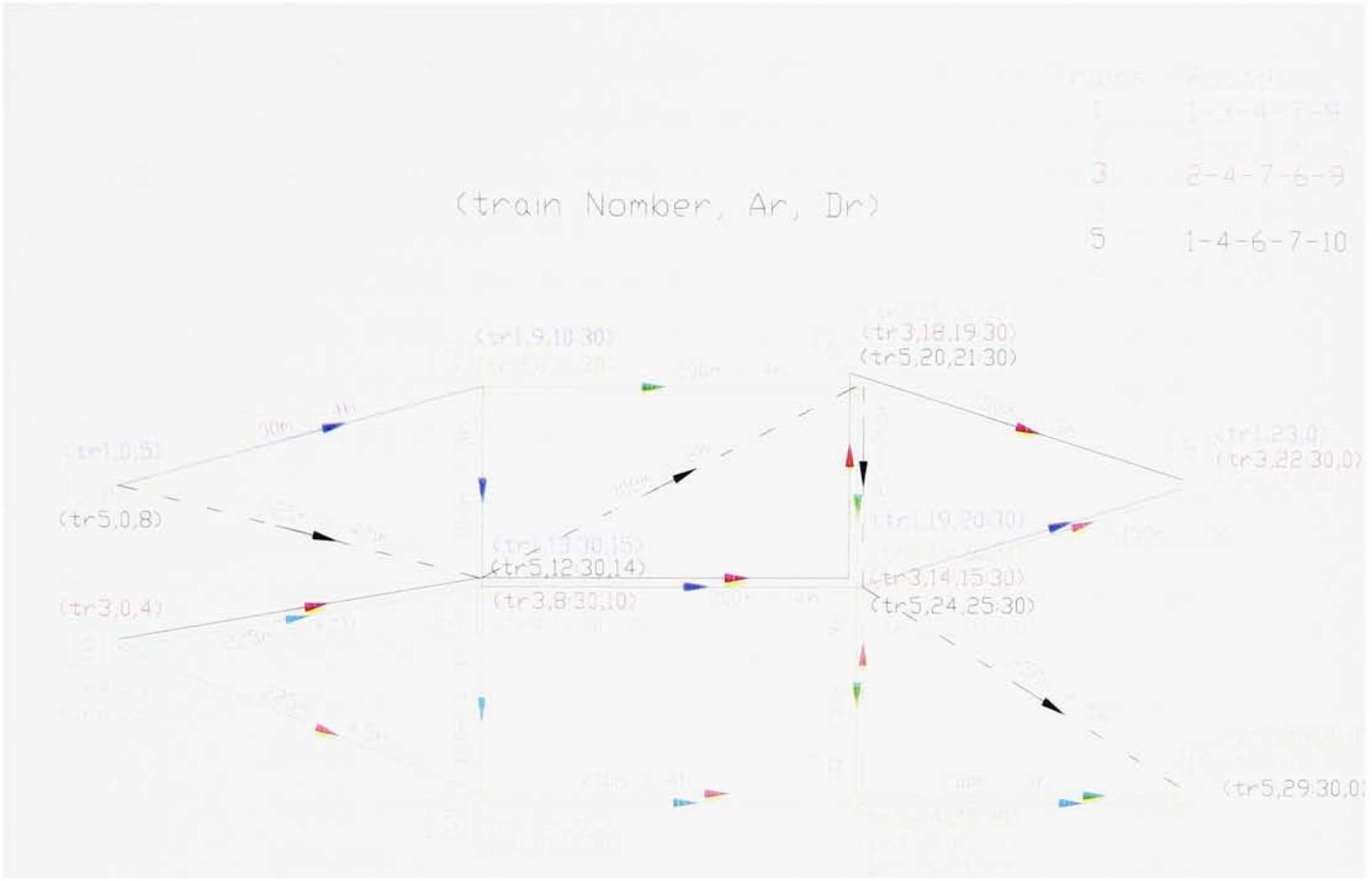
(train Nomber, Ar, Dr)
(tr5,0,8)
(tr3,0,4)
(tr5,12,30,14)
(tr3,8,30,10)
(tr3,18,19,30)
(tr5,20,21,30)
(tr3,14,15,30)
(tr5,24,25,30)
(tr3,22,30,0)
(tr5,29,30,0)

Fig. 1

job tables and provides the input files of all the ten experiments to Cplex. The optimal feasible solutions are calculated within 10-14 hours per experiment.

3.2 Heuristic Models

To avoid the long running time needed by Cplex, we create and compare the behavior of three heuristic models. In all of them, the insight is in the order of execution of the jobs. In the first two of them, we rank the transportation jobs according to their "weight". Then we use the developed C/C++ code (see *Attachment 4*) to provide all the feasible transportation routes that satisfy the time constraints. Next we apply Shortest Path Solver based on Dijkstra's algorithm to select the most cost effective routes among all time feasible ones by solving a sequence of shortest path problems for each job and do the entire job before the next in rank. The whole procedure takes up to 1.5 hours per experiment.

The common heuristic algorithm:

1. Calculate the "weight of each job" according to the heuristic rules and rank the jobs.
2. Start execution the job with the highest rank. If the number of containers of executed job is less or equal to each of capacities of segments of shortest path route calculated by the solver, execute the job end subtract the number of containers of the current job from the capacities of each used segment (IMC).
3. If the number of containers of current job is greater than some of capacities of the segments of the shortest path route, calculate the minimum possible capacity and transport up to this number of containers. Subtract the number of containers of the current job from the capacities of each used segment and cutoff the segment(s) with zero capacity from the network.
4. Solve the remaining part of the job with the solver under the updated network. If the system does not have resource for transportation of some of the containers (the solver returns infinity cost), imply penalty of double

cost of previous container transportation from the same job for each undelivered container on time.

5. Continue with the next job in rank until all the jobs are executed.
6. To calculate the total transportation cost sum all the cost of the jobs and the fixed costs of the used transportation means.

The formal presentation of the common algorithm of GDC and EDT heuristics is as follows:

Step 0: Calculate job weights w_i according to heuristic rules and rank w_1- w_4,
Set $o_i^* \leftarrow 0$, $i \leftarrow 1$
Step 1: Conduct SPP search for job(w_i) with Solver
If Solver returns no feasible route – apply penalty
Step 2: If $o_i \leq p_{irk}$ for every r,k , $p_{irk} \leftarrow p_{irk} - o_i$, go to Step 4
If $\exists\ p_{irk} = 0$, cutoff the link (k,k+1)/r, go to Step 4
Step 3: If $p_{irk} < o_i$, set $o_i^* \leftarrow o_i$, $o_i \leftarrow \min p_{irk}$,Go to Step 1
Step 4: Set $o_i^* \leftarrow o_i^* - o_i$, if $o_i^* > 0$ go to Step1,
Else $i \leftarrow i+1$, if $i \leq N$ go to Step 1

3.2.1 "Greedy Distance x Containers" (GDC) Calculate the "weight of each job" by multiplying the total transportation distance by number of containers of each job and rank the jobs according to their "weights".

3.2.2 "Earliest Due Time" (EDT) – the weight is assigned according to the due time – the first, the highest.

3.2.3 "Flexible Transportation Utilization" (FTU) the "weight of each job" is the same like of GDC, however we follow different algorithm. The idea is to utilize once used transportation mean as much as possible. Hence, reducing the number of the used means will allow us to eliminate some of the fixed cost(s).

Algorithm "FTU":

1. Calculate the "weight of each job" according to GDC heuristic and rank the jobs.
2. Start execution the job with the highest rank. If the number of containers of executed job is less or equal to each of capacities of the segments of the shortest path route calculated by the solver, execute the job end subtract the number of containers of the current job from the capacities of each used

segment (IMC). Null all the variable costs of the used transportation means over all the segments in the network.

3. If the number of containers of current job is greater than some of capacities of segments of shortest path route, calculate the minimum possible capacity and transport up to this number of containers. Subtract the number of containers of the current job from the capacities of each used segment and cutoff the segment(s) with zero capacity from the network. Null all the variable costs of the used transportation means over all the segments in the network.
4. Solve the remaining part of the job with the solver under the updated network. To calculate the final transportation cost, use the original variable costs of all the segments. If the system does not have resource for transportation of some of the containers (the solver returns infinity cost), imply penalty of double cost of previous container transportation from the same job for each undelivered container on time.
5. Continue with the next job in rank until all the jobs are executed.
6. To calculate the total transportation cost sum all the cost of the jobs and the fixed costs of the used transportation means.

The formal presentation of FTU algorithm is as follows:

Step 0: Calculate job weights w_i according to heuristic rules and rank w_1- w_4,
Set $o_i^* \leftarrow 0$, $i \leftarrow 1$

Step 1: Conduct SPP search for job(w_i) with Solver
If Solver returns no feasible route – apply penalty

Step 2: If $o_i \leq p_{irk}$ for every r,k , $p_{irk} \leftarrow p_{irk} - o_i$, set all $c_{ijrk,(k+1)} = 0$ of every link of every used mode *r*, go to Step 4
If $\exists\ p_{irk} = 0$, cutoff the link (k,k+1)/r, go to Step 4

Step 3: If $p_{irk} < o_i$, set $o_i^* \leftarrow o_i$, $o_i \leftarrow \min p_{irk}$, Go to Step 1

Step 4: Set $o_i^* \leftarrow o_i^* - o_i$, if $o_i^* > 0$ go to Step1,
Else $i \leftarrow i+1$, if $i \leq N$ go to Step 1

We will review the calculation procedures of Test 1. The files with the calculations of the remaining nine tests are loaded on the attached CD named "Opt Sol Files".

The optimal solution of the first test - Objective Function Value = $115,014 (see *Attachment 5*) is provided in 37855.02 sec (10.52h).

Then we conduct the tests with the three heuristics.

First we calculate the rank of the jobs for GDC and FTU heuristics (*Table 1*).

Test 1				
Job	[illegible]	Distance	[illegible]	Rank
1	19	890	16910	1
2	20	700	14000	4
3	21	710	14910	3
4	22	680	14960	2

Table 1.

Then we apply C-code for calculation of the time feasible routes using the data from tables "traindatabase Test1" and "jobdata Test1". Based on its results (see *Attachment 6*) we prepare the table of list of time feasible routes (*Table 2*). It allows us to prepare easily the original data for the Shortest Path Solver (see *Attachment 7*).

List of Time Feasible Routes Test #1

Num Job	Num route	Start IMC	Routes					Destination
1	1	1	4_5	4_1	7_1	7_2	8_2	10_2
1	2	1	4_5	4_4	5_4	8_4	8_2	10_2
1	3	1	4_5	4_4	5_4	8_4		10_4
1	4	1	3_1	4_1	7_1	8_2		10_2
1	5	1	3_1	3_2	6_2	7_2	8_2	10_2
2	1	1	4_5	4_1	7_1			9_1
2	2	1	3_1	3_2	6_2	6_3		9_3
2	3	1	3_1	4_1	7_1			9_1
2	4	1	3_1	3_2	6_2	7_2	7_1	9_1
3	1	2	4_3	7_3	6_3			9_3
4	1	2	5_6	5_4	8_4	8_2		10_2
4	2	2	5_6	8_6	8_2			10_2
4	3	2	5_6	5_4	8_4			10_4
4	4	2	5_6	8_6	8_4			10_4
4	5	2	4_3	4_1	7_1	8_2		10_2
4	6	2	4_4	4_1	7_1	8_2		10_2
4	7	2	4_3	7_3	7_2	8_2		10_2
4	8	2	4_3	4_4	5_4	8_4	8_2	10_2
4	9	2	4_4	5_4	8_4	8_2		10_2
4	10	2	4_3	4_4	5_4	8_4		10_4
4	11	2	4_4	5_4	8_4			10_4
4	12	2	5_6	8_6	7_6	8_2		10_2

Table 2

Having loaded the feasible graph and job data in a file, we start searching the shortest path calculations with GDC heuristic:

The job with the highest rank is No 1 – 19 containers from IMC 1 to IMC 10. The result from the solver is route

1 → 4/5 → 4/4 → 5/4 → 8/4 → 10 (node/by train) with unit cost \$890/container.

Since all of the capacities of the used trains are greater then the numbers of containers to be transferred i.e.19, the job is completed and the cost calculated 19x890 = \$16,910. After reducing the capacities of the used segments in "traindatabase Test1" table with 19, we proceed with the next job in rank, No 4 – 22 containers from IMC 2 to IMC 10.

The solver returns the route

2 → 5/6 → 5/4 → 8/4 → 10 with unit cost \$680/container.

Because the capacity of train 4 at IMC 8 is 5 we transfer exactly 5 containers on this route. The cost of this operation is 5 x 680 = \$3,400. Then we cut the link 8/4 → 10 off from the graph in the solver memory, update the capacities of the used links, and search again for the cheapest route. The result is route

2 → 5/6 → 8/6 → 8/2 → 10 with unit cost \$680/container.

All the capacities > than the remaining 17 containers of the job, so we perform the transfer with cost of 17 x 680 = \$11,560.

After reducing the capacities of the used links, we proceed with the next job in rank – No 3 – 21 containers from IMC 2 to IMC 9. The shortest path returned from the solver is

2 → 4/3 → 4/1 → 7/1 → 9 with unit cost \$710/container.

All the capacities of the used links > 21, so we perform the transfer with cost 21 x 710 = \$14,910. After updating the capacities, we search for the next cheapest path – job No 2 – 20 containers from IMC 1 to IMC 9.

The proposed solution from the solver is route

1 → 3/1 → 3/2 → 6/2 → 6/3→ 9 with unit cost \$700/container.

All the capacities of these links > 20, so we transfer all the 20 containers, and calculate the cost of the job 20 x 700 = \$14,000.

To perform all the four jobs we used all the six trains with total fixed costs = $61,250. So, the Total Cost of this test including all the four jobs performed in this order is 60,780 + 61,250 = $122,030.

To perform the test with the next heuristic EDT, we use its ranking of the jobs: 3 – 2 – 4 – 1. The procedure is the same like with the previous heuristic. In these calculations we encounter 2 cutoffs of used links and use all of the six trains. The Total Cost of all the jobs is 61,120 + 61,250 = $122,370.

The procedure of FTU heuristic deserves more attention. We use the same sequence like GDC heuristic 1 – 4 – 3 - 2. With the first job we have the same proposed route

1 → 4/5 → 4/4 → 5/4 → 8/4 → 10 with the same unit cost $890/container.

All the capacities of the used trains > 19, and we execute the job. Its cost is 19 x 890 = $16,910. The new insight is the assign of zero cost of all the links in the graph with the used trains i.e. for train 4: 2→4/4→5/4→8/4→10/4,

and train 5: 1→4/5→6/5→7/5→10/5.

After these temporary cost changes, the algorithm will give preference to the links operated by these already used trains. The idea is to utilize once used train as much as possible. With the calculation of the job costs we will use the original costs of all the used links.

The next in rank job is No 4 – 22 containers from IMC 2 to IMC 10. The cheapest calculated route is

2 → 4/4 → 5/4 → 8/4 → 10. The capacity of link 8/4 → 10 is 5, so we transfer 5 containers in this route. The cost is 5 x 870 = $4,350.

After cutting off link 8/4 → 10 and updating the capacities, we proceed with a new search for the remaining 17 containers. The respond is

2 → 4/4 → 5/4 → 8/4 →8/2 → 10. We perform transfer of 10 containers and reduce the container capacities. Link 4/4 → 5/4 is cutoff because of zero capacity and all the links of the newly used train 2 are made with zero cost. The cost of this sub-job is 10 x 890 = $ 8,900. Then we continue with path search for the remaining 7 containers. The respond of the solver is

2 → 4/4 → 4/1 → 7/1 → 7/2 →8/2 → 10. All the capacities > 7 and we perform the transfer. We update the capacities and make all the links of train 1 zero. The cost of this sub-job is 7 x 910 = $6,370.

Next job in rank is No 3 – 21 containers from IMC 2 to IMC 9. The proposed route is

2 → 4/3 → 7/3 → 6/3 → 9 with unit cost $834/container.

The capacities > 21 and we transfer all the containers. The cost of the job is 21 x 834 = $17,514. After reducing the capacities of the used links and making the cost of all links of train 3 zero, we proceed with last job – No 4: 20 containers from IMC 1 to IMC 9. The calculated path is

1 → 3/1 → 4/1 → 7/1 → 9 with unit cost $840/container. All the capacities of the links > 20, so we perform the job. The cost is 20 x 840 = $16,800.

With this procedure we used trains 1, 2, 3, 4, and 5, with sum of fixed costs = $50,750. So, the Total Cost of all the jobs of test 1 executed in this away is 70,844 + 50,750 = $121,594.

4. Solution Results

The solution results of all ten tests are summarized in the following *Table 3*:

		HEURISTICS					
		GDC		EDT		FTU	
TEST #	Optimal Cost	Routes	Total Cost	Routes	Total Cost	Routes	Total Cost
1	**$115,014**	1-4-3-2	122,030	3-2-4-1	122,370	1-4-3-2	121,594
% Deviation from Optimum			6.10%		6.40%		5.72%
2	**$125,338**	1-3-4-2	130,420	3-2-4-1	143,180	1-3-4-2	126,380
% Deviation from Optimum			4.05%		14.24%		0.83%
3	**$113,998**	4-3-2-1	133,300	3-2-4-1	138,880	4-3-2-1	136,988
% Deviation from Optimum			16.93%		21.83%		20.17%
4	**$104,530**	3-2-1-4	122,640	3-2-4-1	124,700	3-2-1-4	126,128
% Deviation from Optimum			17.33%		19.30%		20.66%
5	**$114,560**	2-3-1-4	124,230	3-2-4-1	128,360	2-3-1-4	119,644
% Deviation from Optimum			8.44%		12.05%		4.44%
6	**$113,570**	4-1-3-2	123,570	3-2-4-1	130,346	4-1-3-2	113,570
% Deviation from Optimum			8.81%		14.77%		0.00%
7	**$118,388**	1-4-3-2	131,580	3-2-1-4	131,580	1-4-3-2	130,472
% Deviation from Optimum			11.14%		11.14%		10.21%
8	**$137,947**	1-3-4-2	145,255	3-2-1-4	145,015	1-3-4-2	144,605
% Deviation from Optimum			5.30%		5.12%		4.83%
9	**$135,165**	4-2-1-3	143,045	3-2-1-4	142,405	4-2-1-3	169,495
% Deviation from Optimum			5.83%		5.36%		25.40%
10	**$138,113**	2-3-4-1	156,138	3-2-1-4	146,570	2-3-4-1	139,507
% Deviation from Optimum			13.05%		6.12%		1.01%

Table 3

5. Analysis of the results

5.1 Statistical analysis.

To compare the results from *Table 3*, first we conduct a two-way analysis of variance with Minitab. The experiment is as follows:

H_0 hypothesis: No difference among the 4 types of calculations,

or

H_0 hypothesis: No difference among the 10 transportation tests

versus the alternative

H_a hypothesis: At least one differs from at least one other.

Two-way ANOVA: TC versus Heuristic, test

```
Analysis of Variance for TC
Source        DF        SS        MS        F        P
Heuristic      3     1.141E+09 380476651     8.94    0.000
test           9     4.314E+09 479363491    11.26    0.000
Error         27     1.150E+09  42582359
Total         39     6.605E+09

                          Individual 95% CI
Heuristic        Mean     -----+---------+---------+---------+------
1              121661     (------*------)
2              133221                    (------*------)
3              135341                        (------*------)
4              132838                   (------*------)
                          -----+---------+---------+---------+------
                          120000    126000    132000    138000

                          Individual 95% CI
test             Mean     --------+---------+---------+---------+---
 1             120252       (-----*------)
 2             131330                 (-----*------)
 3             130789                (------*-----)
 4             119500      (-----*------)
 5             121699        (------*-----)
 6             120264       (-----*------)
 7             128005              (------*------)
 8             143206                          (-----*------)
 9             147528                              (------*-----)
10             145082                           (------*------)
                          --------+---------+---------+---------+---
                             120000    130000    140000    150000
```

The ANOVA results show high F-value and low P-value that indicate that the null hypothesis must be rejected. With p-value $< \alpha = 0.05$ there is enough evidence that at least one of them differs from at least one from the others.

To evaluate exactly which one differs from which of the rest, we conduct paired T-tests .

Paired T-Test and CI: OPT, EDT

```
Paired T for OPT - EDT

                 N      Mean     StDev   SE Mean
OPT             10    121661     11808      3734
EDT             10    135341      8900      2814
Difference      10    -13679      6213      1965

95% upper bound for mean difference: -10078
T-Test of mean difference = 0 (vs < 0): T-Value = -6.96  P-Value = 0.000
```

Paired T-Test and CI: OPT, GDC

```
Paired T for OPT - GDC

                 N      Mean     StDev   SE Mean
OPT             10    121661     11808      3734
GDC             10    133221     11490      3634
Difference      10    -11559      5246      1659

95% upper bound for mean difference: -8518
T-Test of mean difference = 0 (vs < 0): T-Value = -6.97  P-Value = 0.000
```

Paired T-Test and CI: OPT, FTU

```
Paired T for OPT - FTU

                 N      Mean     StDev   SE Mean
OPT             10    121661     11808      3734
FTU             10    132838     16014      5064
Difference      10    -11177     11486      3632

95% upper bound for mean difference: -4519
T-Test of mean difference = 0 (vs < 0): T-Value = -3.08  P-Value = 0.007
```

Paired T-Test and CI: GDC, FTU

```
Paired T for GDC - FTU

                 N      Mean     StDev   SE Mean
GDC             10    133221     11490      3634
FTU             10    132838     16014      5064
Difference      10       383     11253      3559

95% upper bound for mean difference: 6906
T-Test of mean difference = 0 (vs < 0): T-Value = 0.11  P-Value = 0.542
```

Paired T-Test and CI: EDT, FTU

```
Paired T for EDT   FTU

                N       Mean      StDev    SE Mean
EDT            10     135341       8900        2814
FTU            10     132838      16014        5064
Difference     10       2502      12331        3899

95% upper bound for mean difference: 9650
T-Test of mean difference = 0 (vs < 0): T-Value = 0.64  P-Value = 0.731
```

Paired T-Test and CI: EDT, GDC

```
Paired T for EDT - GDC

                N       Mean      StDev    SE Mean
EDT            10     135341       8900        2814
GDC            10     133221      11490        3634
Difference     10       2120       5853        1851

95% upper bound for mean difference: 5513
T-Test of mean difference = 0 (vs < 0): T-Value = 1.15  P-Value = 0.859
```

The results show that there is significant difference in the Optimal calculations compared to the other three Heuristics calculations, and there is no evidence for significant difference among the three heuristics.

5.2 Engineering analysis.

Although statistically equivalent, the results produced by the three heuristics are different from engineering point of view. To perform all the jobs of the 10 tests GDC and EDT heuristics utilize all the six trains in each test (*Table 4*).

Number of Used Transportation Means

Test	Optimal	GDC	EDT	FTU
1	5	6	6	5
2	5	6	6	5
3	4	6	6	6
4	4	6	6	6
5	5	6	6	5
6	5	6	6	5
7	4	6	6	5
8	5	6	6	5
9	5	6	6	6
10	5	6	6	5
Total trains	47	60	60	53
Average	4.7	6	6	5.3

Table 4

With these experiments the results of FTU are 12% better – it utilizes average 5.3 trains. The advantages of this approach are clear:

- Higher utilization of the transportation means
- Lower train maintenance costs and overhaul expenses
- Smaller personal and crew number
- Reduced probabilities for breakdowns

To compare the transportation mode utilization, we conduct paired T-test with data from *Table 4*. Again, the null hypothesis is that there is no difference in the train utilization, and the alternative is that they are not equal with 95% confidence intervals (CI).

Paired T-Test and CI: GDC, FTU

```
Paired T for GDC - FTU

                N      Mean     StDev   SE Mean
GDC            10     6.000     0.000     0.000
FTU            10     5.300     0.483     0.153
Difference     10     0.700     0.483     0.153

95% CI for mean difference: (0.354, 1.046)
T-Test of mean difference = 0 (vs not = 0): T-Value = 4.58  P-Value = 0.001
```

Paired T-Test and CI: GDC, OPT

```
Paired T for GDC - OPT

                N      Mean     StDev   SE Mean
GDC            10     6.000     0.000     0.000
OPT            10     4.700     0.483     0.153
Difference     10     1.300     0.483     0.153

95% CI for mean difference: (0.954, 1.646)
T-Test of mean difference = 0 (vs not = 0): T-Value = 8.51  P-Value = 0.000
```

Paired T-Test and CI: FTU, OPT

```
Paired T for FTU   OPT

                N      Mean     StDev   SE Mean
FTU            10     5.300     0.483     0.153
OPT            10     4.700     0.483     0.153
Difference     10     0.600     0.843     0.267

95% CI for mean difference: (-0.003, 1.203)
T-Test of mean difference = 0 (vs not = 0): T-Value = 2.25  P-Value = 0.051
```

GDC and EDT heuristics have the same utilization numbers. The results of the experiment show that heuristics GDC and EDT are different to FTU and OPT, and that FTU heuristic statistically provides the same train utilization like OPT.

6. Conclusions

The results from the conducted transportation tests show that in 70% of the cases the difference between the Total Costs calculated by FTU heuristic and the Optimal one is less than 10%. In addition, the transportation utilization with FTU is nearly 90% of the optimal. The main advantage is the shorter calculation time. With these 10 tests of four jobs the required time for calculation of FTU is in average 1/10 of the time required for calculation of Optimal Cost.

Therefore, in our opinion the proposed FTU algorithm can be used as a good approximation with practical application in the calculation of total transportation cost in a net of IMC's, where the costs of transportation have a dominant fixed part.

7. Proposals for future research

The proposed deterministic model and heuristic algorithms are good enough for practical calculation of optimum freight transportation cost in a system of IMC's. In case of leak of information for the container loads, we would suggest stochastic approach to be applied. In this case every transportation mean will serve as an independent agent that will have to bid to get any job. Trying to be competitive and profitable, it will have to optimize the offered costs.

The deterministic and stochastic approaches could be combined, so that the deterministic one calculates the loads and costs of the agent by some level of utilization that guarantees the use of the agent, recall 60% of load ability. The stochastic procedure will calculate the biding prices for the remaining 40% that will improve the profitability and compatibility of the agent.

Bibliography

[1] Pallottino S., M. Scutella (1997) *Shortest Path Algorithms in Transportation models: classical and innovative aspects.*

[2] Pallottino S., M. Scutella (2001) *A new algorithm for reoptimizing shortest paths when the arc cost change.*

[3] Jones W.B., C.R.Cassady, R.O.Bowden (2000), *Developing a standard definition of intermodal transportation.*

[4] Bixby R.E. (1981-82) *Combinatorial Optimization*, Northwestern University notes

[5] Chabini I., Glenn, A., Pallottino, S., M.Scutella (2001), *Reoptimization Algorithms for Minimum-Time Path Problems in Dynamic Networks.*

[6] Galo G., S. Pallottino, M.Scutella, A. Frangioni, F. Farinaccio, *Transportation and logistics: Models and Algorithms.*

[7] Marcotte P., S. Nguyen,(1995) Hyperpath Formulations of Traffic Assignment Problem

[8] De Leone R., D. Pretolani, (1998) *Auction Algorithms for Shortest Hyperpath Problems.*

[9] Ahuja R., J. Orlin, S. Pallottino, M.Scutella, (2001) *Dynamic Shortest Paths Minimizing Travel Times and Costs.*

[10] Frangioni A., A. Manca, (2001) *A Computational Study of Cost Reoptimization for Min Cost Flow Problems*

[11] Rardin R, *Optimization in Operations Research,* Prentice-Hall, 2000

[12] Askin R., J. Goldberg, *Design and Analysis of Lean Production Systems,* John Wiley & Sons, 2002

[13] Mendenhall W., R. Beaver, M. Beaver, *Introduction to Probability and Statistics- 11th edition,* Brooks/Coole-Thomson Learning, 2003

[14] Ziliaskopoulos A., W. Wardell, Design and Implementation of an Intermodal Optimum Path Algorithm for Multimodal Networks with Dynamic Arc Travel Times and Switching Delays (1997) http://trans.civil.nwu.edu/~thanasis/itdsp/itdsp.html

[15] Ilog Cplex Manual, 2002

Table of Contents

Attachments 1+ 2 /traindata and job data/

Test 1

24

train # (r)	start ITC	Depart (Dr)	end ITC	Arrive (Ar)	cont.cap.(p)	tr.var.cost	tr.fixed cost	tr.change cost
1	1	5.00	3	9.00	21	240	10000	20
1	3	10.30	4	13.30	41	180	10000	20
1	4	15.00	7	19.00	31	240	10000	20
1	7	20.30	9	23.00	51	180	10000	20
2	3	11.30	6	15.30	42	240	10500	20
2	6	17.00	7	19.30	32	144	10500	20
2	7	21.00	8	24.00	52	180	10500	20
2	8	25.30	10	28.30	62	180	10500	20
3	2	4.00	4	8.30	33	270	10000	20
3	4	10.00	7	14.00	43	240	10000	20
3	7	15.30	6	18.00	53	144	10000	20
3	6	19.30	9	22.30	43	180	10000	20
4	2	7.00	4	11.30	44	270	10000	20
4	4	13.00	5	16.00	34	180	10000	20
4	5	17.30	8	21.30	54	240	10000	20
4	8	23.00	10	26.00	24	180	10000	20
5	1	8.00	4	12.30	35	270	10250	20
5	4	14.00	6	20.00	45	360	10250	20
5	6	21.30	7	24.00	25	144	10250	20
5	7	25.30	10	29.30	55	300	10250	20
6	2	6.00	5	10.00	26	240	10500	20
6	5	11.30	8	15.30	46	240	10500	20
6	8	17.00	7	20.00	36	180	10500	20
6	7	21.30	9	24.30	46	180	10500	20

job #	start ITC	end ITC	# cont	start time	deadline	jobdist
1	1	10	19	4.00	29.00	890
2	1	9	20	4.00	23.45	700
3	2	9	21	4.00	22.45	710
4	2	10	22	4.00	29.00	680
		total # cont	**82**			

TEST #2

train # (r)	start ITC	Depart (Dr)	end ITC	Arrive (Ar)	cont.cap.(p)	tr.var.cost	tr.fixed cost	tr.change cost
1	1	5.00	3	9.00	41	240	10000	20
1	3	10.30	4	13.30	31	180	10000	20
1	4	15.00	7	19.00	21	240	10000	20
1	7	20.30	9	23.00	51	180	10000	20
2	3	11.30	6	15.30	22	240	10500	20
2	6	17.00	7	19.30	32	144	10500	20
2	7	21.00	8	24.00	52	180	10500	20
2	8	25.30	10	28.30	42	180	10500	20
3	2	4.00	4	8.30	23	270	10000	20
3	4	10.00	7	14.00	43	240	10000	20
3	7	15.30	6	18.00	53	144	10000	20
3	6	19.30	9	22.30	33	180	10000	20
4	2	7.00	4	11.30	44	270	10000	20
4	4	13.00	5	16.00	54	180	10000	20
4	5	17.30	8	21.30	34	240	10000	20
4	8	23.00	10	26.00	24	180	10000	20
5	1	8.00	4	12.30	35	270	10250	20
5	4	14.00	6	20.00	45	360	10250	20
5	6	21.30	7	24.00	25	144	10250	20
5	7	25.30	10	29.30	55	300	10250	20
6	2	6.00	5	10.00	26	240	10500	20
6	5	11.30	8	15.30	46	240	10500	20
6	8	17.00	7	20.00	36	180	10500	20
6	7	21.30	9	24.30	46	180	10500	20

job #	start ITC	end ITC	# cont	start time	deadline
1	1	10	22	4.00	29.00
2	1	9	23	4.00	23.45
3	2	9	24	4.00	23.15
4	2	10	25	4.00	29.00
		total # cont	**94**		

TEST #3

train # (r)	start ITC	Depart (Dr)	end ITC	Arrive (Ar)	cont.cap.(p)	tr.var.cost	tr.fixed cost	tr.change cost
1	1	5.00	3	9.00	41	240	10000	20
1	3	10.30	4	13.30	31	180	10000	20
1	4	15.00	7	19.00	21	240	10000	20
1	7	20.30	9	23.00	51	180	10000	20
2	3	11.30	6	15.30	22	240	10500	20
2	6	17.00	7	19.30	32	144	10500	20
2	7	21.00	8	24.00	52	180	10500	20
2	8	25.30	10	28.30	42	180	10500	20
3	2	4.00	4	8.30	23	270	10000	20
3	4	10.00	7	14.00	43	240	10000	20
3	7	15.30	6	18.00	53	144	10000	20
3	6	19.30	9	22.30	33	180	10000	20
4	2	7.00	4	11.30	44	270	10000	20
4	4	13.00	5	16.00	54	180	10000	20
4	5	17.30	8	21.30	34	240	10000	20
4	8	23.00	10	26.00	24	180	10000	20
5	1	8.00	4	12.30	35	270	10250	20
5	4	14.00	6	20.00	45	360	10250	20
5	6	21.30	7	24.00	25	144	10250	20
5	7	25.30	10	29.30	55	300	10250	20
6	2	6.00	5	10.00	26	240	10500	20
6	5	11.30	8	15.30	46	240	10500	20
6	8	17.00	7	20.00	36	180	10500	20
6	7	21.30	9	24.30	46	180	10500	20

job #	start ITC	end ITC	# cont	start time	deadline
1	1	10	15	4.00	29.00
2	1	9	20	4.00	23.45
3	2	9	25	4.00	23.15
4	2	10	30	4.00	29.00
		total # cont	**90**		

TEST #4

train # (r)	start ITC	Depart (Dr)	end ITC	Arrive (Ar)	cont.cap.(p)	tr.var.cost	tr.fixed cost	tr.change cost
1	1	5.00	3	9.00	41	240	10000	20
1	3	10.30	4	13.30	31	180	10000	20
1	4	15.00	7	19.00	21	240	10000	20
1	7	20.30	9	23.00	51	180	10000	20
2	3	11.30	6	15.30	22	240	10500	20
2	6	17.00	7	19.30	32	144	10500	20
2	7	21.00	8	24.00	52	180	10500	20
2	8	25.30	10	28.30	42	180	10500	20
3	2	4.00	4	8.30	23	270	10000	20
3	4	10.00	7	14.00	43	240	10000	20
3	7	15.30	6	18.00	53	144	10000	20
3	6	19.30	9	22.30	33	180	10000	20
4	2	7.00	4	11.30	44	270	10000	20
4	4	13.00	5	16.00	54	180	10000	20
4	5	17.30	8	21.30	34	240	10000	20
4	8	23.00	10	26.00	24	180	10000	20
5	1	8.00	4	12.30	35	270	10250	20
5	4	14.00	6	20.00	45	360	10250	20
5	6	21.30	7	24.00	25	144	10250	20
5	7	25.30	10	29.30	55	300	10250	20
6	2	6.00	5	10.00	26	240	10500	20
6	5	11.30	8	15.30	46	240	10500	20
6	8	17.00	7	20.00	36	180	10500	20
6	7	21.30	9	24.30	46	180	10500	20

job #	start ITC	end ITC	# cont	start time	deadline
1	1	10	18	4.00	29.00
2	1	9	23	4.00	23.45
3	2	9	23	4.00	23.15
4	2	10	18	4.00	29.00
		total # cont	**82**		

TEST #5

train # (r)	start ITC	Depart (Dr)	end ITC	Arrive (Ar)	cont.cap.(p)	tr.var.cost	tr.fixed cost	tr.change cost
1	1	5.00	3	9.00	41	240	10000	20
1	3	10.30	4	13.30	31	180	10000	20
1	4	15.00	7	19.00	21	240	10000	20
1	7	20.30	9	23.00	51	180	10000	20
2	3	11.30	6	15.30	22	240	10500	20
2	6	17.00	7	19.30	32	144	10500	20
2	7	21.00	8	24.00	52	180	10500	20
2	8	25.30	10	28.30	42	180	10500	20
3	2	4.00	4	8.30	23	270	10000	20
3	4	10.00	7	14.00	43	240	10000	20
3	7	15.30	6	18.00	53	144	10000	20
3	6	19.30	9	22.30	33	180	10000	20
4	2	7.00	4	11.30	44	270	10000	20
4	4	13.00	5	16.00	54	180	10000	20
4	5	17.30	8	21.30	34	240	10000	20
4	8	23.00	10	26.00	24	180	10000	20
5	1	8.00	4	12.30	35	270	10250	20
5	4	14.00	6	20.00	45	360	10250	20
5	6	21.30	7	24.00	25	144	10250	20
5	7	25.30	10	29.30	55	300	10250	20
6	2	6.00	5	10.00	26	240	10500	20
6	5	11.30	8	15.30	46	240	10500	20
6	8	17.00	7	20.00	36	180	10500	20
6	7	21.30	9	24.30	46	180	10500	20

job #	start ITC	end ITC	# cont	start time	deadline
1	1	10	18	4.00	29.00
2	1	9	25	4.00	23.45
3	2	9	24	4.00	23.15
4	2	10	18	4.00	29.00
		total # cont	**85**		

TEST #6

train # (r)	start ITC	Depart (Dr)	end ITC	Arrive (Ar)	cont.cap.(p)	tr.var.cost	tr.fixed cost	tr.change cost
1	1	5.00	3	9.00	41	240	10000	20
1	3	10.30	4	13.30	31	180	10000	20
1	4	15.00	7	19.00	21	240	10000	20
1	7	20.30	9	23.00	51	180	10000	20
2	3	11.30	6	15.30	22	240	10500	20
2	6	17.00	7	19.30	32	144	10500	20
2	7	21.00	8	24.00	52	180	10500	20
2	8	25.30	10	28.30	42	180	10500	20
3	2	4.00	4	8.30	23	270	10000	20
3	4	10.00	7	14.00	43	240	10000	20
3	7	15.30	6	18.00	53	144	10000	20
3	6	19.30	9	22.30	33	180	10000	20
4	2	7.00	4	11.30	44	270	10000	20
4	4	13.00	5	16.00	54	180	10000	20
4	5	17.30	8	21.30	34	240	10000	20
4	8	23.00	10	26.00	24	180	10000	20
5	1	8.00	4	12.30	35	270	10250	20
5	4	14.00	6	20.00	45	360	10250	20
5	6	21.30	7	24.00	25	144	10250	20
5	7	25.30	10	29.30	55	300	10250	20
6	2	6.00	5	10.00	26	240	10500	20
6	5	11.30	8	15.30	46	240	10500	20
6	8	17.00	7	20.00	36	180	10500	20
6	7	21.30	9	24.30	46	180	10500	20

job #	start ITC	end ITC	# cont	start time	deadline
1	1	10	18	4.00	29.00
2	1	9	20	4.00	23.45
3	2	9	22	4.00	23.15
4	2	10	24	4.00	29.00
		total # cont	**84**		

TEST #7

train # (r)	start ITC	Depart (Dr)	end ITC	Arrive (Ar)	cont.cap.(p)	tr.var.cost	tr.fixed cost	tr.change cost
1	1	5.00	3	9.00	30	250	10000	20
1	3	10.30	4	13.30	40	170	10000	20
1	4	15.00	7	19.00	35	250	10000	20
1	7	20.30	9	23.00	25	170	10000	20
2	3	11.30	6	15.30	32	230	10500	20
2	6	17.00	7	19.30	22	144	10500	20
2	7	21.00	8	24.00	42	190	10500	20
2	8	25.30	10	28.30	31	190	10500	20
3	2	4.00	4	8.30	33	280	10000	20
3	4	10.00	7	14.00	23	245	10000	20
3	7	15.30	6	18.00	33	144	10000	20
3	6	19.30	9	22.30	43	170	10000	20
4	2	7.00	4	11.30	34	260	10000	20
4	4	13.00	5	16.00	38	190	10000	20
4	5	17.30	8	21.30	24	235	10000	20
4	8	23.00	10	26.00	38	190	10000	20
5	1	8.00	4	12.30	45	260	10250	20
5	4	14.00	6	20.00	25	370	10250	20
5	6	21.30	7	24.00	30	154	10250	20
5	7	25.30	10	29.30	25	290	10250	20
6	2	6.00	5	10.00	23	250	10500	20
6	5	11.30	8	15.30	33	250	10500	20
6	8	17.00	7	20.00	26	170	10500	20
6	7	21.30	9	24.30	36	170	10500	20

job #	start ITC	end ITC	# cont	start time	deadline
1	1	10	22	4.00	29.00
2	1	9	23	4.00	25.00
3	2	9	24	4.00	24.00
4	2	10	25	4.00	30.00
		total # cont	**94**		

TEST # 8

train # (r)	start ITC	Depart (Dr)	end ITC	Arrive (Ar)	cont.cap.(p)	tr.var.cost	tr.fixed cost	tr.change cost
1	1	5.00	3	9.00	30	250	10000	20
1	3	10.30	4	13.30	40	170	10000	20
1	4	15.00	7	19.00	35	250	10000	20
1	7	20.30	9	23.00	25	170	10000	20
2	3	11.30	6	15.30	32	230	10500	20
2	6	17.00	7	19.30	22	144	10500	20
2	7	21.00	8	24.00	42	190	10500	20
2	8	25.30	10	28.30	31	190	10500	20
3	2	4.00	4	8.30	33	280	10000	20
3	4	10.00	7	14.00	23	245	10000	20
3	7	15.30	6	18.00	33	144	10000	20
3	6	19.30	9	22.30	43	170	10000	20
4	2	7.00	4	11.30	34	260	10000	20
4	4	13.00	5	16.00	38	190	10000	20
4	5	17.30	8	21.30	24	235	10000	20
4	8	23.00	10	26.00	38	190	10000	20
5	1	8.00	4	12.30	45	260	10250	20
5	4	14.00	6	20.00	25	370	10250	20
5	6	21.30	7	24.00	30	154	10250	20
5	7	25.30	10	29.30	25	290	10250	20
6	2	6.00	5	10.00	23	250	10500	20
6	5	11.30	8	15.30	33	250	10500	20
6	8	17.00	7	20.00	26	170	10500	20
6	7	21.30	9	24.30	36	170	10500	20

job #	start ITC	end ITC	# cont	start time	deadline
1	1	10	30	4.00	29.00
2	1	9	25	4.00	25.00
3	2	9	30	4.00	24.00
4	2	10	25	4.00	30.00
		total # cont	**110**		

TEST # 9

train # (r)	start ITC	Depart (Dr)	end ITC	Arrive (Ar)	cont.cap.(p)	tr.var.cost	tr.fixed cost	tr.change cost
1	1	5.00	3	9.00	30	250	10000	20
1	3	10.30	4	13.30	40	170	10000	20
1	4	15.00	7	19.00	35	250	10000	20
1	7	20.30	9	23.00	25	170	10000	20
2	3	11.30	6	15.30	32	230	10500	20
2	6	17.00	7	19.30	22	144	10500	20
2	7	21.00	8	24.00	42	190	10500	20
2	8	25.30	10	28.30	31	190	10500	20
3	2	4.00	4	8.30	33	280	10000	20
3	4	10.00	7	14.00	23	245	10000	20
3	7	15.30	6	18.00	33	144	10000	20
3	6	19.30	9	22.30	43	170	10000	20
4	2	7.00	4	11.30	34	260	10000	20
4	4	13.00	5	16.00	38	190	10000	20
4	5	17.30	8	21.30	24	235	10000	20
4	8	23.00	10	26.00	38	190	10000	20
5	1	8.00	4	12.30	45	260	10250	20
5	4	14.00	6	20.00	25	370	10250	20
5	6	21.30	7	24.00	30	154	10250	20
5	7	25.30	10	29.30	25	290	10250	20
6	2	6.00	5	10.00	23	250	10500	20
6	5	11.30	8	15.30	33	250	10500	20
6	8	17.00	7	20.00	26	170	10500	20
6	7	21.30	9	24.30	36	170	10500	20

job #	start ITC	end ITC	# cont	start time	deadline
1	1	10	24	4.00	29.00
2	1	9	30	4.00	25.00
3	2	9	24	4.00	24.00
4	2	10	30	4.00	30.00
		total # cont	**108**		

TEST # 10

train # (r)	start ITC	Depart (Dr)	end ITC	Arrive (Ar)	cont.cap.(p)	tr.var.cost	tr.fixed cost	tr.change cost
1	1	5.00	3	9.00	30	250	10000	20
1	3	10.30	4	13.30	40	170	10000	20
1	4	15.00	7	19.00	35	250	10000	20
1	7	20.30	9	23.00	25	170	10000	20
2	3	11.30	6	15.30	32	230	10500	20
2	6	17.00	7	19.30	22	144	10500	20
2	7	21.00	8	24.00	42	190	10500	20
2	8	25.30	10	28.30	31	190	10500	20
3	2	4.00	4	8.30	33	280	10000	20
3	4	10.00	7	14.00	23	245	10000	20
3	7	15.30	6	18.00	33	144	10000	20
3	6	19.30	9	22.30	43	170	10000	20
4	2	7.00	4	11.30	34	260	10000	20
4	4	13.00	5	16.00	38	190	10000	20
4	5	17.30	8	21.30	24	235	10000	20
4	8	23.00	10	26.00	38	190	10000	20
5	1	8.00	4	12.30	45	260	10250	20
5	4	14.00	6	20.00	25	370	10250	20
5	6	21.30	7	24.00	30	154	10250	20
5	7	25.30	10	29.30	25	290	10250	20
6	2	6.00	5	10.00	23	250	10500	20
6	5	11.30	8	15.30	33	250	10500	20
6	8	17.00	7	20.00	26	170	10500	20
6	7	21.30	9	24.30	36	170	10500	20

job #	start ITC	end ITC	# cont	start time	deadline
1	1	10	22	4.00	29.00
2	1	9	32	4.00	25.00
3	2	9	30	4.00	24.00
4	2	10	28	4.00	30.00
		total # cont	**112**		

```
// Attachment 3
#include <stdio.h>
#include <conio.h>
#include <windows.h>
#define MAXJOBS 5

struct JobTable(
    int numJob;
    int jobnum[MAXJOBS];
    int startITC[5];
    int endITC[5];
    int numCont[5];
    double startt[5];
    double deadlinet[5];
};

struct TrainTable(
    int numSegments;
    int trainnum[25];
    int tstITC[25];
    double departt[25];
    int tendITC[25];
    double arrivet[25];
    int tconcap[25];
    double tvarcost[25];
    double tfixcost[25];
    double tchgecost[25];
};

void objfunc(struct JobTable *,struct TrainTable *);
void getJobTable(struct JobTable *);
void getTrainTable(struct TrainTable *);
void constraintl( struct JobTable *,struct TrainTable *, FILE *, int);
void constraint2(struct JobTable *,struct TrainTable *,FILE *, int);
void boundaries( struct JobTable *,struct TrainTable *, FILE *, int);

int main()
{
    struct JobTable jtable;
    struct TrainTable ttable;

    getJobTable(&jtable);
    getTrainTable(&ttable);

    objfunc(&jtable,&ttable);
    return 0;
}

void getJobTable(struct JobTable *pt)
{
    FILE * jobdatabase;
    int counter = 0;
    jobdatabase = fopen("jobdata.txt", "r");

    fscanf(jobdatabase,"%d",  &pt->numJob);
    int numRows = pt->numJob;

    while (numRows >= 1)
    (
    fscanf(jobdatabase,"%d", &pt->jobnum[counter]);
    fscanf(jobdatabase,"%d", &pt->startITC[counter]);
    fscanf(jobdatabase,"%d", &pt->endITC[counter]);
    fscanf(jobdatabase,"%d", &pt->numCont[counter]);
    fscanf(jobdatabase,"%lf",&pt->startt[counter]);
    fscanf(jobdatabase,"%lf",&pt->deadlinet[counter]);
    numRows--;
```

```
    counter ++;
    }

    for (int t = 0; t < pt->numJob; t++)
        printf("\n%d    %d  %d  %d  %4.2lf  %4.2lf",pt->jobnum[t], pt->startITC[t],pt->
    endITC[t],pt->numCont[t],pt->startt[t],pt->deadlinet[t]);
    return;
}


void getTrainTable(struct TrainTable *pt){
FILE * traindatabase;
    int numRows = 0;
    int counter = 0;

traindatabase = fopen("traindata.txt", "r");
while (traindatabase == NULL)
{
    printf("\nIt didn't open the file");
    return;
}
fscanf(traindatabase, "%d", &pt->numSegments);

int totRow = pt->numSegments;
while (totRow >= 1) {
    fscanf(traindatabase, "%d", &pt->trainnum[counter]);
    fscanf(traindatabase, "%d", &pt->tstITC[counter]);
    fscanf(traindatabase, "%lf", &pt->departt[counter]);
    fscanf(traindatabase, "%d", &pt->tendITC[counter]);
    fscanf(traindatabase, "%lf", &pt->arrivet[counter]);
    fscanf(traindatabase, "%d", &pt->tconcap[counter]);
    fscanf(traindatabase, "%lf", &pt->tvarcost[counter]);
    fscanf(traindatabase, "%lf", &pt->tfixcost[counter]);
    fscanf(traindatabase, "%lf", &pt->tchgecost[counter]);
    totRow--;
    counter ++;
    }
for (int t = 0; t < pt->numSegments; t++)
    printf("\n%d    %d  %4.2lf  %d  %4.2lf  %d  %4.2lf  %4.2lf  %4.2lf",pt->trainnum[t],
    pt->tstITC[t], pt->departt[t],pt->tendITC[t],pt->arrivet[t],pt->tconcap[t],pt->
    tvarcost[t],pt->tfixcost[t],pt->tchgecost[t]);

}


void objfunc(struct JobTable *pt, struct TrainTable *ptr) {

FILE * objffile;
    char cFileName[35];
    int numRows = 0;
    int jn;
    int counter = 0;

    sprintf(cFileName, "cInput_AJtl0.lp");
    objffile = fopen(cFileName, "w");
    fprintf(objffile, "\nMIN");

for (int x = 1; x <= pt->numJob; x++)
{
    for (int t = 1; t <= pt->numCont[x-1]; t++)
    {
        for (int r = 0; r < ptr->numSegments; r++)
        {
        fprintf(objffile, "\n%8.2lfX%d_%d_%d_%d_%d +",ptr->tvarcost[r], x,t,ptr->trainnum
    [r],ptr->tstITC[r],ptr->tendITC[r]);
        fprintf(objffile, "\t%6.2lfZ%d_%d_%d_%d +", ptr->tchgecost[r], x, t,ptr->trainnum
```

```
    [r],ptr->tendITC[r]);
        //fprintf(objffile, "\nt = [%d]    r = [%d]",t, r);
        }
    }
}
     counter=1;
    for (int tr = 1; tr <= 24; tr+=4)
    {
        if (tr < 21)    fprintf(objffile,"\n\n%10.2lfY%d +", ptr->tfixcost[tr-1],
    counter);

        else    fprintf(objffile,"\n\n%10.2lfY%d\n\n st", ptr->tfixcost[tr-1],counter );

        counter++;
    }

for ( x = 1; x <= pt->numJob; x++)
    {
    constraint1(pt, ptr, objffile, x);

    int q,p,o,s;

    switch (x) {
    case 1 :    {for (q =1;q <= pt->numCont[x-1];q++)
                {
                    fprintf(objffile, "\n\nX%d_%d_3_2_4 = 0\nX%d_%d_4_2_4 = 0\nX%d_%
    d_1_7_9 = 0\nX%d_%d_3_6_9 = 0\nX%d_%d_6_2_5 = 0\nX%d_%d_6_7_9 = 0\nZ%d_%d_1_9 = 0\nZ%
    d_%d_2_10 = 0\nZ%d_%d_3_9 = 0\nZ%d_%d_4_10 = 0\nZ%d_%d_5_10 = 0\nZ%d_%d_6_9 = 0",x,q,
    x,q,x,q,x,q,x,q,x,q,x,q,x,q,x,q,x,q);
                    //fprintf(objffile, "\nX1_%d_3_2_4 = 0\nX1_%d_4_2_4 = 0\nX1_%
    d_1_7_9 = 0\nX1_%d_3_6_9 = 0\nZ1_%d_1_9 = 0\nZ1_%d_2_10 = 0\nZ1_%d_3_9 = 0\nZ1_%
    d_4_10 = 0\nZ1_%d_5_10 = 0",q,q,q,q,q,q,q,q,q);
                }
                break;}
    case 2 :    {for (p =1;p <= pt->numCont[x-1];p++)
                {
                    fprintf(objffile, "\n\nX%d_%d_3_2_4 = 0\nX%d_%d_4_2_4 = 0\nX%d_%
    d_6_2_5 = 0\nX%d_%d_2_8_10 = 0\nX%d_%d_4_8_10 = 0\nX%d_%d_5_7_10 = 0\nZ%d_%d_1_9 = 0\
    nZ%d_%d_2_10 = 0\nZ%d_%d_3_9 = 0\nZ%d_%d_4_10 = 0\nZ%d_%d_5_10 = 0\nZ%d_%d_6_9 = 0",x
    ,p,x,p,x,p,x,p,x,p,x,p,x,p,x,p,x,p,x,p,x,p);
                    //fprintf(objffile, "\nX2_%d_3_2_4 = 0\nX2_%d_4_2_4 = 0\nX2_%
    d_2_8_10 = 0\nX2_%d_4_8_10 = 0\nX2_%d_5_7_10 = 0\nZ2_%d_1_9 = 0\nZ2_%d_2_10 = 0\nZ2_%
    d_3_9 = 0\nZ2_%d_4_10 = 0\nZ2_%d_5_10 = 0",p,p,p,p,p,p,p,p,p,p);
                }
                break;}
    case 3 :    {for (s =1;s <= pt->numCont[x-1];s++)
                {
                    fprintf(objffile, "\n\nX%d_%d_1_1_3 = 0\nX%d_%d_5_1_4 = 0\nX%d_%
    d_2_8_10 = 0\nX%d_%d_4_8_10 = 0\nX%d_%d_5_7_10 = 0\nZ%d_%d_1_9 = 0\nZ%d_%d_2_10 = 0\
    nZ%d_%d_3_9 = 0\nZ%d_%d_4_10 = 0\nZ%d_%d_5_10 = 0\nZ%d_%d_6_9 = 0",x,s,x,s,x,s,x,s,x,
    s,x,s,x,s,x,s,x,s,x,s,x,s);
                    //fprintf(objffile, "\nX3_%d_1_1_3 = 0\nX3_%d_5_1_4 = 0\nX3_%
    d_2_8_10 = 0\nX3_%d_4_8_10 = 0\nX3_%d_5_7_10 = 0\nZ3_%d_1_9 = 0\nZ3_%d_2_10 = 0\nZ3_%
    d_3_9 = 0\nZ3_%d_4_10 = 0\nZ3_%d_5_10 = 0",s,s,s,s,s,s,s,s,s,s);
                }
                break;}
    case 4 :    {for (o =1;o <= pt->numCont[x-1];o++)
                {
                    fprintf(objffile, "\n\nX%d_%d_1_1_3 = 0\nX%d_%d_5_1_4 = 0\nX%d_%
    d_1_7_9 = 0\nX%d_%d_3_6_9 = 0\nX%d_%d_6_7_9 = 0\nZ%d_%d_1_9 = 0\nZ%d_%d_2_10 = 0\nZ%
    d_%d_3_9 = 0\nZ%d_%d_4_10 = 0\nZ%d_%d_5_10 = 0\nZ%d_%d_6_9 = 0",x,o,x,o,x,o,x,o,x,o,x
    ,o,x,o,x,o,x,o,x,o,x,o);
                    //fprintf(objffile, "\nX4_%d_1_1_3 = 0\nX4_%d_5_1_4 = 0\nX4_%d_1_7_9
     = 0\nX4_%d_3_6_9 = 0\nZ4_%d_1_9 = 0\nZ4_%d_2_10 = 0\nZ4_%d_3_9 = 0\nZ4_%d_4_10 = 0\n
     Z4_%d_5_10 = 0",o,o,o,o,o,o,o,o,o);
                }
```

```
                    break;}
    default:    printf ("\n Error - not admissable job in Job Table"); jn = 0;
            }

    }

constraint2 (pt,ptr, objffile, x);

boundaries (pt, ptr, objffile, x);
}


void constraint1( struct JobTable *pt,struct TrainTable *ptr, FILE *constrFile, int
    jobNum)
{
    //printf("\nTEST1 %d", jobNum);
    //printf("TEST2 %d", pt->numCont[jobNum-1]);
    for (int a = 1; a <= pt->numCont[jobNum-1]; a++ )
    {

        fprintf(constrFile,"\n\nX%d_%d_1_1_3 + X%d_%d_5_1_4 + X%d_%d_3_2_4 + X%d_%d_4_2_4
     + X%d_%d_6_2_5 = 1",jobNum,a, jobNum,a,jobNum, a, jobNum, a,jobNum,a);
        fprintf(constrFile,"\nX%d_%d_1_1_3 - X%d_%d_1_3_4 - X%d_%d_2_3_6 = 0",jobNum,a,
    jobNum,a,jobNum,a);
        fprintf(constrFile,"\nX%d_%d_1_3_4 + X%d_%d_5_1_4 + X%d_%d_3_2_4 + X%d_%d_4_2_4 -
     X%d_%d_1_4_7 - X%d_%d_3_4_7 - X%d_%d_4_4_5 - X%d_%d_5_4_6 = 0",jobNum,a,jobNum,a,
    jobNum,a,jobNum,a,jobNum,a,jobNum,a,jobNum,a);
        fprintf(constrFile,"\nX%d_%d_4_4_5 + X%d_%d_6_2_5 - X%d_%d_4_5_8 - X%d_%d_6_5_8 =
     0", jobNum,a,jobNum,a,jobNum,a,jobNum,a);
        fprintf(constrFile,"\nX%d_%d_2_3_6 + X%d_%d_5_4_6 + X%d_%d_3_7_6 - X%d_%d_2_6_7 -
     X%d_%d_3_6_9 - X%d_%d_5_6_7 = 0",jobNum,a,jobNum,a,jobNum,a,jobNum,a,jobNum,a,jobNum
    ,a);
        fprintf(constrFile,"\nX%d_%d_1_4_7 + X%d_%d_2_6_7 + X%d_%d_3_4_7 + X%d_%d_5_6_7 +
     X%d_%d_6_8_7 - X%d_%d_3_7_6 - X%d_%d_1_7_9 - X%d_%d_2_7_8 - X%d_%d_5_7_10 - X%d_%
    d_6_7_9 = 0",jobNum,a,jobNum,a,jobNum,a,jobNum,a,jobNum,a,jobNum,a,jobNum,a,jobNum,a,
    jobNum,a,jobNum,a);
        fprintf(constrFile,"\nX%d_%d_2_7_8 + X%d_%d_4_5_8 + X%d_%d_6_5_8 - X%d_%d_2_8_10
    - X%d_%d_4_8_10 - X%d_%d_6_8_7 = 0", jobNum,a,jobNum, a, jobNum, a,jobNum,a,jobNum,a,
    jobNum,a);
        fprintf(constrFile,"\nX%d_%d_1_7_9 + X%d_%d_2_8_10 + X%d_%d_3_6_9 + X%d_%d_4_8_10
     + X%d_%d_5_7_10 + X%d_%d_6_7_9 = 1", jobNum,a,jobNum, a, jobNum, a,jobNum,a,jobNum,a
    ,jobNum,a);

        fprintf(constrFile,"\n%4.2lfX%d_%d_1_1_3 - %4.2lfX%d_%d_1_3_4 - %4.2lfX%d_%
    d_2_3_6 < 0", ptr->arrivet[0],jobNum,a, ptr->departt[1],jobNum,a, ptr->departt[4],
    jobNum,a);
        fprintf(constrFile,"\n%4.2lfX%d_%d_1_3_4 + %4.2lfX%d_%d_5_1_4 + %4.2lfX%d_%
    d_3_2_4 + %4.2lfX%d_%d_4_2_4 - %4.2lfX%d_%d_1_4_7 - %4.2lfX%d_%d_3_4_7 - %4.2lfX%d_%
    d_4_4_5 - %4.2lfX%d_%d_5_4_6< 0", ptr->arrivet[1],jobNum,a,ptr->arrivet[16] ,jobNum,a
    ,ptr->arrivet[8] ,jobNum,a,ptr->arrivet[12] ,jobNum,a, ptr->departt[2],jobNum,a, ptr
    ->departt[9],jobNum,a,ptr->departt [13],jobNum,a,ptr->departt[17] ,jobNum,a);
        fprintf(constrFile,"\n%4.2lfX%d_%d_4_4_5 + %4.2lfX%d_%d_6_2_5 - %4.2lfX%d_%
    d_4_5_8 -%4.2lfX%d_%d_6_5_8 < 0", ptr->arrivet[13] ,jobNum,a,ptr->arrivet[20], jobNum
    ,a,ptr->departt[14] ,jobNum,a, ptr->departt[21], jobNum,a);
        fprintf(constrFile,"\n%4.2lfX%d_%d_2_3_6 + %4.2lfX%d_%d_3_7_6 + %4.2lfX%d_%
    d_5_4_6 - %4.2lfX%d_%d_2_6_7 - %4.2lfX%d_%d_5_6_7 - %4.2lfX%d_%d_3_6_9 < 0",ptr->
    arrivet[4] ,jobNum,a,ptr->arrivet[10] ,jobNum,a,ptr->arrivet[17] ,jobNum,a, ptr->
    departt[5],jobNum,a,ptr->departt[18] ,jobNum,a,ptr->departt[11] ,jobNum,a);
        fprintf(constrFile,"\n%4.2lfX%d_%d_1_4_7 + %4.2lfX%d_%d_2_6_7 + %4.2lfX%d_%
    d_3_4_7 + %4.2lfX%d_%d_5_6_7 + %4.2lfX%d_%d_6_8_7 - %4.2lfX%d_%d_1_7_9 - %4.2lfX%d_%
    d_2_7_8 - %4.2lfX%d_%d_3_7_6 - %4.2lfX%d_%d_5_7_10 -%4.2lfX%d_%d_6_7_9 < 0",ptr->
    arrivet[2] ,jobNum,a,ptr->arrivet[5] ,jobNum,a, ptr->arrivet[9],jobNum,a, ptr->
    arrivet[18],jobNum,a,ptr->arrivet[22],jobNum,a,ptr->departt[3],jobNum,a,ptr->departt
    [6] ,jobNum,a,ptr->departt[10] ,jobNum,a,ptr->departt[19] ,jobNum,a,ptr->departt[23],
    jobNum,a);
        fprintf(constrFile,"\n%4.2lfX%d_%d_4_5_8 + %4.2lfX%d_%d_2_7_8 + %4.2lfX%d_%
```

```
d_6_5_8 - %4.2lfX%d_%d_2_8_10 - %4.2lfX%d_%d_4_8_10 - %4.2lfX%d_%d_6_8_7 < 0", ptr-> ↙
arrivet[14],jobNum,a,ptr->arrivet[6],jobNum,a,ptr->arrivet[21],jobNum,a,ptr->departt ↙
[7] ,jobNum,a,ptr->departt[15] ,jobNum,a,ptr->departt[22],jobNum,a);
}


for ( a = 1; a <= pt->numCont[jobNum-1]; a++ )
    {
    fprintf(constrFile,"\n\nX%d_%d_1_1_3 + X%d_%d_2_3_6 - Z%d_%d_1_3 < 1", jobNum,a, ↙
jobNum, a, jobNum, a);
    fprintf(constrFile,"\nX%d_%d_1_3_4 + X%d_%d_5_4_6 + X%d_%d_3_4_7 + X%d_%d_4_4_5 -↙
 Z%d_%d_1_4 < 1", jobNum,a,jobNum, a,jobNum, a,jobNum,a,jobNum, a);
    fprintf(constrFile,"\nX%d_%d_1_4_7 + X%d_%d_2_7_8 + X%d_%d_3_7_6 + X%d_%d_5_7_10 ↙
+ X%d_%d_6_7_9 - Z%d_%d_1_7 < 1", jobNum,a,jobNum, a,jobNum, a,jobNum,a,jobNum,a,    ↙
jobNum,a);
    fprintf(constrFile,"\nX%d_%d_2_3_6 + X%d_%d_5_6_7 + X%d_%d_3_6_9 - Z%d_%d_2_6 < 1↙
", jobNum,a,jobNum, a,jobNum, a,jobNum,a);
    fprintf(constrFile,"\nX%d_%d_2_6_7 + X%d_%d_5_7_10 + X%d_%d_1_7_9 + X%d_%d_3_7_6 ↙
+ X%d_%d_6_7_9 - Z%d_%d_2_7 < 1", jobNum,a,jobNum, a,jobNum,a,jobNum,a,jobNum,a,    ↙
jobNum,a);
    fprintf(constrFile,"\nX%d_%d_2_7_8 + X%d_%d_4_8_10 + X%d_%d_6_8_7 - Z%d_%d_2_8 < ↙
1", jobNum,a,jobNum, a,jobNum, a,jobNum,a);
    fprintf(constrFile,"\nX%d_%d_3_2_4 + X%d_%d_1_4_7 + X%d_%d_4_4_5 + X%d_%d_5_4_6 -↙
 Z%d_%d_3_4 < 1", jobNum,a,jobNum, a,jobNum,a,jobNum,a,jobNum, a);
    fprintf(constrFile,"\nX%d_%d_3_4_7 + X%d_%d_1_7_9 + X%d_%d_2_7_8 + X%d_%d_5_7_10 ↙
+ X%d_%d_6_7_9 - Z%d_%d_3_7 < 1", jobNum,a,jobNum, a,jobNum,a,jobNum,a,jobNum,a,    ↙
jobNum,a);
    fprintf(constrFile,"\nX%d_%d_3_7_6 + X%d_%d_2_6_7 + X%d_%d_5_6_7 - Z%d_%d_3_6 < 1↙
", jobNum,a,jobNum, a,jobNum, a,jobNum,a);
    fprintf(constrFile,"\nX%d_%d_4_2_4 + X%d_%d_1_4_7 + X%d_%d_3_4_7 + X%d_%d_5_4_6 -↙
 Z%d_%d_4_4 < 1", jobNum,a,jobNum, a,jobNum,a,jobNum,a,jobNum, a);
    fprintf(constrFile,"\nX%d_%d_4_4_5 + X%d_%d_6_5_8 - Z%d_%d_4_5 < 1", jobNum,a,   ↙
jobNum, a,jobNum,a);
    fprintf(constrFile,"\nX%d_%d_4_5_8 + X%d_%d_2_8_10 + X%d_%d_6_8_7 - Z%d_%d_4_8 < ↙
1", jobNum,a,jobNum, a,jobNum,a,jobNum,a);
    fprintf(constrFile,"\nX%d_%d_5_1_4 + X%d_%d_1_4_7 + X%d_%d_4_4_5 + X%d_%d_3_4_7 -↙
 Z%d_%d_5_4 < 1", jobNum,a,jobNum, a,jobNum,a,jobNum,a,jobNum, a);
    fprintf(constrFile,"\nX%d_%d_5_4_6 + X%d_%d_2_6_7 + X%d_%d_3_6_9 - Z%d_%d_5_6 < 1↙
", jobNum,a,jobNum, a,jobNum, a,jobNum,a);
    fprintf(constrFile,"\nX%d_%d_5_6_7 + X%d_%d_1_7_9 + X%d_%d_2_7_8 + X%d_%d_3_7_6 +↙
 X%d_%d_6_7_9 - Z%d_%d_5_7 < 1", jobNum,a,jobNum, a,jobNum,a,jobNum,a,jobNum,a,jobNum↙
,a);
    fprintf(constrFile,"\nX%d_%d_6_2_5 + X%d_%d_4_5_8 - Z%d_%d_6_5 < 1", jobNum,a,   ↙
jobNum, a,jobNum,a);
    fprintf(constrFile,"\nX%d_%d_6_5_8 + X%d_%d_4_8_10 + X%d_%d_2_8_10 - Z%d_%d_6_8 <↙
 1", jobNum,a,jobNum, a,jobNum,a,jobNum,a);
    fprintf(constrFile,"\nX%d_%d_6_8_7 + X%d_%d_1_7_9 + X%d_%d_2_7_8 + X%d_%d_3_7_6 +↙
 X%d_%d_5_7_10 - Z%d_%d_6_7 < 1",jobNum,a,jobNum,a,jobNum,a,jobNum,a,jobNum,a,jobNum,↙
a);
    }

fprintf(constrFile,"\n\n");

for ( a = 1; a <= pt->numCont[jobNum-1]; a++ )
    {
fprintf(constrFile,"\n%8.2lfX%d_%d_1_7_9 < %8.2lf ", ptr->arrivet[3],jobNum,a,pt->   ↙
deadlinet[jobNum-1]);
fprintf(constrFile,"\n%8.2lfX%d_%d_2_8_10 < %8.2lf ", ptr->arrivet[7],jobNum,a,pt->  ↙
deadlinet[jobNum-1]);
fprintf(constrFile,"\n%8.2lfX%d_%d_3_6_9 < %8.2lf ", ptr->arrivet[11],jobNum,a,pt->  ↙
deadlinet[jobNum-1]);
fprintf(constrFile,"\n%8.2lfX%d_%d_4_8_10 < %8.2lf ", ptr->arrivet[15],jobNum,a,pt-> ↙
deadlinet[jobNum-1]);
fprintf(constrFile,"\n%8.2lfX%d_%d_5_7_10 < %8.2lf ", ptr->arrivet[19],jobNum,a,pt-> ↙
deadlinet[jobNum-1]);
fprintf(constrFile,"\n%8.2lfX%d_%d_6_7_9 < %8.2lf ", ptr->arrivet[23],jobNum,a,pt->  ↙
```

```
    deadlinet[jobNum-1]);
        }

    /*  fprintf(constrFile,"\n");
    for (int b = 1; b <= pt->numJob; b++)
        {
        if (b <= (pt->numJob)-1) {
            for ( a = 1; a <= pt->numCont[b-1]; a++ )
                fprintf(constrFile,"\tX%d_%d_1_4_7 + ",b,a);
                        }
        else {
            for ( a = 1; a <= pt->numCont[b-1]-1; a++ )
                fprintf(constrFile,"\tX%d_%d_1_4_7 + ",b,a);

                if (a <= pt->numCont[b-1]-2)
                    fprintf(constrFile,"\tX%d_%d_1_4_7 + ",b,a);
                else
                    fprintf(constrFile,"\tX%d_%d_1_4_7 < %d",b,a,ptr->tconcap[2]);
                }
        } */

}

void constraint2(struct JobTable *pt,struct TrainTable *ptr, FILE *constr2File, int b)
{
    int k = pt->numJob;

for ( b=1; b<=pt->numJob;b++)
    {
        for ( int a = 1; a <= pt->numCont[b-1]; a++ )
        {
            if ((b <= k) & (a <= pt->numCont[k-1]-1)) fprintf(constr2File,"\nX%d_%d_1_1_3↙
     + X%d_%d_1_3_4 + X%d_%d_1_4_7 + X%d_%d_1_7_9 +", b,a,b,a,b,a,b,a);

            else fprintf(constr2File,"\nX%d_%d_1_1_3 + X%d_%d_1_3_4 + X%d_%d_1_4_7 + X%d_↙
    %d_1_7_9 - %8.2lfY1 < 0", b,a,b, a, b, a,b,a,ptr->tfixcost[0]);
        }
    }

    for (b=1; b<=pt->numJob;b++)
    {
        for ( int a = 1; a <= pt->numCont[b-1]; a++ )
        {
        if ((b <= k) & (a <= pt->numCont[k-1]-1))   fprintf(constr2File,"\nX%d_%d_2_3_6 +↙
     X%d_%d_2_6_7 + X%d_%d_2_7_8 + X%d_%d_2_8_10 +", b,a,b,a,b,a,b,a);

        else fprintf(constr2File,"\nX%d_%d_2_3_6 + X%d_%d_2_6_7 + X%d_%d_2_7_8 + X%d_%   ↙
    d_2_8_10 - %8.2lfY2 < 0", b,a,b,a,b,a,b,a,ptr->tfixcost[4]);

        }
    }

    for (b=1; b<=pt->numJob;b++)
    {
        for ( int a = 1; a <= pt->numCont[b-1]; a++ )
        {
        if ((b <= k) & (a <= pt->numCont[k-1]-1)) fprintf(constr2File,"\nX%d_%d_3_2_4 + X↙
    %d_%d_3_4_7 + X%d_%d_3_7_6 + X%d_%d_3_6_9 +", b,a,b,a,b,a,b,a);

        else fprintf(constr2File,"\nX%d_%d_3_2_4 + X%d_%d_3_4_7 + X%d_%d_3_7_6 + X%d_%   ↙
    d_3_6_9 - %8.2lfY3 < 0", b,a,b, a,b, a,b,a,ptr->tfixcost[8] );
        }
    }

    for (b=1; b<=pt->numJob;b++)
    {
```

```
        for ( int a = 1; a <= pt->numCont[b-1]; a++ )
        {
        if ((b <= k) & (a <= pt->numCont[k-1]-1))   fprintf(constr2File,"\nX%d_%d_4_2_4 +↙
     X%d_%d_4_4_5 + X%d_%d_4_5_8 + X%d_%d_4_8_10 +",b,a,b,a,b,a,b,a);

        else fprintf(constr2File,"\nX%d_%d_4_2_4 + X%d_%d_4_4_5 + X%d_%d_4_5_8 + X%d_%   ↙
    d_4_8_10 - %8.2lfY4 < 0",b,a,b,a,b,a,b,a,ptr->tfixcost[12] );
        }
    }

    for (b=1; b<=pt->numJob;b++)
    {
        for ( int a = 1; a <= pt->numCont[b-1]; a++ )
        {
        if ((b <= k) & (a <= pt->numCont[k-1]-1))   fprintf(constr2File,"\nX%d_%d_5_1_4 +↙
     X%d_%d_5_4_6 + X%d_%d_5_6_7 + X%d_%d_5_7_10 +",b,a,b,a,b,a,b,a);
        else fprintf(constr2File,"\nX%d_%d_5_1_4 + X%d_%d_5_4_6 + X%d_%d_5_6_7 + X%d_%   ↙
    d_5_7_10 - %8.2lfY5 < 0",b,a,b, a,b, a,b,a,ptr->tfixcost[16] );
        }
    )

    for ( b=1; b<=pt->numJob;b++)
    {
        for ( int a = 1; a <= pt->numCont[b-1]; a++ )
        {
            if ((b <= k) & (a <= pt->numCont[k-1]-1)) fprintf(constr2File,"\nX%d_%d_6_2_5↙
     + X%d_%d_6_5_8 + X%d_%d_6_8_7 + X%d_%d_6_7_9 +", b,a,b,a,b,a,b,a);

            else fprintf(constr2File,"\nX%d_%d_6_2_5 + X%d_%d_6_5_8 + X%d_%d_6_8_7 + X%d_↙
    %d_6_7_9 - %8.2lfY6 < 0", b,a,b, a, b, a,b,a,ptr->tfixcost[20]);
        )
    )

fprintf(constr2File,"\n");

    for (b=1; b<=pt->numJob;b++)
    {
        for ( int a = 1; a <= pt->numCont[b-1]; a++ )
        {
        if ((b <= k) & (a <= pt->numCont[k-1]-1)) fprintf(constr2File,"\nX%d_%d_1_1_3 + "↙
    ,b,a);
        else fprintf(constr2File,"\tX%d_%d_1_1_3 + P1_1 = %d",b,a,ptr->tconcap[0]);
        )
    )

    for (b=1; b<=pt->numJob;b++)
    {
        for ( int a = 1; a <= pt->numCont[b-1]; a++ )
        {
        if ((b <= k) & (a <= pt->numCont[k-1]-1)) fprintf(constr2File,"\nX%d_%d_1_3_4 + "↙
    ,b,a);
        else fprintf(constr2File,"\tX%d_%d_1_3_4 + P1_3 = %d",b,a,ptr->tconcap[1]);
        )
    }

    for (b=1; b<=pt->numJob;b++)
    {
        for ( int a = 1; a <= pt->numCont[b-1]; a++ )
        {
        if ((b <= k) & (a <= pt->numCont[k-1]-1)) fprintf(constr2File,"\nX%d_%d_1_4_7 + "↙
    ,b,a);
        else fprintf(constr2File,"\tX%d_%d_1_4_7 + P1_4 = %d",b,a,ptr->tconcap[2]);
        }
    )

    for (b=1; b<=pt->numJob;b++)
```

```
    {
        for ( int a = 1; a <= pt->numCont[b-1]; a++ )
        {
        if ((b <= k) & (a <= pt->numCont[k-1]-1)) fprintf(constr2File,"\nX%d_%d_1_7_9 + "↙
    ,b,a);
        else fprintf(constr2File,"\tX%d_%d_1_7_9 + P1_7 = %d",b,a,ptr->tconcap[3]);
        )
    }

    for (b=1; b<=pt->numJob;b++)
    {
        for ( int a = 1; a <= pt->numCont[b-1]; a++ )
        {
        if ((b <= k) & (a <= pt->numCont[k-1]-1)) fprintf(constr2File,"\nX%d_%d_2_3_6 + "↙
    ,b,a);
        else fprintf(constr2File,"\tX%d_%d_2_3_6 + P2_3 = %d",b,a,ptr->tconcap[4]);
        }
    }

    for (b=1; b<=pt->numJob;b++)
    {
        for ( int a = 1; a <= pt->numCont[b-1]; a++ )
        {
        if ((b <= k) & (a <= pt->numCont[k-1]-1)) fprintf(constr2File,"\nX%d_%d_2_6_7 + "↙
    ,b,a);
        else fprintf(constr2File,"\tX%d_%d_2_6_7 + P2_6 = %d",b,a,ptr->tconcap[5]);
        }
    )

    for (b=1; b<=pt->numJob;b++)
    {
        for ( int a = 1; a <= pt->numCont[b-1]; a++ )
        {
        if ((b <= k) & (a <= pt->numCont[k-1]-1)) fprintf(constr2File,"\nX%d_%d_2_7_8 + "↙
    ,b,a);
        else fprintf(constr2File,"\tX%d_%d_2_7_8 + P2_7 = %d",b,a,ptr->tconcap[6]);
        }
    )

    for (b=1; b<=pt->numJob;b++)
    {
        for ( int a = 1; a <= pt->numCont[b-1]; a++ )
        {
        if ((b <= k) & (a <= pt->numCont[k-1]-1)) fprintf(constr2File,"\nX%d_%d_2_8_10 + ↙
    ",b,a);
        else fprintf(constr2File,"\tX%d_%d_2_8_10 + P2_8 = %d",b,a,ptr->tconcap[7]);
        }
    }

    for (b=1; b<=pt->numJob;b++)
    {
        for ( int a = 1; a <= pt->numCont[b-1]; a++ )
        {
        if ((b <= k) & (a <= pt->numCont[k-1]-1)) fprintf(constr2File,"\nX%d_%d_3_2_4 + "↙
    ,b,a);
        else fprintf(constr2File,"\tX%d_%d_3_2_4 + P3_2 = %d",b,a,ptr->tconcap[8]);
        }
    )

    for (b=1; b<=pt->numJob;b++)
    {
        for ( int a = 1; a <= pt->numCont[b-1]; a++ )
        {
        if ((b <= k) & (a <= pt->numCont[k-1]-1)) fprintf(constr2File,"\nX%d_%d_3_4_7 + "↙
    ,b,a);
        else fprintf(constr2File,"\tX%d_%d_3_4_7 + P3_4 = %d",b,a,ptr->tconcap[9]);
```

```
        )
    )

    for (b=1; b<=pt->numJob;b++)
    {
        for ( int a = 1; a <= pt->numCont[b-1]; a++ )
        {
        if ((b <= k) & (a <= pt->numCont[k-1]-1)) fprintf(constr2File,"\nX%d_%d_3_7_6 + "↙
    ,b,a);
        else fprintf(constr2File,"\tX%d_%d_3_7_6 + P3_7 = %d",b,a,ptr->tconcap[10]);
        )
    )

    for (b=1; b<=pt->numJob;b++)
    {
        for ( int a = 1; a <= pt->numCont[b-1]; a++ )
        {
        if ((b <= k) & (a <= pt->numCont[k-1]-1)) fprintf(constr2File,"\nX%d_%d_3_6_9 + "↙
    ,b,a);
        else fprintf(constr2File,"\tX%d_%d_3_6_9 + P3_6 = %d",b,a,ptr->tconcap[11]);
        )
    )

    for (b=1; b<=pt->numJob;b++)
    {
        for ( int a = 1; a <= pt->numCont[b-1]; a++ )
        {
        if ((b <= k) & (a <= pt->numCont[k-1]-1)) fprintf(constr2File,"\nX%d_%d_4_2_4 + "↙
    ,b,a);
        else fprintf(constr2File,"\tX%d_%d_4_2_4 + P4_2 = %d",b,a,ptr->tconcap[12]);
        )
    )

    for (b=1; b<=pt->numJob;b++)
    {
        for ( int a = 1; a <= pt->numCont[b-1]; a++ )
        {
        if ((b <= k) & (a <= pt->numCont[k-1]-1)) fprintf(constr2File,"\nX%d_%d_4_4_5 + "↙
    ,b,a);
        else fprintf(constr2File,"\tX%d_%d_4_4_5 + P4_4 = %d",b,a,ptr->tconcap[13]);
        )
    )

    for (b=1; b<=pt->numJob;b++)
    {
        for ( int a = 1; a <= pt->numCont[b-1]; a++ )
        {
        if ((b <= k) & (a <= pt->numCont[k-1]-1)) fprintf(constr2File,"\nX%d_%d_4_5_8 + "↙
    ,b,a);
        else fprintf(constr2File,"\tX%d_%d_4_5_8 + P4_5 = %d",b,a,ptr->tconcap[14]);
        )
    )

    for (b=1; b<=pt->numJob;b++)
    {
        for (int a = 1; a <= pt->numCont[b-1]; a++ )
        {
        if ((b <= k) & (a <= pt->numCont[k-1]-1)) fprintf(constr2File,"\nX%d_%d_4_8_10 + ↙
    ",b,a);
        else fprintf(constr2File,"\tX%d_%d_4_8_10 + P4_8 = %d",b,a,ptr->tconcap[15]);
        )
    )

    for (b=1; b<=pt->numJob;b++)
    {
        for ( int a = 1; a <= pt->numCont[b-1]; a++ )
```

```
    {
    if ((b <= k) & (a <= pt->numCont[k-1]-1)) fprintf(constr2File,"\nX%d_%d_5_1_4 + "↙
,b,a);
    else fprintf(constr2File,"\tX%d_%d_5_1_4 + P5_1 = %d",b,a,ptr->tconcap[16]);
    )
}

for (b=1; b<=pt->numJob;b++)
{
    for ( int a = 1; a <= pt->numCont[b-1]; a++ )
    {
    if ((b <= k) & (a <= pt->numCont[k-1]-1)) fprintf(constr2File,"\nX%d_%d_5_4_6 + "↙
,b,a);
    else fprintf(constr2File,"\tX%d_%d_5_4_6 + P5_4 = %d",b,a,ptr->tconcap[17]);
    }
}

for (b=1; b<=pt->numJob;b++)
{
    for ( int a = 1; a <= pt->numCont[b-1]; a++ )
    {
    if ((b <= k) & (a <= pt->numCont[k-1]-1)) fprintf(constr2File,"\nX%d_%d_5_6_7 + "↙
,b,a);
    else fprintf(constr2File,"\tX%d_%d_5_6_7 + P5_6 = %d",b,a,ptr->tconcap[18]);
    }
}

for (b=1; b<=pt->numJob;b++)
{
    for ( int a = 1; a <= pt->numCont[b-1]; a++ )
    {
    if ((b <= k) & (a <= pt->numCont[k-1]-1)) fprintf(constr2File,"\n X%d_%d_5_7_10 +↙
 ",b,a);
    else fprintf(constr2File,"\tX%d_%d_5_7_10 + P5_7 = %d",b,a,ptr->tconcap[19]);
    }
}

for (b=1; b<=pt->numJob;b++)
{
    for ( int a = 1; a <= pt->numCont[b-1]; a++ )
    {
    if ((b <= k) & (a <= pt->numCont[k-1]-1)) fprintf(constr2File,"\n X%d_%d_6_2_5 + ↙
",b,a);
    else fprintf(constr2File,"\tX%d_%d_6_2_5 + P6_2 = %d",b,a,ptr->tconcap[20]);
    }
}

for (b=1; b<=pt->numJob;b++)
{
    for ( int a = 1; a <= pt->numCont[b-1]; a++ )
    {
    if ((b <= k) & (a <= pt->numCont[k-1]-1)) fprintf(constr2File,"\n X%d_%d_6_5_8 + ↙
",b,a);
    else fprintf(constr2File,"\tX%d_%d_6_5_8 + P6_5 = %d",b,a,ptr->tconcap[21]);
    }
}

for (b=1; b<=pt->numJob;b++)
{
    for ( int a = 1; a <= pt->numCont[b-1]; a++ )
    {
    if ((b <= k) & (a <= pt->numCont[k-1]-1)) fprintf(constr2File,"\n X%d_%d_6_8_7 + ↙
",b,a);
    else fprintf(constr2File,"\tX%d_%d_6_8_7 + P6_8 = %d",b,a,ptr->tconcap[22]);
    }
)
```

```
    for (b=1; b<=pt->numJob;b++)
    (
        for ( int a = 1; a <= pt->numCont[b-1]; a++ )
        (
        if ((b <= k) & (a <= pt->numCont[k-1]-1)) fprintf(constr2File,"\n X%d_%d_6_7_9 + ↙
    ",b,a);
        else fprintf(constr2File,"\tX%d_%d_6_7_9 + P6_7 = %d",b,a,ptr->tconcap[23]);
        )
    }

}

void boundaries (struct JobTable *pt,struct TrainTable *ptr, FILE *boundFile, int jobNum ↙
     )
 {
fprintf(boundFile,"\n\nbinary");
for (int b = 1; b <= 6; b++)
    (
        fprintf(boundFile, "\nY%d", b);
    }

for (int k = 1; k <= pt->numJob; k++)
    (
    for (int c = 1; c <= pt->numCont[k-1]; c++)
        (
    fprintf(boundFile,"\n\nX%d_%d_1_1_3",k,c};
    fprintf(boundFile,"\nX%d_%d_1_3_4",k,c);
    fprintf(boundFile,"\nX%d_%d_1_4_7",k,c);
    fprintf(boundFile,"\nX%d_%d_1_7_9",k,c);
    fprintf(boundFile,"\nX%d_%d_2_3_6",k,c);
    fprintf(boundFile,"\nX%d_%d_2_6_7",k,c);
    fprintf(boundFile,"\nX%d_%d_2_7_8",k,c);
    fprintf(boundFile,"\nX%d_%d_2_8_10",k,c);
    fprintf(boundFile,"\nX%d_%d_3_2_4",k,c);
    fprintf(boundFile,"\nX%d_%d_3_4_7",k,c);
    fprintf(boundFile,"\nX%d_%d_3_7_6",k,c);
    fprintf(boundFile,"\nX%d_%d_3_6_9",k,c);
    fprintf(boundFile,"\nX%d_%d_4_2_4",k,c);
    fprintf(boundFile,"\nX%d_%d_4_4_5",k,c);
    fprintf(boundFile,"\nX%d_%d_4_5_8",k,c);
    fprintf(boundFile,"\nX%d_%d_4_8_10",k,c);
    fprintf(boundFile,"\nX%d_%d_5_1_4",k,c);
    fprintf(boundFile,"\nX%d_%d_5_4_6",k,c);
    fprintf(boundFile,"\nX%d_%d_5_6_7",k,c);
    fprintf(boundFile,"\nX%d_%d_5_7_10",k,c);
    fprintf(boundFile,"\nX%d_%d_6_2_5",k,c);
    fprintf(boundFile,"\nX%d_%d_6_5_8",k,c);
    fprintf(boundFile,"\nX%d_%d_6_8_7",k,c);
    fprintf(boundFile,"\nX%d_%d_6_7_9",k,c);

    fprintf(boundFile,"\n\nZ%d_%d_1_3",k,c);
    fprintf(boundFile,"\nZ%d_%d_1_4",k,c);
    fprintf(boundFile,"\nZ%d_%d_1_7",k,c);
    fprintf(boundFile,"\nZ%d_%d_2_6",k,c);
    fprintf(boundFile,"\nZ%d_%d_2_7",k,c);
    fprintf(boundFile,"\nZ%d_%d_2_8",k,c);
    fprintf(boundFile,"\nZ%d_%d_3_4",k,c);
    fprintf(boundFile,"\nZ%d_%d_3_7",k,c);
    fprintf(boundFile,"\nZ%d_%d_3_6",k,c);
    fprintf(boundFile,"\nZ%d_%d_4_4",k,c);
    fprintf(boundFile,"\nZ%d_%d_4_5",k,c);
    fprintf(boundFile,"\nZ%d_%d_4_8",k,c);
    fprintf(boundFile,"\nZ%d_%d_5_4",k,c);
    fprintf(boundFile,"\nZ%d_%d_5_6",k,c);
    fprintf(boundFile,"\nZ%d_%d_5_7",k,c);
```

```
    fprintf(boundFile,"\nZ%d_%d_6_5",k,c);
    fprintf(boundFile,"\nZ%d_%d_6_8",k,c);
    fprintf(boundFile,"\nZ%d_%d_6_7",k,c);
        }

    }
        fprintf(boundFile,"\nend");
}
```

.

```
//Attachment 4
#include <stdio.h>
#include <conio.h>
#include <windows.h>
#define MAXJOBS 5

struct JobTable{
    int numJob;
    int jobnum[MAXJOBS];
    int startITC[5];
    int endITC[5];
    int numCont[5];
    double startt[5];
    double deadlinet[5];
};

struct TrainTable{
    int numSegments;
    int trainnum[25];
    int tstITC[25];
    double departt[25];
    int tendITC[25];
    double arrivet[25];
    int tconcap[25];
    double tvarcost[25];
    double tfixcost[25];
    double tchgecost[25];
};

void shortpath(struct JobTable *,struct TrainTable *);
void getJobTable(struct JobTable *);
void getTrainTable(struct TrainTable *);


int main()
{
    struct JobTable jtable;
    struct TrainTable ttable;

    getJobTable(&jtable);
    getTrainTable(&ttable);

    shortpath(&jtable,&ttable);
    return 0;
}

void getJobTable(struct JobTable *pt)
{
    FILE * jobdatabase;
    int counter = 0;
    jobdatabase = fopen("jobdata1.txt", "r");

    fscanf(jobdatabase,"%d",  &pt->numJob);
    int numRows = pt->numJob;

    while (numRows >= 1)
    {
    fscanf(jobdatabase,"%d", &pt->jobnum[counter]);
    fscanf(jobdatabase,"%d", &pt->startITC[counter]);
    fscanf(jobdatabase,"%d", &pt->endITC[counter]);
    fscanf(jobdatabase,"%d", &pt->numCont[counter]);
    fscanf(jobdatabase,"%lf",&pt->startt[counter]);
    fscanf(jobdatabase,"%lf",&pt->deadlinet[counter]);
    numRows--;
    counter ++;
    }
```

```
    for (int t = 0; t < pt->numJob; t++)
        printf("\n%d    %d  %d  %d  %4.2lf  %4.2lf",pt->jobnum[t], pt->startITC[t],pt->
    endITC[t],pt->numCont[t],pt->startt[t],pt->deadlinet[t]);
    return;
}


void getTrainTable(struct TrainTable *pt){
FILE * traindatabase;
    int numRows = 0;
    int counter = 0;

traindatabase = fopen("traindatal.txt", "r");
while (traindatabase == NULL)
{
    printf("\nIt didn't open the file");
    return;
}
fscanf(traindatabase, "%d", &pt->numSegments);

int totRow = pt->numSegments;
while (totRow >= 1) {
    fscanf(traindatabase, "%d", &pt->trainnum[counter]);
    fscanf(traindatabase, "%d", &pt->tstITC[counter]);
    fscanf(traindatabase, "%lf", &pt->departt[counter]);
    fscanf(traindatabase, "%d", &pt->tendITC[counter]);
    fscanf(traindatabase, "%lf", &pt->arrivet[counter]);
    fscanf(traindatabase, "%d", &pt->tconcap[counter]);
    fscanf(traindatabase, "%lf", &pt->tvarcost[counter]);
    fscanf(traindatabase, "%lf", &pt->tfixcost[counter]);
    fscanf(traindatabase, "%lf", &pt->tchgecost[counter]);
    totRow--;
    counter ++;
    }
for (int t = 0; t < pt->numSegments; t++)
    printf("\n%d    %d  %4.2lf  %d  %4.2lf  %d  %4.2lf  %4.2lf  %4.2lf",pt->trainnum[t],
    pt->tstITC[t], pt->departt[t],pt->tendITC[t],pt->arrivet[t],pt->tconcap[t],pt->
    tvarcost[t],pt->tfixcost[t],pt->tchgecost[t]);

}
```

```
void shortpath(struct JobTable *pt, struct TrainTable *ptr)
{
    FILE * chpathfile;
    char cFileName[35];
    int numRows = 0;

    sprintf(cFileName, "cOutputCPtl.txt");
    chpathfile = fopen(cFileName, "w");
```

```
    fprintf(chpathfile, "\nGraph of Time Satisfying Routes");

    for (int x = 1; x <= pt->numJob; x++)
    {
        int cn = pt->endITC[x-1];
        double jd = pt->deadlinet[x-1];

    fprintf(chpathfile, "\n\t\t\t\tDestination ITC of Job %d = %d", x,pt->endITC[x-1]);
    fprintf(chpathfile, "\n");


            int nv[20][9], tu[20][9],nn[20][9],ncount[9];
            double jdt[20][9];
            int k,l;
            for (k=1; k<=20; k++)
            {
                for (l=1; l<=9; l++)
                {
                    nv[k-1][l-1] = 0;
                    tu[k-1][l-1] = 0;
                    nn[k-1][l-1] = 0;
                    jdt[k-1][l-1] = 0;
                }
            }

            for (l=1; l<=9; l++)
            {
                ncount[l-1] = 0;
            }

             k=0; l=1;
            int y=1,r=1;
            fprintf(chpathfile, "\nVisisted ITC's at level %d:",l);

            while (y<=ptr->numSegments)
            {
                if ((cn==ptr->tendITC[y-1])&&(jd>=ptr->arrivet[y-1]))
                {
                    k++;
                    nv[k-1][l-1] = cn;
                    tu[k-1][l-1] = ptr->trainnum[y-1];

                    nn[k-1][l-1] = ptr->tstITC[y-1];

                    fprintf(chpathfile, "\t\t\t %d_%d",nn[k-1][l-1],tu[k-1][l-1]);

                    jdt[k-1][l-1] = ptr->departt[y-1];
                    ncount[l-1]++;
                }

                y++;

            };
            fprintf(chpathfile, "\t\t\t#ITC on level %d:%d",l,ncount[l-1] );

            fprintf(chpathfile, "\n");

            do
            {
                l++;
                fprintf(chpathfile, "\nVisisted ITC's at level %d:",l);
                int q = 1;
                for (k = 1; k<=ncount[l-2]+1;k++)
                {
                    cn = nn[k-1][l-2];
                    jd = jdt[k-1][l-2];
```

```
                if ((cn==1)||(cn==2)) continue;

                if ((cn > 0)&&(cn < 9))
                {
        printf("\nFIRST IF IN cn=[%d] \t jd=[%1f]",cn,jd);  getchar();

                    for (y=1; y<= ptr->numSegments; y++)
                    {
        printf("\nFOR Y IN cn=[%d]\tptr->tendITC[y-1]=[%d]\n jd=[%1f]\t ptr->arrivet
    [y-1]=[%1f]",cn, ptr->tendITC[y-1],jd,ptr->arrivet[y-1]);
        getchar();
                        if ((cn==ptr->tendITC[y-1])&&(jd>=ptr->arrivet[y-1]))
                        {
        printf("\nSECOND IF");
        printf("\n cn[%d]\tptr->tstITC[y-1]=[%d]\njd=[%1f]\tptr->arrivet[y-1]=[%1f]\
    tq-1=%d,l-1=%d",cn,ptr->tstITC[y-1],jd,ptr->arrivet[y-1],q-1,l-1);
        getchar();
                        q++;
                        nv[q-1][l-1] = cn;
                        tu[q-1][l-1] = ptr->trainnum[y-1];

                            if ((ptr->tstITC[y-1]>0)&&(ptr->tstITC[y-1]<9))
                            {
                            nn[q-1][l-1] = ptr->tstITC[y-1];
                            fprintf(chpathfile, "\t%d_%d",nn[q-1][l-1],tu[q-1][l-1]);

                            jdt[q-1][l-1] = ptr->departt[y-1];
                            ncount[l-1]++;

        printf("\nTHIRD IF nn[q-1][l-1]=[%d]\tjdt[q-1][l-1]=[%1f]\tncount[l-1]=[%d]",
    nn[q-1][l-1],jdt[q-1][l-1],ncount[l-1]);
        getchar();

                                if ((nn[q-1][l-1] == 1)||(nn[q-1][l-1] == 2))
                                {
                                    if (nn[q-1][1-1] == pt->startITC[x-1])
                                    {
                                    fprintf(chpathfile, " EndRoute %d", r);
                                    r++;
                                    }
                                    continue;
                                }
                            }
                        }
                    )

                }

            fprintf(chpathfile,"\t*");
            }
            fprintf(chpathfile, "\t#ITC on level %d:%d",l,ncount[l-1] );

    fprintf(chpathfile, "\n---  ---  ---   ---   ---   ---                ---      --
-   ---   ---   ---   ---   ---");

        }

        while (ncount[l-1]!=0);
        fprintf(chpathfile, "\nEND of cycle for JOB %d",x);
    }
fprintf(chpathfile, "\n");
}
```

Attachment 5

Log started (V8.1.0) Sun May 09 12:01:10 2004

Integer optimal, tolerance (0.0001/1e-006) : Objective = 1.1501400000e+005

Variable Name	Solution Value
X1_1_1_4_7	1.000000
Z1_1_1_7	1.000000
X1_1_2_7_8	1.000000
X1_1_2_8_10	1.000000
X1_1_5_1_4	1.000000
Z1_1_5_4	1.000000
X1_2_1_4_7	1.000000
Z1_2_1_7	1.000000
X1_2_2_7_8	1.000000
X1_2_2_8_10	1.000000
X1_2_5_1_4	1.000000
Z1_2_5_4	1.000000
X1_3_1_4_7	1.000000
Z1_3_1_7	1.000000
X1_3_2_7_8	1.000000
X1_3_2_8_10	1.000000
X1_3_5_1_4	1.000000
Z1_3_5_4	1.000000
X1_4_1_4_7	1.000000
Z1_4_1_7	1.000000
X1_4_2_7_8	1.000000
X1_4_2_8_10	1.000000
X1_4_5_1_4	1.000000
Z1_4_5_4	1.000000
X1_5_1_4_7	1.000000
Z1_5_1_7	1.000000
X1_5_2_7_8	1.000000
X1_5_2_8_10	1.000000
X1_5_5_1_4	1.000000
Z1_5_5_4	1.000000
X1_6_1_4_7	1.000000
Z1_6_1_7	1.000000
X1_6_2_7_8	1.000000
X1_6_2_8_10	1.000000
X1_6_5_1_4	1.000000
Z1_6_5_4	1.000000
X1_7_1_4_7	1.000000
Z1_7_1_7	1.000000
X1_7_2_7_8	1.000000
X1_7_2_8_10	1.000000
X1_7_5_1_4	1.000000
Z1_7_5_4	1.000000
X1_8_1_4_7	1.000000
Z1_8_1_7	1.000000
X1_8_2_7_8	1.000000
X1_8_2_8_10	1.000000
X1_8_5_1_4	1.000000
Z1_8_5_4	1.000000
X1_9_1_4_7	1.000000
Z1_9_1_7	1.000000
X1_9_2_7_8	1.000000
X1_9_2_8_10	1.000000
X1_9_5_1_4	1.000000
Z1_9_5_4	1.000000
X1_10_1_4_7	1.000000
Z1_10_1_7	1.000000

	result test1
X1_10_2_7_8	1.000000
X1_10_2_8_10	1.000000
X1_10_5_1_4	1.000000
Z1_10_5_4	1.000000
X1_11_1_4_7	1.000000
Z1_11_1_7	1.000000
X1_11_2_7_8	1.000000
X1_11_2_8_10	1.000000
X1_11_5_1_4	1.000000
Z1_11_5_4	1.000000
X1_12_1_4_7	1.000000
Z1_12_1_7	1.000000
X1_12_2_7_8	1.000000
X1_12_2_8_10	1.000000
X1_12_5_1_4	1.000000
Z1_12_5_4	1.000000
X1_13_1_4_7	1.000000
Z1_13_1_7	1.000000
X1_13_2_7_8	1.000000
X1_13_2_8_10	1.000000
X1_13_5_1_4	1.000000
Z1_13_5_4	1.000000
X1_14_1_4_7	1.000000
Z1_14_1_7	1.000000
X1_14_2_7_8	1.000000
X1_14_2_8_10	1.000000
X1_14_5_1_4	1.000000
Z1_14_5_4	1.000000
X1_15_1_4_7	1.000000
Z1_15_1_7	1.000000
X1_15_2_7_8	1.000000
X1_15_2_8_10	1.000000
X1_15_5_1_4	1.000000
Z1_15_5_4	1.000000
X1_16_1_4_7	1.000000
Z1_16_1_7	1.000000
X1_16_2_7_8	1.000000
X1_16_2_8_10	1.000000
X1_16_5_1_4	1.000000
Z1_16_5_4	1.000000
X1_17_1_4_7	1.000000
Z1_17_1_7	1.000000
X1_17_2_7_8	1.000000
X1_17_2_8_10	1.000000
X1_17_5_1_4	1.000000
Z1_17_5_4	1.000000
X1_18_1_4_7	1.000000
Z1_18_1_7	1.000000
X1_18_2_7_8	1.000000
X1_18_2_8_10	1.000000
X1_18_5_1_4	1.000000
Z1_18_5_4	1.000000
X1_19_1_4_7	1.000000
Z1_19_1_7	1.000000
X1_19_2_7_8	1.000000
X1_19_2_8_10	1.000000
X1_19_5_1_4	1.000000
Z1_19_5_4	1.000000
X2_1_1_1_3	1.000000
Z2_1_1_3	1.000000
X2_1_2_3_6	1.000000
Z2_1_2_6	1.000000
X2_1_3_6_9	1.000000

	result test1
x2_2_1_1_3	1.000000
z2_2_1_3	1.000000
x2_2_2_3_6	1.000000
z2_2_2_6	1.000000
x2_2_3_6_9	1.000000
x2_3_1_1_3	1.000000
z2_3_1_3	1.000000
x2_3_2_3_6	1.000000
z2_3_2_6	1.000000
x2_3_3_6_9	1.000000
x2_4_1_1_3	1.000000
z2_4_1_3	1.000000
x2_4_2_3_6	1.000000
z2_4_2_6	1.000000
x2_4_3_6_9	1.000000
x2_5_1_1_3	1.000000
z2_5_1_3	1.000000
x2_5_2_3_6	1.000000
z2_5_2_6	1.000000
x2_5_3_6_9	1.000000
x2_6_1_1_3	1.000000
z2_6_1_3	1.000000
x2_6_2_3_6	1.000000
z2_6_2_6	1.000000
x2_6_3_6_9	1.000000
x2_7_1_1_3	1.000000
z2_7_1_3	1.000000
x2_7_2_3_6	1.000000
z2_7_2_6	1.000000
x2_7_3_6_9	1.000000
x2_8_1_1_3	1.000000
z2_8_1_3	1.000000
x2_8_2_3_6	1.000000
z2_8_2_6	1.000000
x2_8_3_6_9	1.000000
x2_9_1_1_3	1.000000
z2_9_1_3	1.000000
x2_9_2_3_6	1.000000
z2_9_2_6	1.000000
x2_9_3_6_9	1.000000
x2_10_1_1_3	1.000000
z2_10_1_3	1.000000
x2_10_2_3_6	1.000000
z2_10_2_6	1.000000
x2_10_3_6_9	1.000000
x2_11_1_1_3	1.000000
z2_11_1_3	1.000000
x2_11_2_3_6	1.000000
z2_11_2_6	1.000000
x2_11_3_6_9	1.000000
x2_12_1_1_3	1.000000
z2_12_1_3	1.000000
x2_12_2_3_6	1.000000
z2_12_2_6	1.000000
x2_12_3_6_9	1.000000
x2_13_1_1_3	1.000000
z2_13_1_3	1.000000
x2_13_2_3_6	1.000000
z2_13_2_6	1.000000
x2_13_3_6_9	1.000000
x2_14_1_1_3	1.000000
z2_14_1_3	1.000000
x2_14_2_3_6	1.000000

	result test1
z2_14_2_6	1.000000
x2_14_3_6_9	1.000000
x2_15_1_1_3	1.000000
z2_15_1_3	1.000000
x2_15_2_3_6	1.000000
z2_15_2_6	1.000000
x2_15_3_6_9	1.000000
x2_16_1_1_3	1.000000
z2_16_1_3	1.000000
x2_16_2_3_6	1.000000
z2_16_2_6	1.000000
x2_16_3_6_9	1.000000
x2_17_1_1_3	1.000000
z2_17_1_3	1.000000
x2_17_2_3_6	1.000000
z2_17_2_6	1.000000
x2_17_3_6_9	1.000000
x2_18_1_1_3	1.000000
z2_18_1_3	1.000000
x2_18_2_3_6	1.000000
z2_18_2_6	1.000000
x2_18_3_6_9	1.000000
x2_19_1_1_3	1.000000
z2_19_1_3	1.000000
x2_19_2_3_6	1.000000
z2_19_2_6	1.000000
x2_19_3_6_9	1.000000
x2_20_1_1_3	1.000000
z2_20_1_3	1.000000
x2_20_2_3_6	1.000000
z2_20_2_6	1.000000
x2_20_3_6_9	1.000000
x3_1_3_2_4	1.000000
x3_1_3_4_7	1.000000
x3_1_3_7_6	1.000000
x3_1_3_6_9	1.000000
x3_2_3_2_4	1.000000
x3_2_3_4_7	1.000000
x3_2_3_7_6	1.000000
x3_2_3_6_9	1.000000
x3_3_3_2_4	1.000000
x3_3_3_4_7	1.000000
x3_3_3_7_6	1.000000
x3_3_3_6_9	1.000000
x3_4_3_2_4	1.000000
x3_4_3_4_7	1.000000
x3_4_3_7_6	1.000000
x3_4_3_6_9	1.000000
x3_5_3_2_4	1.000000
x3_5_3_4_7	1.000000
x3_5_3_7_6	1.000000
x3_5_3_6_9	1.000000
x3_6_3_2_4	1.000000
x3_6_3_4_7	1.000000
x3_6_3_7_6	1.000000
x3_6_3_6_9	1.000000
x3_7_3_2_4	1.000000
x3_7_3_4_7	1.000000
x3_7_3_7_6	1.000000
x3_7_3_6_9	1.000000
x3_8_3_2_4	1.000000
x3_8_3_4_7	1.000000
x3_8_3_7_6	1.000000

X3_8_3_6_9	1.000000
X3_9_3_2_4	1.000000
X3_9_3_4_7	1.000000
X3_9_3_7_6	1.000000
X3_9_3_6_9	1.000000
X3_10_3_2_4	1.000000
X3_10_3_4_7	1.000000
X3_10_3_7_6	1.000000
X3_10_3_6_9	1.000000
X3_11_3_2_4	1.000000
X3_11_3_4_7	1.000000
X3_11_3_7_6	1.000000
X3_11_3_6_9	1.000000
X3_12_3_2_4	1.000000
X3_12_3_4_7	1.000000
X3_12_3_7_6	1.000000
X3_12_3_6_9	1.000000
X3_13_3_2_4	1.000000
X3_13_3_4_7	1.000000
X3_13_3_7_6	1.000000
X3_13_3_6_9	1.000000
X3_14_3_2_4	1.000000
X3_14_3_4_7	1.000000
X3_14_3_7_6	1.000000
X3_14_3_6_9	1.000000
X3_15_3_2_4	1.000000
X3_15_3_4_7	1.000000
X3_15_3_7_6	1.000000
X3_15_3_6_9	1.000000
X3_16_3_2_4	1.000000
X3_16_3_4_7	1.000000
X3_16_3_7_6	1.000000
X3_16_3_6_9	1.000000
X3_17_3_2_4	1.000000
X3_17_3_4_7	1.000000
X3_17_3_7_6	1.000000
X3_17_3_6_9	1.000000
X3_18_3_2_4	1.000000
X3_18_3_4_7	1.000000
X3_18_3_7_6	1.000000
X3_18_3_6_9	1.000000
X3_19_3_2_4	1.000000
X3_19_3_4_7	1.000000
X3_19_3_7_6	1.000000
X3_19_3_6_9	1.000000
X3_20_3_2_4	1.000000
X3_20_3_4_7	1.000000
X3_20_3_7_6	1.000000
X3_20_3_6_9	1.000000
X3_21_3_2_4	1.000000
X3_21_3_4_7	1.000000
X3_21_3_7_6	1.000000
X3_21_3_6_9	1.000000
X4_1_2_8_10	1.000000
X4_1_6_2_5	1.000000
X4_1_6_5_8	1.000000
Z4_1_6_8	1.000000
X4_2_2_8_10	1.000000
X4_2_6_2_5	1.000000
X4_2_6_5_8	1.000000
Z4_2_6_8	1.000000
X4_3_2_8_10	1.000000
X4_3_6_2_5	1.000000

	result test1
X4_3_6_5_8	1.000000
Z4_3_6_8	1.000000
X4_4_2_8_10	1.000000
X4_4_6_2_5	1.000000
X4_4_6_5_8	1.000000
Z4_4_6_8	1.000000
X4_5_2_8_10	1.000000
X4_5_6_2_5	1.000000
X4_5_6_5_8	1.000000
Z4_5_6_8	1.000000
X4_6_2_8_10	1.000000
X4_6_6_2_5	1.000000
X4_6_6_5_8	1.000000
Z4_6_6_8	1.000000
X4_7_2_8_10	1.000000
X4_7_6_2_5	1.000000
X4_7_6_5_8	1.000000
Z4_7_6_8	1.000000
X4_8_2_8_10	1.000000
X4_8_6_2_5	1.000000
X4_8_6_5_8	1.000000
Z4_8_6_8	1.000000
X4_9_2_8_10	1.000000
X4_9_6_2_5	1.000000
X4_9_6_5_8	1.000000
Z4_9_6_8	1.000000
X4_10_2_8_10	1.000000
X4_10_6_2_5	1.000000
X4_10_6_5_8	1.000000
Z4_10_6_8	1.000000
X4_11_2_8_10	1.000000
X4_11_6_2_5	1.000000
X4_11_6_5_8	1.000000
Z4_11_6_8	1.000000
X4_12_2_8_10	1.000000
X4_12_6_2_5	1.000000
X4_12_6_5_8	1.000000
Z4_12_6_8	1.000000
X4_13_2_8_10	1.000000
X4_13_6_2_5	1.000000
X4_13_6_5_8	1.000000
Z4_13_6_8	1.000000
X4_14_2_8_10	1.000000
X4_14_6_2_5	1.000000
X4_14_6_5_8	1.000000
Z4_14_6_8	1.000000
X4_15_2_8_10	1.000000
X4_15_6_2_5	1.000000
X4_15_6_5_8	1.000000
Z4_15_6_8	1.000000
X4_16_2_8_10	1.000000
X4_16_6_2_5	1.000000
X4_16_6_5_8	1.000000
Z4_16_6_8	1.000000
X4_17_2_8_10	1.000000
X4_17_6_2_5	1.000000
X4_17_6_5_8	1.000000
Z4_17_6_8	1.000000
X4_18_2_8_10	1.000000
X4_18_6_2_5	1.000000
X4_18_6_5_8	1.000000
Z4_18_6_8	1.000000
X4_19_2_8_10	1.000000

	result test1
X4_19_6_2_5	1.000000
X4_19_6_5_8	1.000000
Z4_19_6_8	1.000000
X4_20_2_8_10	1.000000
X4_20_6_2_5	1.000000
X4_20_6_5_8	1.000000
Z4_20_6_8	1.000000
X4_21_2_8_10	1.000000
X4_21_6_2_5	1.000000
X4_21_6_5_8	1.000000
Z4_21_6_8	1.000000
X4_22_2_8_10	1.000000
X4_22_6_2_5	1.000000
X4_22_6_5_8	1.000000
Z4_22_6_8	1.000000
Y1	1.000000
Y2	1.000000
Y3	1.000000
Y5	1.000000
Y6	1.000000
P1_1	1.000000
P1_3	41.000000
P1_4	12.000000
P1_7	51.000000
P2_3	22.000000
P2_6	32.000000
P2_7	33.000000
P2_8	21.000000
P3_2	12.000000
P3_4	22.000000
P3_7	32.000000
P3_6	2.000000
P4_2	44.000000
P4_4	34.000000
P4_5	54.000000
P4_8	24.000000
P5_1	16.000000
P5_4	45.000000
P5_6	25.000000
P5_7	55.000000
P6_2	4.000000
P6_5	24.000000
P6_8	36.000000
P6_7	46.000000

All other variables in the range 1-3966 are zero.

Attachment 6

Graph of Time Satisfying Routes

Destination ITC of Job 1 = 10

Visisted ITC's at level 1: 8_2 8_4 #ITC on level 1:2

Visisted ITC's at level 2: 7_2 5_4 5_6 * 5_4 5_6 * * #ITC on level 2:5

Visisted ITC's at level 3: * 4_1 6_2 4_3 8_6 * 4_4 2_6 * 2_6 * 4_4 2_6 * 2_6 * #ITC on level 3:10

Visisted ITC's at level 4: * 3_1 2_3 2_4 1_5 EndRoute 1 * 3_2 * 2_3 * 5_6 * 2_3 2_4 1_5 EndRoute 2 * 2_3 2_4 1_5 EndRoute 3 * #ITC on level 4:13

Visisted ITC's at level 5: * 1_1 EndRoute 4 * 1_1 EndRoute 5 * 2_6 * #ITC on level 5:3

Visisted ITC's at level 6: * #ITC on level 6:0

END of cycle for JOB 1

Destination ITC of Job 2 = 9

Visisted ITC's at level 1: 7_1 6_3 #ITC on level 1:2

Visisted ITC's at level 2: 4_1 6_2 4_3 8_6 * 3_2 7_3 * * #ITC on level 2:6

Visisted ITC's at level 3: * 3_1 2_3 2_4 1_5 EndRoute 1 * 3_2 * 2_3 * 5_6 * 1_1 EndRoute 2 * 4_3 * #ITC on level 3:9

Visisted ITC's at level 4: * 1_1 EndRoute 3 * 1_1 EndRoute 4 * 2_6 * 2_3 * #ITC on level 4:4

Visisted ITC's at level 5: * #ITC on level 5:0

END of cycle for JOB 2

Destination ITC of Job 3 = 9

Visisted ITC's at level 1: 6_3 #ITC on level 1:1

Visisted ITC's at level 2: 3_2 7_3 * * #ITC on level 2:2

Visisted ITC's at level 3: * 1_1 * 4_3 * #ITC on level 3:2

```
---  ---  ---   ---   ---   ---                    ---      ---   ---   ---   ---   ---   ---
Visisted ITC's at level 4:      *       2_3 EndRoute 1  *       #ITC on level 4:1
---  ---  ---   ---   ---   ---                    ---      ---   ---   ---   ---   ---   ---
Visisted ITC's at level 5:      *       #ITC on level 5:0
---  ---  ---   ---   ---   ---                    ---      ---   ---   ---   ---   ---   ---
END of cycle for JOB 3
                                Destination ITC of Job 4 = 10

Visisted ITC's at level 1:                          8_2                     8_4                     #ITC on
level 1:2

Visisted ITC's at level 2:      7_2     5_4     5_6     *       5_4     5_6     *       *       #ITC on
level 2:5
---  ---  ---   ---   ---   ---                    ---      ---   ---   ---   ---   ---   ---
Visisted ITC's at level 3:      *       4_1     6_2     4_3     8_6     *       4_4     2_6 EndRoute 1  *
2_6 EndRoute 2  *       4_4     2_6 EndRoute 3  *       2_6 EndRoute 4  *       #ITC on level 3:10
---  ---  ---   ---   ---   ---                    ---      ---   ---   ---   ---   ---   ---
Visisted ITC's at level 4:      *       3_1     2_3 EndRoute 5  2_4 EndRoute 6  1_5     *       3_2     *
2_3 EndRoute 7  *       5_6     *       2_3 EndRoute 8  2_4 EndRoute 9  1_5     *       2_3 EndRoute 10 2_4
EndRoute 11     1_5     *       #ITC on level 4:13
---  ---  ---   ---   ---   ---                    ---      ---   ---   ---   ---   ---   ---
Visisted ITC's at level 5:      *       1_1     *       1_1     *       2_6 EndRoute 12 *       #ITC on
level 5:3
---  ---  ---   ---   ---   ---                    ---      ---   ---   ---   ---   ---   ---
Visisted ITC's at level 6:      *       #ITC on level 6:0
---  ---  ---   ---   ---   ---                    ---      ---   ---   ---   ---   ---   ---
END of cycle for JOB 4
```

SHORTEST ROUTE -- ORIGINAL DATA

Title: thesis-alljobs t1

	N1:1	N2:2	N3:3/1	N4:3/2	N5:4/1	N6:4/3
N1:1		infinity	240.00	infinity	infinity	infinity
N2:2	infinity		infinity	infinity	infinity	270.00
N3:3/1	infinity	infinity		20.00	180.00	infinity
N4:3/2	infinity	infinity	20.00		infinity	infinity
N5:4/1	infinity	infinity	infinity	infinity		infinity
N6:4/3	infinity	infinity	infinity	infinity	20.00	
N7:4/4	infinity	infinity	infinity	infinity	20.00	infinity
N8:4/5	infinity	infinity	infinity	infinity	20.00	infinity
N9:5/4	infinity	infinity	infinity	infinity	infinity	infinity
N10:5/6	infinity	infinity	infinity	infinity	infinity	infinity
N11:6/2	infinity	infinity	infinity	infinity	infinity	infinity
N12:6/3	infinity	infinity	infinity	infinity	infinity	infinity
N13:6/5	infinity	infinity	infinity	infinity	infinity	infinity
N14:7/1	infinity	infinity	infinity	infinity	infinity	infinity
N15:7/2	infinity	infinity	infinity	infinity	infinity	infinity
N16:7/3	infinity	infinity	infinity	infinity	infinity	infinity
N17:7/5	infinity	infinity	infinity	infinity	infinity	infinity
N18:7/6	infinity	infinity	infinity	infinity	infinity	infinity
N19:8/2	infinity	infinity	infinity	infinity	infinity	infinity
N20:8/4	infinity	infinity	infinity	infinity	infinity	infinity
N21:8/6	infinity	infinity	infinity	infinity	infinity	infinity
N22:9	infinity	infinity	infinity	infinity	infinity	infinity
N23:10	infinity	infinity	infinity	infinity	infinity	infinity

	N7:4/4	N8:4/5	N9:5/4	N10:5/6	N11:6/2	N12:6/3
N1:1	infinity	270.00	infinity	infinity	infinity	infinity
N2:2	270.00	infinity	infinity	240.00	infinity	infinity
N3:3/1	infinity	infinity	infinity	infinity	infinity	infinity
N4:3/2	infinity	infinity	infinity	infinity	240.00	infinity
N5:4/1	infinity	20.00	infinity	infinity	infinity	infinity
N6:4/3	20.00	20.00	infinity	infinity	infinity	infinity
N7:4/4		20.00	180.00	infinity	infinity	infinity
N8:4/5	20.00		infinity	infinity	infinity	infinity
N9:5/4	infinity	infinity		infinity	infinity	infinity
N10:5/6	infinity	infinity	20.00		infinity	infinity
N11:6/2	infinity	infinity	infinity	infinity		20.00
N12:6/3	infinity	infinity	infinity	infinity	infinity	
N13:6/5	infinity	infinity	infinity	infinity	infinity	infinity
N14:7/1	infinity	infinity	infinity	infinity	infinity	infinity
N15:7/2	infinity	infinity	infinity	infinity	infinity	infinity
N16:7/3	infinity	infinity	infinity	infinity	infinity	144.00
N17:7/5	infinity	infinity	infinity	infinity	infinity	infinity
N18:7/6	infinity	infinity	infinity	infinity	infinity	infinity
N19:8/2	infinity	infinity	infinity	infinity	infinity	infinity
N20:8/4	infinity	infinity	infinity	infinity	infinity	infinity
N21:8/6	infinity	infinity	infinity	infinity	infinity	infinity
N22:9	infinity	infinity	infinity	infinity	infinity	infinity
N23:10	infinity	infinity	infinity	infinity	infinity	infinity

	N13:6/5	N14:7/1	N15:7/2	N16:7/3	N17:7/5	N18:7/6
N1:1	infinity	infinity	infinity	infinity	infinity	infinity
N2:2	infinity	infinity	infinity	infinity	infinity	infinity

N3:3/1	infinity	infinity	infinity	infinity	infinity	infinity
N4:3/2	infinity	infinity	infinity	infinity	infinity	infinity
N5:4/1	infinity	240.00	infinity	infinity	infinity	infinity
N6:4/3	infinity	infinity	infinity	240.00	infinity	infinity
N7:4/4	infinity	infinity	infinity	infinity	infinity	infinity
N8:4/5	360.00	infinity	infinity	infinity	infinity	infinity
N9:5/4	infinity	infinity	infinity	infinity	infinity	infinity
N10:5/6	infinity	infinity	infinity	infinity	infinity	infinity
N11:6/2	20.00	infinity	144.00	infinity	infinity	infinity
N12:6/3	20.00	infinity	infinity	infinity	infinity	infinity
N13:6/5		infinity	infinity	infinity	144.00	infinity
N14:7/1	infinity		20.00	infinity	20.00	20.00
N15:7/2	infinity	20.00		infinity	20.00	20.00
N16:7/3	infinity	20.00	20.00		20.00	20.00
N17:7/5	infinity	infinity	infinity	infinity		infinity
N18:7/6	infinity	20.00	20.00	infinity	20.00	
N19:8/2	infinity	infinity	infinity	infinity	infinity	infinity
N20:8/4	infinity	infinity	infinity	infinity	infinity	infinity
N21:8/6	infinity	infinity	infinity	infinity	infinity	180.00
N22:9	infinity	infinity	infinity	infinity	infinity	infinity
N23:10	infinity	infinity	infinity	infinity	infinity	infinity

	N19:8/2	N20:8/4	N21:8/6	N22:9	N23:10
N1:1	infinity	infinity	infinity	infinity	infinity
N2:2	infinity	infinity	infinity	infinity	infinity
N3:3/1	infinity	infinity	infinity	infinity	infinity
N4:3/2	infinity	infinity	infinity	infinity	infinity
N5:4/1	infinity	infinity	infinity	infinity	infinity
N6:4/3	infinity	infinity	infinity	infinity	infinity
N7:4/4	infinity	infinity	infinity	infinity	infinity
N8:4/5	infinity	infinity	infinity	infinity	infinity
N9:5/4	infinity	240.00	infinity	infinity	infinity
N10:5/6	infinity	infinity	240.00	infinity	infinity
N11:6/2	infinity	infinity	infinity	infinity	infinity
N12:6/3	infinity	infinity	infinity	180.00	infinity
N13:6/5	infinity	infinity	infinity	infinity	infinity
N14:7/1	infinity	infinity	infinity	180.00	infinity
N15:7/2	180.00	infinity	infinity	infinity	infinity
N16:7/3	infinity	infinity	infinity	infinity	infinity
N17:7/5	infinity	infinity	infinity	infinity	infinity
N18:7/6	infinity	infinity	180.00	infinity	infinity
N19:8/2		infinity	infinity	infinity	180.00
N20:8/4	20.00		infinity	infinity	180.00
N21:8/6	20.00	20.00		infinity	infinity
N22:9	infinity	infinity	infinity		infinity
N23:10	infinity	infinity	infinity	infinity	

SHORTEST ROUTES -- FLOYD'S OUTPUT SUMMARY

Title: thesis-alljobs t1

From	To	Distance	Route
1-1	23-10	890.00	1- 8- 7- 9- 20- 23


FUJIFI
CD-R
700MB
80min.
OPTSOLFILES