Train Dispatching: Heuristic Optimization

Sanusi Ayinla Afeez

Master Thesis
Computer
Engineering
Nr: E3139D

2006

DEGREE PROJECT
Computer Engineering

Programme	Reg number	Extent
Master of Science in Computer Engineering	E3139D	30 ECTS
Name of student	Year-Month-Day	
Sanusi Ayinla Afeez	2006-May-11	
Supervisor	Examiner	
Pascal Rebreyend	Prof. Mark Dougherty	
Company/Department	Supervisor at the Company/Department	
Computer Engineering Department	Pascal Rebreyend	
Title		
Train Dispatching: Heuristic Optimization		
Keywords		
Train Dispatching, Train Scheduling and Rescheduling, Heuristic Optimization		

Abstract

Train dispatchers faces lots of challenges due to conflicts which causes delays of trains as a result of solving possible dispatching problems the network faces. The major challenge is for the train dispatchers to make the right decisions and have reliable, cost effective and much more faster approaches needed to solve dispatching problems. This thesis work provides detail information on the implementation of different heuristic algorithms for train dispatchers in solving train dispatching problems. The library data files used are in xml file format and deals with both single and double tracks between main stations. The main objective of this work is to build different heuristic algorithms to solve unexpected delays faced by train dispatchers and to help in making right decisions on steps to take to have reliable and cost effective solution to the problems. These heuristics algorithms proposed were able to help the dispatcher in making right decisions when solving train dispatching problems.

ACKNOWLEDGMENTS

I wish to acknowledge the patience of my supervisor Dr. Pascal Rebreyend for this thesis work and to thank him for this topic which has helped me in having more knowledge in the area of Artificial Intelligence.

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

Railroad systems have come of age in the development of more reliable and cost effective mode of transportation in the world. Its role in the daily lives of passengers for long distance journeys and movement of goods for large and small scale industries has made rail systems to be more competitive with other means of transportation in providing cost effective and efficient means of transportation. Railroad systems are planned to have detailed timetable to meet customers' demands and reduces delays that may occur due to unforeseen circumstances during its journey between stations. Due to the planning of its operations, rail systems are considered to be more reliable, cost effective, and safer than other means of transportation.

In Sweden, the railroad systems have different types of trains involved in its daily operation in meeting customers' demands. Depending on customers' choices, different types of trains involved in daily operation are trucking trains (also known as freight trains) for goods movement, high speed trains (X2000) and intercity trains for passengers are the common ones in Sweden's rail system. The railroad systems has proved to be reliable in meeting customers demands, which can be proven as a result of different approaches used in solving dispatching problems or scheduling problems.

The railroad networks are mostly composed of both single and double line tracks. Single line tracks are tracks that can accommodate only one train at a time while double line tracks are tracks that can accommodate up to two trains at a time traveling in either opposite direction or in the same direction. Trains moving along a single line tracks can only overtake and cross each other at specific locations such as sidings and meeting points located at regular intervals along the lines [4].

The use of a well planned and detailed timetable for trains in the rail network is highly required in order to keep trains from having collisions and at the same time move the trains efficiently over the network. The train dispatcher makes use of the timetable to plan daily movements of trains within the network by dispatching trains at the specified time given in the timetable and also guides the arrival of trains to their destinations. The timetable consists of departure and arrival times for each train in the network along its track sections of its route

by strictly meeting traffic constraints rules, and also making sure that all trains crossing and overtakes happen at passing sidings or intermediate stations. Due to delays train dispatchers may encounter conflicts within the network, the timetable will also contain information about the trains delayed due to conflicts with other trains. This information will make it easier to solve train dispatching problems during its operation [4].

Delays in railroad systems can be divided into primary and secondary delays. Primary delays are the types of delays caused on the train from outside and not by other trains. Primary delays can be caused by malfunctioning infrastructure, malfunctioning roll stock, excessive alighting, and bad weather condition etc, while secondary delays are those caused by earlier delays of other trains within the network because of shared use of same infrastructures, passenger transfers, transfers in crew schedules, rolling stock connections, dispatching actions etc [1].

In solving conflicts encountered by train dispatcher, a reliable and cost effective approach or steps needs to be employed to achieve this goal. This approach or steps will help the dispatchers in making reliable decisions that will give better or best way of solving conflicts encountered. There has been lots of work done in respect of solving train dispatching or scheduling problems and still require better solutions to achieve a more reliable railway systems.

1.1 RAILROAD OPERATION

Rail transport is a form of movement of passengers and goods along two parallel rails mostly made of steel. The rail tracks provide very smooth and hard surfaces on which the wheels of the train may roll with a minimum of frictions [13]. The weights of the trains considered to move along the tracks are carefully chosen to avoid accident, so different tracks may be available to different trains being used depending on the usage of the train involved.

The planning of the railway operations needs some important infrastructures that will make the operation of the systems to function well. These infrastuctures needed includes tracks, signals, switches, crossovers etc, are all needed to have a reliable railroad system. Railroads are built with different number of tracks depending on the network of the system. Rail lines that carry little traffic are often built with a single track used by trains in both directions, passing sidings, crossovers which consist of short stretches of double track, are used along the line to allow trains to pass each other and travel in opposite directions [13]. Double line tracks

operate very much simpler than single line track, opposing trains do not exist, as long as every train keeps to its proper track of assigned direction using the dispatching plan. Double line tracks can accommodate up to two trains at a time traveling in either opposite direction or in the same direction by strictly following the well detailed timetable to avoid collision with other trains within the network.

1.2 RAILROAD DISASTER AND SAFETY

The safety of railroad system is an important issue in its developments for providing passengers comfort and reliable means of transportation. There are different types of hazards that can occur during railroad operation, such as human errors, recklessness, mechanical failures, collisions and wrecks. In order to reduce such hazards, safety rules are employed to help in providing a well secured means of transportation and such rules needs to be strictly followed [13]. Accidents that can happen during train operations are train derailment, head-on collision with other trains coming from the opposite direction and also collision with vehicles at the level crossing points. To reduce and take care of such accidents, there are different measures employed to combat these accidents. One important safety measure is using railroad signalling to prevent trains from colliding with other trains. This helps in preventing collision since trains run on fixed rails and cannot stop in time to avoid colliding with obstacle by reducing its speed as quickly as vehicles does. Another means of preventing accident is using train whistles, it is used to warn other trains or others of presence of trains.

1.3 ORGANIZATION OF THE THESIS

Giving the details of the organization of this thesis report, chapter 2 will discuss about the research problems studied in details and also review current literatures on using heuristics optimization approaches on train dispatching problems or scheduling problems. While chapter 3 give the project aims and goals, specifying the required problems to be solve. Chapter 4 will give details of the simulated problems used and proposed heuristics approaches used in solving the problems. Chapter 5 will give the results and analysis of the proposed heuristic approaches used in solving the scheduling problems. While chapter 6, discusses the conclusions and recommendation for future research.

CHAPTER 2

PROBLEMS DESCRIPTION AND LITERATURE REVIEW

2.1 PROBLEM DESCRIPTION

The train dispatcher dispatches trains within the rail network according to the dispatching plans scheduled using the detailed timetable for its daily operations. The train timetable is well planned to strictly meet rail traffic constraints such as track capacities, operational constraints that abide by the rail traffic movement etc. The timetable contains the arrival and departure times for all trains at different stations within the network, it also contains different stations, yards, locomotives, crews etc. The scheduled timetable are well planned to meet customers demands and thereby making the transportation system to be reliable and cost effective.

The train dispatching system may encounter problems due to breakdown of train within the network, this problem will cause the track it occupied not to be available for operation for other trains scheduled to use it and therefore causing conflicts within the network. As a result of this breakdown, conflicts will be experienced and the timetable needs to be modified to correct these problems to meet customers' demands. In order to solve these problems encountered, the train dispatcher will have to reschedule the timetable to eliminate the conflicts encountered within the network and make changes to the original timetable to restore normal traffic movements. These problems are called *train dispatching problems* or *train scheduling problems* and aims to determine detailed train movements and timetables over a rail network using operational constraints and restrictions to minimize trains from deviating from planned schedule [4]. Solving these problems with different heuristic approaches is needed to help the train dispatcher choose solutions which are optimal or has reduced cost of solving the problems and making it possible to meet customers' demands.

2.1.1 RAILROAD TRAFFIC AND REQUIREMENT RULES

For possible movement of trains within the rail network, some certain rules are needed to make the journey smooth and avoiding accidents. Some of these main rules are:

- There must exist all stations specified within the rail network and also tracks connecting the stations together.
- For a train to take a journey within the network, the train should not depart before the departure time as specified by the planned timetable.
- All trains within the network should not exceed the maximum speed limit or maximum safe speed specified for it to use on any track during its journey.
- The weight of trains specified to run on any track within the network must not exceed the maximum allowable weight resistance of that track etc.

Traffic rules that guarantee train crossing operations are needed to ensure that all trains crossing and overtakes happen at intermediate stations or sidings without problems or accidents for smooth operation of the system, rules that helps on these are [2]:

- **Crossing constraints**: Two trains in opposite direction must not use the same one-way track simultaneously, and crossing of two trains can be done only on a two-way track and at stations where one of the two trains has been detoured from the main track.

- **Expedition time constraints**: Time needed for a detoured train to be back on main track and leave the station must exist.

- **Reception time constraints**: Time needed to detour a train from the main track for easy crossing and overtaking to take place must exist.

The diagram below illustrates the train crossing in relation with possible constraints [2]:

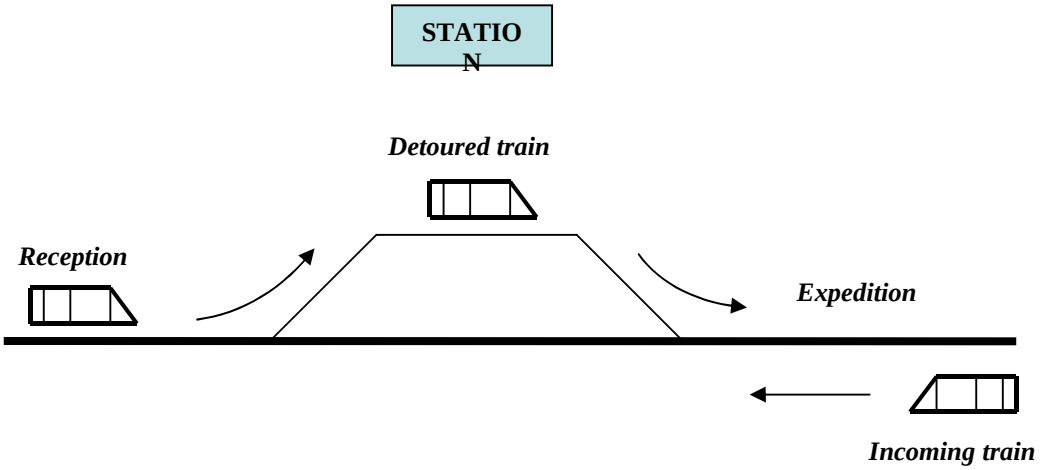

Figure 2.1: Showing train crossing with related constraints

2.1.2 SCHEDULING AND RESCHEDULING

The scheduler is used to generate feasible timetable for the railroad system. It allocates departure and arrival times for all trains within the network and also tracks needed for their journeys. In order to build feasible timetable, scheduling rules needs to be strictly followed to avoid conflicts in the timetable. Some of the major scheduling constraints used are [5][6][3]:

- **Speed constraints**: Trains within the network should not exceed the maximum allowed or maximum safe speed specified for it to use on any track during their journey.

- **Station entrance constraints**: There must exist fixed time interval between two trains arriving a stations using same track line.

- **Weight constraints**: The weight of trains specified to run on any track within the network must not exceed the maximum allowed weight resistance of that track.

Timetable conflicts occur when the trains within the network does not abide by the scheduling rules, possible causes of these problems can be breakdown of trains or stochastic events. These problems needs to be solved to have a reliable timetable to deal with the problems encountered. Possible timetable conflicts are [3]:

- **Trains meeting conflict**: This occurs when trains moving in opposite direction on the same track are about to pass the same fixed point. One major way of solving this problem is using signal systems. The signal system of the railway system are used in giving priority to one of the trains to use the track at a particular time while the other wait for the track to be free before using it.

- **Station capacity conflict**: All stations within the railway system have limited number of tracks for trains to use for parking, if a train needs to use the track for sometime and all available tracks in the stations are occupied, this problem is said to be station capacity conflict.

- **Trains headway conflict**: This conflict occurs when trains within the network moving in the same direction and using the same track are heading to their destination but the

headway time between the trains are less than the required time needed. This will cause problem within the network and needs to be solve to avoid accidents.

In solving the train dispatching problems, a tool is needed to help the dispatcher in solving these problems. Different solutions may be available but the one with fast methods or approaches and cost effective is needed to have a reliable railroad system. When a feasible solution is found, modification of the timetable with the new one from the solution technique used is known as rescheduling i.e. modification of the original timetable.

2.1.3 RAILROAD TRAFFIC PUNCTUALITY AND RELIABILITY

Customers depend on the planned train timetable to plan their journey and expect the schedule to be effective as planned. Average delays and observed punctuality are used in evaluating reliability of railroad system. Punctuality of trains is mostly used to measure reliability of a railroad system (Schaafsma, 2001). This is used to calculate the percentage of trains arriving within a certain number of minutes from the planned arrival time, likewise the departure punctuality can be measured too [1].

2.2 LITERATURE REVIEW

In finding solutions to train dispatching problems, lots of work has been done in the past to finding optimal solutions to possible problems the train dispatcher may encounter. These problems are complex and are NP-complete [8] [7]. Some of the previous work done on train dispatching problems will be reviewed to shed more light on the results achieved so far.

2.2.1 OPERATIONAL RESEARCH

Some of the previous work done on train dispatching problems has been able to assist in trying to solve problems faced by train dispatchers optimally. One of the earliest work was done by Szpigel (1973) [10], he used linear programming model in determining the best crossing and overtaking positions for trains on a single-line track using fixed velocities and departure times of the trains.

Later, R. L. Sauder and W. Westerman (1983) [9], used a computer-aided dispatching system for generating effective train schedules on single-line track using branch and bound algorithm.

Higgins et al. (1996 and 1997) [11], formulated a mixed integer linear problem with the aims of minimizing the total weighted travel times while studying the problems of dispatching

freight trains on a single-line track. He first proposed a branch and bound method in solving the problem, and later used local search heuristics, tabu search, genetic algorithms and hybrid algorithms.

Cai et al. (1998) [12], employed greedy approach (heuristic method) in solving dispatching problems for timetabling and dispatching trains on single line track. This approach uses greedy local criterion in selecting the train to go through the conflict segment.

Sahin (1999) [8], proposed heuristic algorithm for solving conflicts encountered by trains on single-line track that appear in time. This algorithm decides which train to be delay when it compares the future conflicts ahead and time of arrival of each train.

More work was done by G. Sahin, R. Ahuja and C. Cunha [4], they proposed new approaches to solving train dispatching problem. They proposed a new integer programming formulation for dispatching problem based on space-time network by using heuristic algorithms to solve it. They considered some realistic constraints that have not been previously considered, such as maximum allowed delays.

Several works done on train dispatching problem has been able to assist in achieving some success most considered single line tracks and small congested problems but double line tracks are considered too in this project work and will consider networks that looks much like a real world timetable.

CHAPTER 3

PROJECT GOALS

The aim of this thesis work is to have an adaptive and general tool to help the dispatcher find some good schedules in case of unexpected delays. And the goal is to try to have some heuristic algorithms that can provide solutions that will be as much as possible closer to better solution or even an optimal one.

CHAPTER 4

IMPLEMENTATION

4.1 MODEL DESCRIPTION

The model considered deals with three types of train within the network such as high and medium speed trains (e.g. X2000 and Intercity trains in Swedish railway system) for passengers transportation and also freight trains for goods. The speeds of the trains considered are assumed to be fixed for simplicity and also tracks used by each train meet its physical constraints and track type for each train. The model deals with single and double line tracks which are two directional tracks. Passing points such as switches, sidings or crossovers are considered as stations for simplicity. And the timetable conflict or stochastic events considered on the model are train departure delay, track block and train's broken down.

The information about the model of the rail system and stochastic events considered are stored in the ***timetable.xml***, ***trains.xml***, ***network.xml*** and ***events.xml***. These files are input files used by the model to simulate and form a feasible timetable and also solve possible conflicts that may arise due to stochastic events by not violating the traffic rules for safety purpose. The input files are in XML format. This file format is called ***Extended Markup Language*** and is a way of creating standard information format and share the data on the web. This file format is flexible, extensible, and reusable and can be used on the World Wide Web (i.e. internet). The input files format and descriptions are given below:

4.1.1 TRAINS.XML

This file contains physical specifications for each trains used within the system. These physical specifications for each ***train_types*** are:

- ***type*****:** This declares the types of trains used within the rail system such as X2000, Intercity and freight train.
- ***length*****:** This gives the length of each train within the rail network and is measured in meters.
- ***maxspeed*****:** This gives the maximum speed limit the train can use within the network and are measured in km/hr.

- ***weight*:** This specifies the weight of each train and is measured in tons.
- ***power*:** This gives the consumption power of each train within the network and measured in kw/hr.

The syntax of the ***timetable.xml*** is given below [3]:

```
<?xml version="1.0" ?>
<!ELEMENT trains (traintype+)>
 <!ELEMENT traintype (maxspeed, weight, length, power)>
  <!ATTLIST traintype type CDATA #REQUIRED>
  <!ELEMENT maxspeed (#PCDATA)>
  <!ELEMENT weight (#PCDATA)>
  <!ELEMENT length (#PCDATA)>
  <!ELEMENT power (#PCDATA)>
```

4.1.2 TIMETABLE.XML

This gives the scheduled feasible timetable for one complete day. It contains information about the journey. It includes the arrival and departure times for each train at each of the stations they are expected to be during their journey. The description of ***timetable.xml*** is given below:

Timetable: This is the root of the file and it has one sub element called *train*.

Train: This represents a journey and has sub elements such as:

- ***train number*:** This gives the identification number for the train type that is to engage in a journey.
- ***train_type*:** This gives the type of the train to engage in the journey.
- ***departure_station*:** This gives the name of the station from which the train will depart.
- ***departure_hour and departure_minute*:** This specifies the departure time of a train at a specific station.
- ***track_id*:** This gives the track identification number of track to be use by a train from one station to the next.
- ***station*:** This gives the information about stations between the source and destination stations of a journey. It has the following sub information:
    - ***name*:** Gives the name of the station.

- ***arrival_hour and arrival minute*:** Gives the arrival time of a train to the station involve.
- ***departure hour and departure minute*:** Gives the departure time of a train from the station.
- ***track_id*:** This gives the identification name for the track to be used from the station to the next.

- ***destination_station*:** This gives the destination station in which the journey will end.
- ***arrival_hour and arrival_minute*:** This specifies the arrival time to the destination station.

The syntax of the ***timetable.xml*** is given below [3]:

```
<?xml version="1.0" ?>
<!ELEMENT timetable (train+)>
 <!ELEMENT train (train_type, departure_station, departure_hour, departure_minute,
                  track_id, station*, destination_station, arrive_hour, arrive_minute)>
 <!ATTLIST train number CDATA #REQUIRED>
 <!ELEMENT train_type (#PCDATA)>
 <!ELEMENT departure_station (#PCDATA)>
 <!ELEMENT departure_hour (#PCDATA)>
 <!ELEMENT departure_minute (#PCDATA)>
 <!ELEMENT track_id (#PCDATA)>
 <!ELEMENT station (name, arrive_hour, arrive_minute, departure_hour,
departure_minute, track_id)>
   <!ELEMENT name (#PCDATA)>
   <!ELEMENT arrive_hour (#PCDATA)>
   <!ELEMENT arrive_minute (#PCDATA)>
 <!ELEMENT destination_station (#PCDATA)>
```

4.1.3 NETWORK.XML

This gives the information about the network stations. It contains information about the stations, yards and tracks between stations and also physical specification of each track within the network.

Network: This is the root element of the file and has two sub elements which are station and connection. The description of the ***network.xml*** is given below:

- ***station*:** This gives the information about the stations that exists within the network, these includes the station's names and their yard capacities:
  - ***name*:** Names of the station.
  - ***yard_capacity*:** Gives an integer number representing the yard capacity.
- ***connection*:** This gives information about the tracks that exists between two stations. This has the following elements:
  - ***station1*:** This specifies one of the two stations connected to the track.
  - ***station2*:** This specifies the other track connected to the track.
  - ***track*:** This gives the physical specification attributed to each track and has the following elements:
    - ***name*:** This is one of the attributes of the tracks used to distinguish tracks within the network. This gives other information about the tracks like:
      a) ***track_length*:** Indicates the length of the track specified in integer and is measured in km.
      b) ***maximum_weight*:** Indicate the maximum weight of the train and are measured in tons.
      c) ***maximum_speed*:** Indicates the maximum allowable speed of the train and are measured in km/hr.

The syntax of the ***timetable.xml*** is given below [3]:

```
<?xml version="1.0" ?>
<!ELEMENT network (station+, connection+)>
 <!ELEMENT station (yard_capacity)>
  <!ATTLIST station name
            (Stockholm|Ludvika|Mora|Uppsala|Falun|Borlange|Linkoping|Nykoping|Kiruna
              |Leksand|Ratvik) #REQUIRED>
  <!ELEMENT yard_capacity (#PCDATA)>
 <!ELEMENT connection (station1, station2, type, track+)>
  <!ELEMENT station1 (#PCDATA)>
  <!ELEMENT station2 (#PCDATA)>
```

<!ELEMENT type (#PCDATA)>

<!ELEMENT track (track_length, maximum-weight, maximum-speed)>

<!ATTLIST track name ID #REQUIRED>

<!ELEMENT track_length (#PCDATA)>

<!ELEMENT maximum-weight (#PCDATA)>

<!ELEMENT maximum-speed (#PCDATA)>

4.1.4 EVENTS.XML

This gives information about the stochastic events used in simulation. These events considered are train departure delay, track blocked and train engine broken down. The file description is given below:

Event: This is the root element of the ***events.xml***, and has the elements described below:

- ***event_ID*:** Gives the number of the event which indicates the type of event involved.
- ***train_ID*:** This specifies the train involved in the event to be considered.
- ***location type*:** This indicates the location where the event occurred which can either be track or station.
- ***Dis_to_start*:** This is used only when the event is track block, it gives the distance between the source to the point where the event occurred.
- ***Origin_Hour and Origin_Minute*:** This indicates the time in which the stochastic event occurred.
- ***DelayTime*:** This indicates the assumed time needed to solve the stochastic events.
- ***Time_To_SURE*:** Indicates the time to be sure period to solve the problem.

The syntax of the ***timetable.xml*** is given below [3]:

<!ELEMENT events (event+)>

<!ELEMENT event (type, train_ID?, location, Dis_to_start?, Origin_Hour, Origin_Minute, Delay Time, Time_To_SURE)>

<!ATTLIST event ID CDATA #REQUIRED>

<!ELEMENT type (#PCDATA)>

<!ELEMENT train_ID (#PCDATA)>

<!ELEMENT location (#PCDATA)>

<!ATTLIST location type CDATA #REQUIRED>

<!ELEMENT Dis_to_start (#PCDATA)>

<!ELEMENT Origin_Hour (#PCDATA)>

<!ELEMENT Origin_Minute (#PCDATA)>

<!ELEMENT Delay Time (#PCDATA)>

<!ELEMENT Time_To_SURE (#PCDATA)>

The figure 4.1 below gives the model platform describing the links between the XML files and the simulator used to build new feasible timetable.

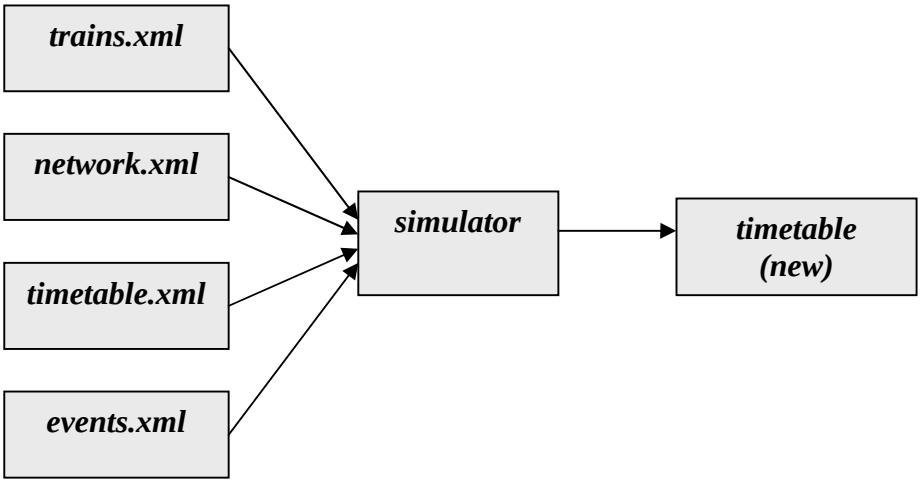

Figure 4.1: Shows the links between the simulator and the input files.

4.2 THE SIMULATION PROCESS

The simulation model used is designed to help the dispatcher in tracking problems in the system and solve them by generating a new timetable as a result of modifying the original one. When conflicts occur within the timetable as a result of delays, the simulator starts and will continue to simulate until all conflicts are solved and a new feasible timetable is generated. The simulator uses the event list and the schedule list in its process.

Events list is used to handle the simulation process and has to be generated for the simulator to start. All events that occur at a particular time interval are inserted in the event list and are periodically updated to correspond to the sequence of events according to the occurrence time in ascending order. From the event list been built, it has a head event which is the event that occur at the earlier occurrence time. The simulator chooses the head event from the event list and executes it using the appropriate subroutine to solve the problem, and then picks the next event in sequence if available from the schedule list and adds it to the event list, then free it and repeat the same procedure. The simulation process end as soon as the event list is empty.
When the simulation starts, the simulation clock starts from one event to another by looking up the scheduled events in the event list, there by having two possibilities which are [3]:

- *If occurrence time of the head event is equal to the simulation time:*
  due to this, it will process this event for simulation.
- *The occurrence time of the event id is not equal to the simulation time:*
  then increase the simulation time by the time spot and check again.

In order to solve the dispatching problems, the event type is used by the simulator to choose the type of subroutine to use. Possible event types are normal or abnormal and stochastic events.

4.2.1 SIMULATION PROCESS WITH SCHEDULING AND RESCHEDULING

Scheduling as previously explained is the building of railroad timetable for the train dispatcher and is done offline. In order to have a reliable timetable, the scheduling rules have to be strictly followed in order to avoid conflicts in the timetable due to unsatisfied scheduling rules. But rescheduling is done online and is used when trains in the rail network are affected by stochastic events or timetable conflicts which cause conflicts or delays. The diagram below

illustrate the shows the relationship of simulator with Scheduling and rescheduling is shown in the figure 4.3 below [3]:

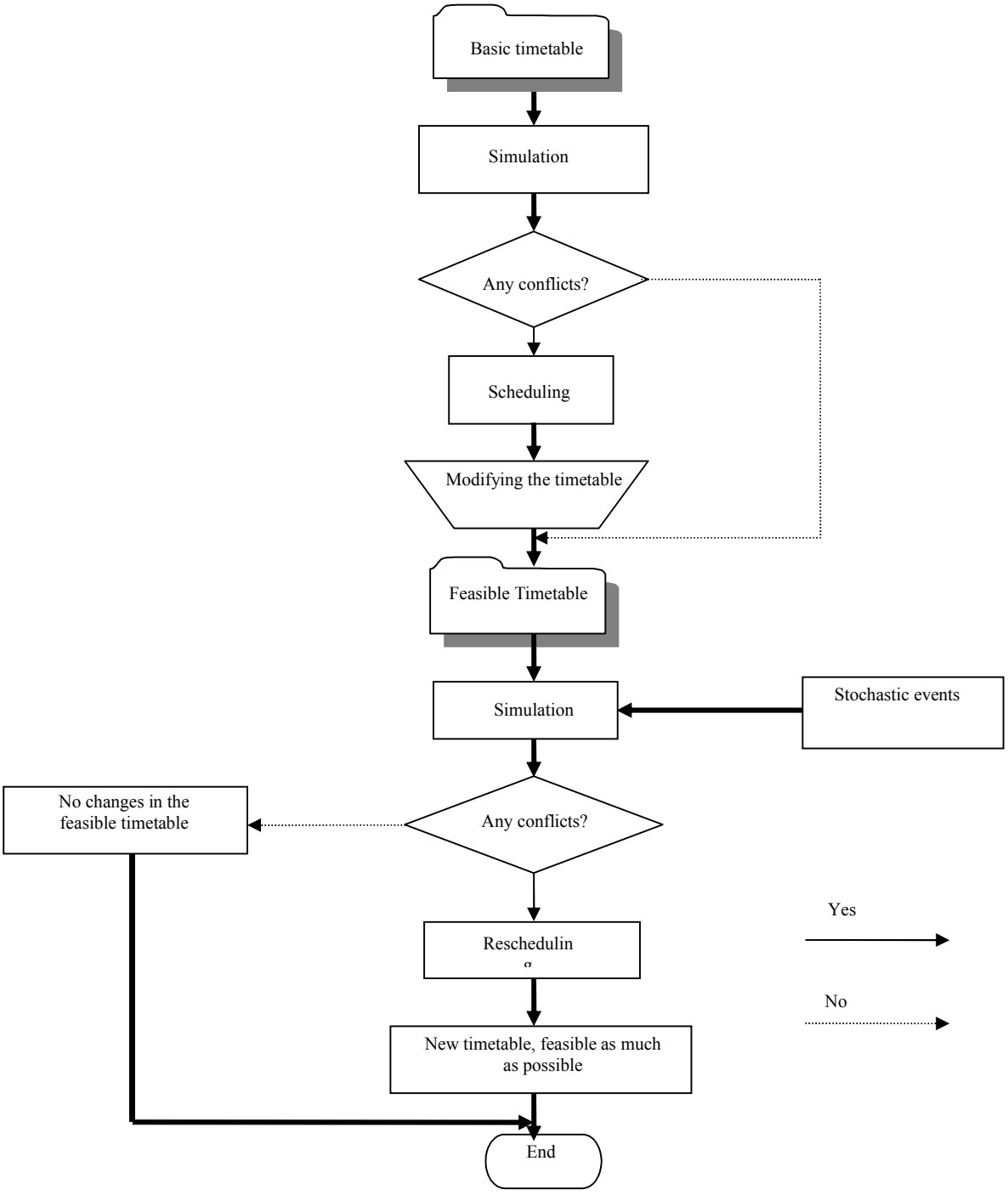

Figure 4.2: Simulation process with scheduling and rescheduling

4.3 PROPOSED HEURISTICS ALGORITHMS

In solving conflicts faced by the train dispatchers, I proposed two heuristics algorithms that can help in finding reliable and cost effective solutions as fast as possible to solve possible problems. There exist different steps that can help in finding solutions to solving trains conflicts, but from all possible options, the best solution that can give less cost and with feasible solution will be highly required. These heuristics algorithms proposed will search for better paths possible or steps to achieving the goals of this work.

DEFINITIONS

Consider a railroad network with different stations and trains involved in daily operations of a railroad system as described in the input files previously, using single and double line tracks. Let $\boldsymbol{T}$ represent set of trains and $\boldsymbol{S}$ representing set of stations within the network system and for simplicity passing points such as sidings, switches and crossovers are considered as stations, we have:

$\boldsymbol{S = \{S_1, S_2, S_3 \ldots S_n\}}$, representing set of stations, and also
$\boldsymbol{T} = \{\boldsymbol{T_1, T_2, T_3 \ldots . T_n}\}$, represent set of trains within the railway network.

Each train can travel within the network from one station and arrive at another using their departure and arrival times specified in the timetable i.e. say from station $\boldsymbol{S_1}$ to $\boldsymbol{S_2}$ (where $\boldsymbol{S_1 \neq S_2}$). This means a train $\boldsymbol{T_i}$ (where $\boldsymbol{i = 1, 2, 3 \ldots n}$) can move from one station to another in any direction depending on the schedule being drawn out using the trains timetable. The train dispatcher contains well detailed timetable as drawn out in the ***timetable.xml*** which makes it easy for trains to move within the network using the departure and arrival times for each train. The dispatcher can be faced with conflicts as a result of stochastic events such as **train departure delay**, **track block** and **train's engine broken**. Two heuristics algorithms are been proposed in helping the train dispatcher to make decisions in solving conflicts between trains and to give fast and cost effective solution in less time.

4.3.1 THE GREEDY HEURISTICS APPROACH (ALGORITHM 1)

The idea of this algorithm is based on using selection rules in making decisions to choose the train to be delayed for the other to travel during the time of conflicts. The algorithm works in such a way that when there is a problem within the network, it checks for the time needed to

solve the problem encounter and then delay the train involved and its successive trips with the time ***t*** required. It then checks to see if there are conflicts along the journey if the train continues after. If there are none, it will update the solution list and make necessary changes to the timetable temporarily for the rest of the day's journey. Otherwise, it will save the conflict encountered in the conflict list, and therefore make use of the selection criteria to decide which train to be delayed for the other to continue its journey (i.e. check to see if the conflict can be solve without violating scheduling rules). It will continue to solve further conflicts in such manner and keep records of conflicts encountered in the conflict list. When a solution is finally found, it will update the solution list with the solution found and also temporarily make changes to the timetable for the rest of the day's trip.

Consider a network of trains as previously defined above, suppose after delaying the train ***T*** that broke down with time ***t*** required to solve it, the algorithm will delay all its successive trips with the delay time ***t*** and then check for any future conflict that may arise because of these changes. Using figure 4.4 below, suppose there is a conflict at time t_i involving train T_1 and T_2, the algorithm will use of the selection rule on the conflict node to decide which train is to be delayed for better solution. Delaying either T_1 or T_2 can give more conflicts along the path chosen, with the selection criteria, the chosen leaf node will be retained while the other leaf will be deleted. The same procedure will be followed until a solution is achieved.

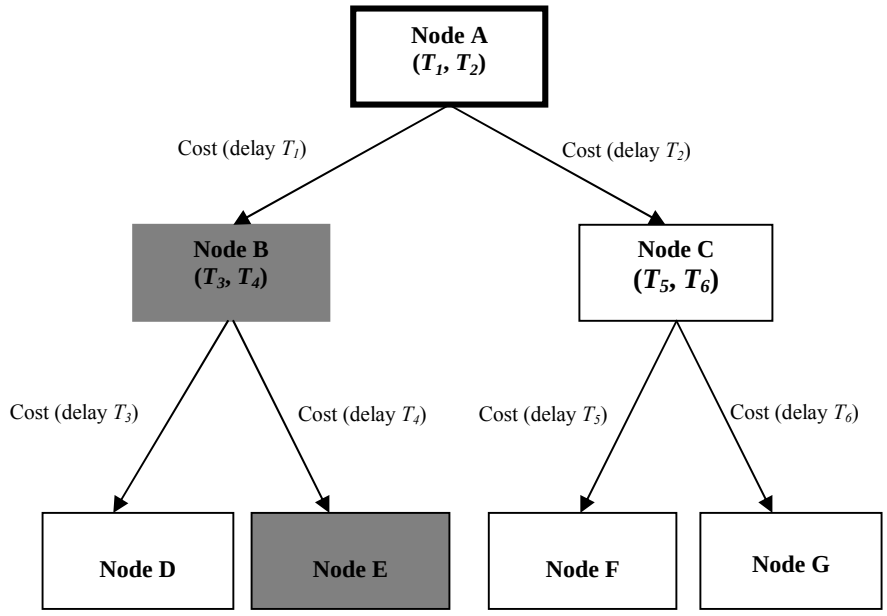

Figure 4.3: Showing the conflict tree and possible path chosen for possible solution.

The selection criteria are set of rules employed to ease decision making in order to achieve our aims of getting results very close to the best solution or optimal solution from the algorithm. Explaining the selection rules used in deciding the train to be delayed during conflicts using the definition below:

Suppose,

T_1, T_2 : represents trains involved in conflict during the process.

cT_1, cT_2 : represents ***cost*** of delaying the trains T_1 and T_2 respectively, the ***cost*** means the time involved in delaying train and all its successive trips.

Defining the selection rules that decides which of the train to be delayed are given below:

if *(* $cT_1 \leq cT_2$ *)* ***then*** *delay train* T_1 ***; else*** *delay train* T_2 *;* ***end if***

The proposed greedy heuristic approach (***algorithm 1***) is given below:

Suppose that,

T*,* T_1 *and* T_2 *are assumed trains involve.*

solution_list ( ) *is a memory space to save solution found and the path of the solution found.*

event_list ( ) *is the event list.*

conflict_list *is a memory space where conflicts found are stored.*

any_conflict (T) *is the subprogram to check if there are conflicts available.*

Procedure *heuristic 1;*

{

int ***k*** *= 0; // counter for the number of conflict encounter.*

int ***current_cost*** *= 0; // is used in getting cost of delays along the path visited.*

int ***cost_delay ( )****; // cost of delaying train involved.*

initialise the ***solution_list ( )****;*

initialise the ***conflict_list ( )****;*

using the current ***event_list ( )****;*

delay the train ***T*** *involved and all its successive trips with the required time* ***t*** *needed;*

current_cost *= cost of delaying* ***T*** *and all successive trips of it;*

any_conflict (T)*;* *// to check if there are conflicts.*

*while (**any_conflict (T)** = = True) do*

{

k = k +1; // increasing the conflict counter

*get the trains involve (e.g. T_1 and T_2), and save the conflict in **conflict_list ( )**;*

for** each train involved (e.g. T_1 and T_2) **do

{

find the required delay time for train T_i with respect to the conflict;

***cost_delay (T_i)** ← get the cost of delaying T_i and its successive trips;*

}

*if ((**cost_delay (T_1)** < **(cost_delay (T_2))** **OR** ($dT_1 < dT_2$)) then*

{

delay successive trips of train T_1 involve;

*update the **solution_list ( )** with changes made;*

***current_cost = current_cost + cost_delay (T_1)**;*

***$T \leftarrow T_1$** ;*

}

else

{

delay successive trips of train T_2 involve;

*update the **solution_list ( )** with changes made;*

***current_cost = current_cost + cost_delay (T_2)**;*

***$T \leftarrow T_2$** ;*

}

end if

***any_conflict (T)**; // to check if there is any conflict*

}

*if (**any_conflict (T)** = = false) then*

{

print ('solution found');

*use the **solution_list ( )** to change the timetable temporarily for the day's trips;*

}

end if

} *// end of algorithm.*

4.3.2 THE BEST-FIRST SEARCH HEURISTIC APPROACH (ALGORITHM 2)

This algorithm is based on the approach of taking very importantly the heuristic values of the leaf node of each node encountered (i.e. conflict node) during its search and using these to search for the best path to take in solving the problems encountered. In solving conflicts, the algorithm checks for the heuristic value of each possible path, then search and explore the path with less cost first. Each possible path it encountered are stored with their heuristic value in order to make use of them during the searching. When a path with less cost is explored, and during the search along that path it noticed that the present cost is more than one or previous possible paths stored, it will backtrack and explore the path with current less cost of search space. Note, each level of the search space nodes involved in finding solutions will be considered but will consider the one with less cost first before exploring other ones (i.e. searches the best path first).

Consider figure 4.5 below, assuming there is problem along a rail network, the algorithm will delay the train involved with the necessary time needed to solve the problem and then delay all the successive trips of the train involve. As a result of these changes conflicts may occur with other trains within the network, thereby making the timetable not abiding with the traffic rules. Assuming because of the changes done a headway conflict between trains is noticed in the timetable, the algorithm will then try to find possible solution to the problem encountered by the dispatcher. If a headway conflict of trains at ***node A*** is encountered, in trying to solve this problem will result into conflicts between trains T_1 and T_2. Delaying the two trains separately with the required time needed for each will result in ***level 2*** of the search space. But the cost of delaying trains T_1 and T_2 will be compared and stored, and then the path with less cost is explored first to try to solve the problem (i.e. if the cost of delaying T_1 is less than delaying T_2, path along T_1 is considered first and then will be explore for solution). If after exploring this path for solution and it was discovered that other conflicts are been encountered and the costs of delaying trains involved are greater than delaying T_2 in the previous conflict, the path for T_2 will then be considered as the best path to search for better solution. Due to this method of search, each level of the tree will be considered. Note, if during the search a solution is found in say ***node E*** but the cost is greater than exploring other paths of the search

tree, the algorithm will consider those paths in ascending order for possible lesser cost of solution.

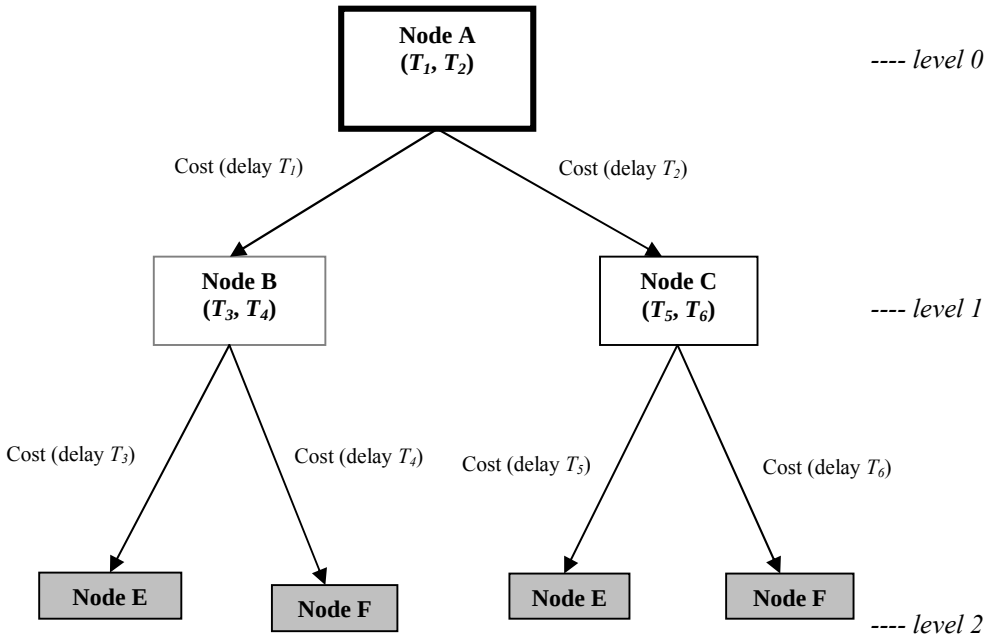

Figure 4.4: Shows the nodes and the costs of delaying the trains involves along the path.

The Best-first search heuristic approach (***algorithm 2***) is given below:

Suppose that,

T, T_1 *and* ***T_2*** *are assumed trains involve.*

solution_path [ ] *a temporary space to save possible paths to solution sorted in ascending order.*

event_list [ ] *is the event list.*

conflict_list [ ] *is a memory space where conflicts found are stored.*

any_conflict (T) *is the subprogram to check if there are conflicts available.*

This algorithm is given below:

Procedure *Heuristic 2;*

{

boolean ***solution found****;*

int ***c*** *= 0; // counter for the number of conflict encounter.*

int ***current_cost*** *= 0; // is used in getting cost of delays along the path visited.*

int ***cost_delay****; //used in getting cost of delaying a train.*

int ***last_best_cost****; // is the cost of the best path in the solution list.*

initialise the ***solution_path [ ]****;*

initialise the ***conflict_list [ ]****;*

using the ***event_list ( )****;*

delay the train ***T*** *involve and all its successive trips with the required time* ***t*** *needed;*

current_cost (T) *←* *cost of delaying* ***T*** *and successive trips of it;*

any_conflict (T)*;* *// to check if there is any conflict*

*if (****any_conflict (T)*** *= = True) then*

{

update the ***conflict_list ( )*** *with current conflicts encountered;*

c *=* ***c*** *+ 1;* *// increasing the conflict counter.*

*while ((****any_conflict (T)*** *= =* ***True****) AND (****feasible solution not found****)) do*

{

get the trains involve (e.g. T_1 *and* T_2*);*

for *each train involved (e.g.* T_1 *and* T_2*)* ***do***

{

cost_delay (T_i ***)*** *←* *get the cost of delaying* T_i *and successive trips with respect to the conflict;*

current_cost (T_i ***)*** *←* *adding* ***cost_delay (*** T_i ***)*** *and previous* ***current_cost****;*

update the ***solution_path [ ]*** *with the cost involved for the train and sort in ascending order;*

any_conflict (T_i ***)****;* *// check if there is any conflict as a result*

*if (****any_conflict (*** T_i ***)*** *= = True) then*

{

save the current conflict in the ***conflict_list ( )****;*

c *=* ***c*** *+ 1;* *// increasing the conflict counter*

}

else

save the solution found in the ***solution_path [ ]****;*

end if

}

// now deciding which path to take in finding a feasible solution

select from the ***solution_path [ ]*** *the path with lowest cost;*

delay the train T_i *involved and all its successive trips with the required time* ***t*** *needed;*

update the ***solution_path [ ]*** *with the cost involved for the train and sort in ascending order;*

*if ((****solution found*** *=* ***True****) AND (****solution found*** *has the lowest cost)) then*

{

print ('Optimal Solution has been found');

use the path found to be optimal to make necessary changes to the timetable temporarily and delete all other paths in the ***solution_path [ ]****;*

delete all conflicts in the ***conflict_list [ ]*** *and set the list to* ***NULL****;*

}

else

{

delete all conflicts node and paths cost with higher cost of path in the ***conflict_list [ ]*** *and* ***solution_path [ ]*** *list respectively;*

select the path from the solution list with lowest cost and check if conflict exists, if it exist continue searching;

}

end if

} // end while

} // end if

else

{

print ('Optimal Solution has been found');

use the changes to make temporary changes to the timetable;

}

end if

} // end of algorithm

CHAPTER 5

RESULTS AND ANALYSIS

5.1 RESULTS

In order to know the ability of these heuristics in solving problems as a result of stochastic events which causes conflicts in rail networks, four different sample problems were tested on three different sample network to test the effectiveness of the proposed heuristics and found these algorithms reliable and also cost effective in helping the dispatchers in making decisions on the paths to take in solving the problems. Each network contains different numbers of trains within each network. These networks contain both single and double line tracks for movements of trains from source to their destinations. For each network, the vertical axes represent the time for departure and arrival of trains, the horizontal axes represents the stations, while the lines between the stations represent the trains' routes. The **headway time** considered between trains is 20 minutes, while the **departure arrival interval** is 5 minutes. Sample networks were used to test the ability of the algorithms and results found were reliable in solving the problems, these networks and problems are given below:

Sample Network 1 (***Problem 1***):

Considering the sample network given below in figure 5.1, suppose there is a **single line track** between **Stockholm** and **Ludvika** and also between **Mora** and **Uppsala**, while there is **double line track** between **Ludvika** and **Mora**. The **headway time** between trains is 20 minutes, while the **departure arrival interval** is 5 minutes. The trains involve in the network are T_1, T_2, T_3, T_4, T_5, T_6, and T_7, and these trains have different speed (i.e. speed of ($T_1 = T_2 = T_3$) < T_4 < ($T_6 = T_7$). Assuming there is a problem within the network between **Stockholm** and **Ludvika**, and the train T_3 is to be delay for 30 minutes. This means the departure delay of T_3 will be 30 minutes, then the departure time of the train will be delayed for 30 minutes. The diagrams below shows the network and possible solutions involved in solving the problems that arise after the delaying of train T_3 for the required time using the proposed algorithms. Using the proposed algorithms to solve the problem, both algorithms will immediately delay the departure of T_3 and all its successive trips with 30 minutes, thereby resulting in other conflicts to surface and then continues to solve the problem until a final solution is achieved. Possible solutions from the algorithms are given in the figure 5.2 below:

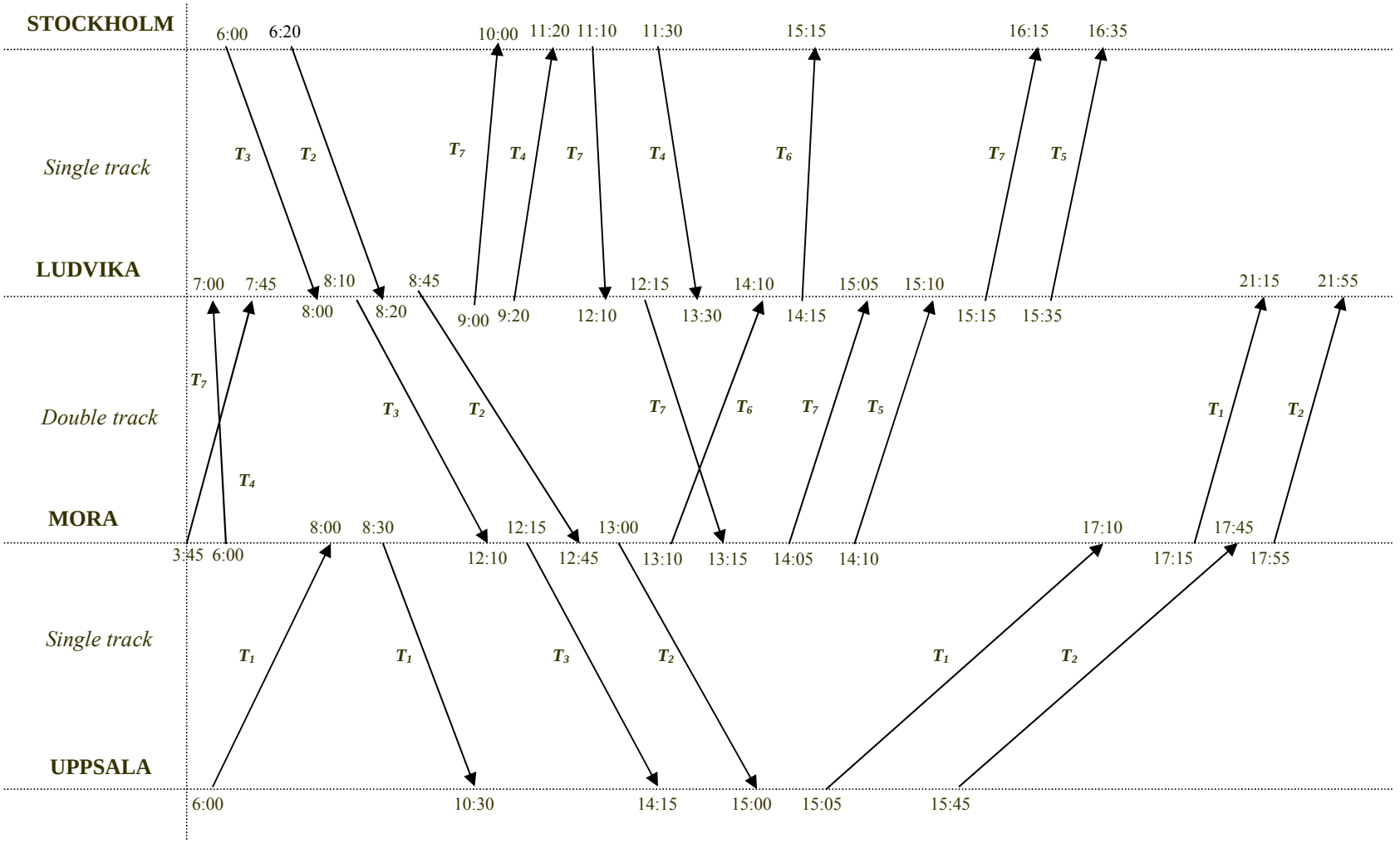

Figure 5.1: Sample Network 1

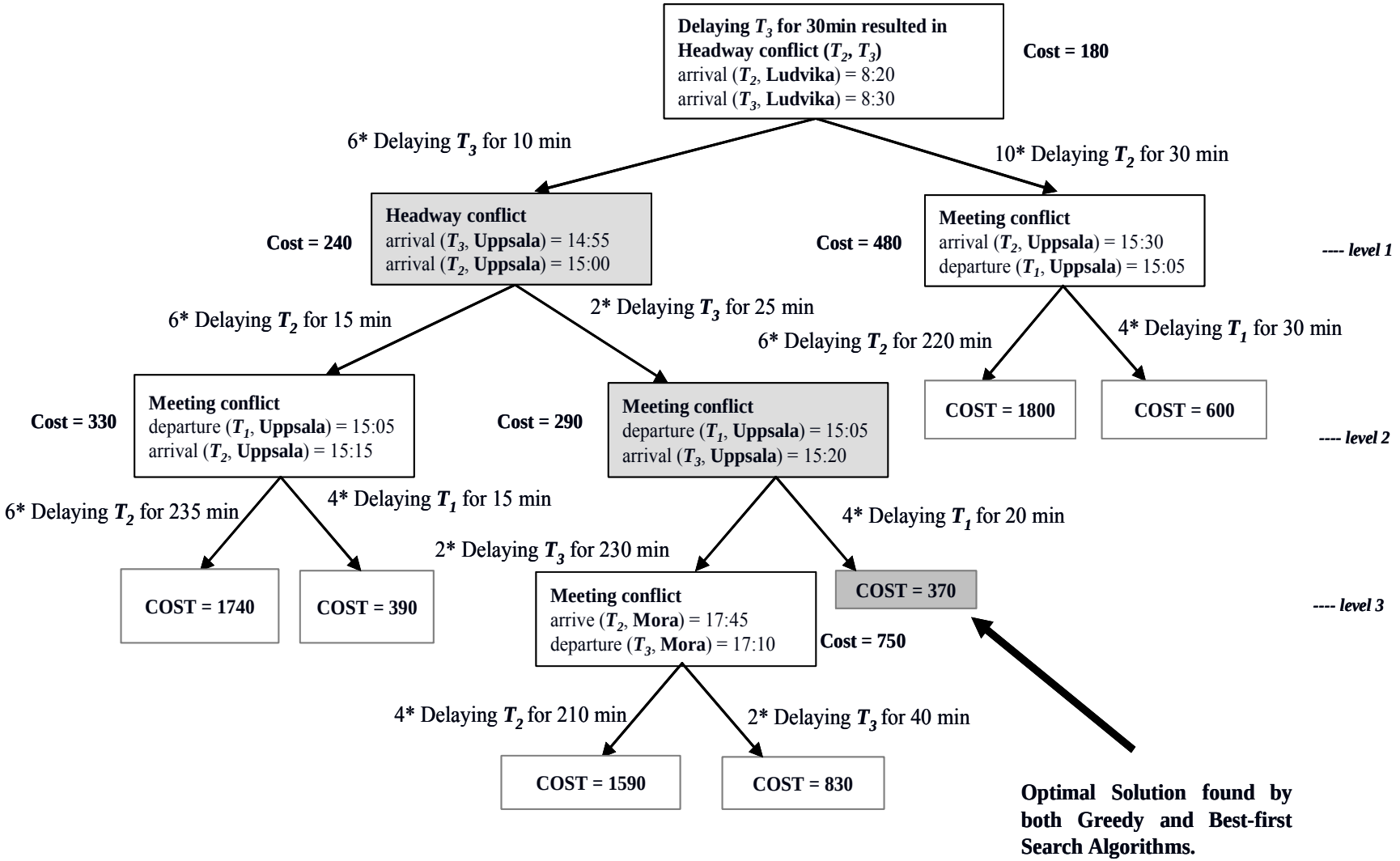

Figure 5.2: Showing possible solutions to problem 1 of Sample Network 1.

Sample Network 2:

Also consider the sample network given below in figure 5.3. Suppose there is a **single line track** between **Falun** and **Hedemora**, **Hedemora** and **Avesta**, **Vasteras** and **Eskiltuna** and between **Avesta** and **Stockholm**, while there is a **double line track** between **Stockholm** and **Vasteras**. The **headway time** between trains is 20 minutes, while the **departure arrival interval** is 5 minutes. The trains involved in the network are T_1, T_2, T_3, T_4, T_5, T_6, T_7, T_8 and T_9 with different speeds (i.e. speed of $T_1 = T_2 = T_3 = T_6 = T_7 = T_8 = T_9 < T_5 < T_4$).

Problem 2: This problem is considered on the network below (figure 5.3). Assuming train T_2 encounter engine breakdown along the journey between **Avesta** and **Hedemora** and the time needed for its repair and possibly back to functioning is 45 minutes, then the train needs to be delayed for that period of time. This means the departure delay of T_2 will be 45 minutes. The diagrams below shows the network and possible solutions involved in solving the problem. Using the proposed algorithms to solve the problem, both algorithms will immediately delay the departure of T_2 and all its successive trips by 45 minutes, thereby resulting in other conflicts to surface and then continues to solve the problems until a final solution is achieved. Possible solutions from the algorithms are given in the figure 5.4 below:

Problem 3*:* This problem is also considered on the network below (figure 5.3). If there is a departure delay problem with train T_1 from Stockholm station and needs to be delayed for 25 minutes, then all successive trips of the train will be delayed for 25 minutes. From the changes made conflicts will occur in the timetable and the algorithms will be needed to solve and generate a possible reliable timetable. Possible solutions from the algorithms are given in the figure 5.5 below:

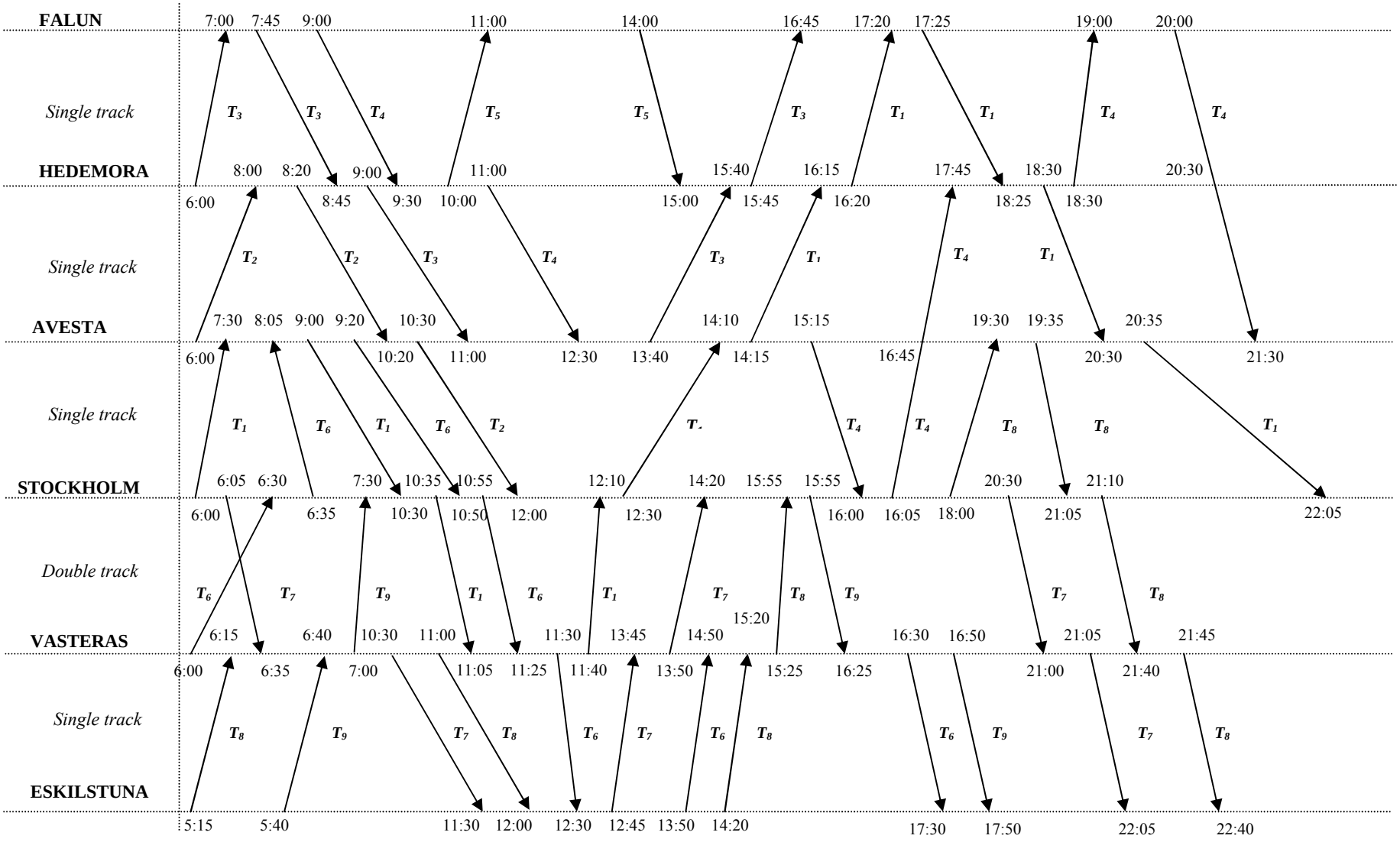

Figure 5.3: Sample Network 2.

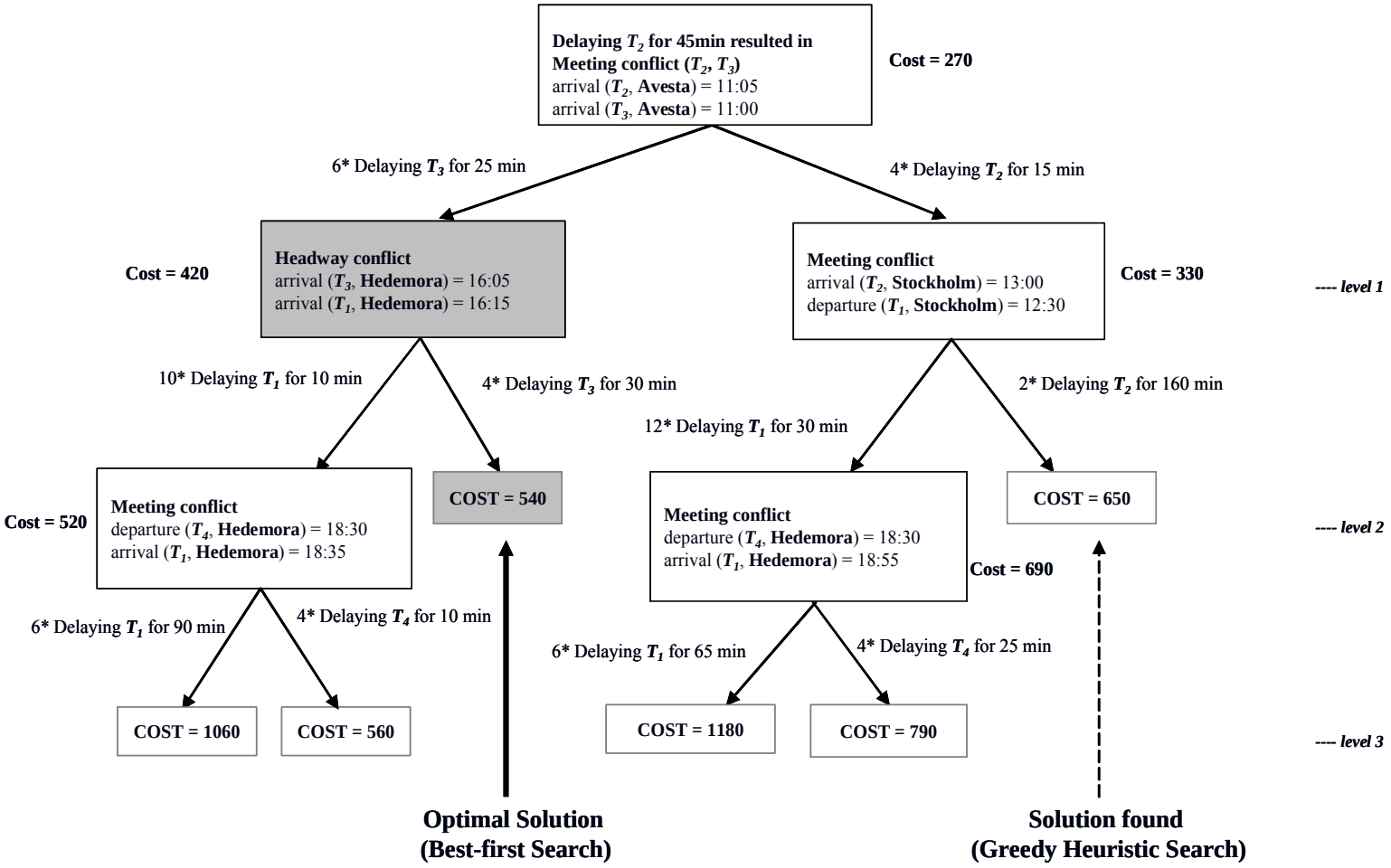

Figure 5.4: Showing possible solutions to the problem 2 on Sample Network 2.

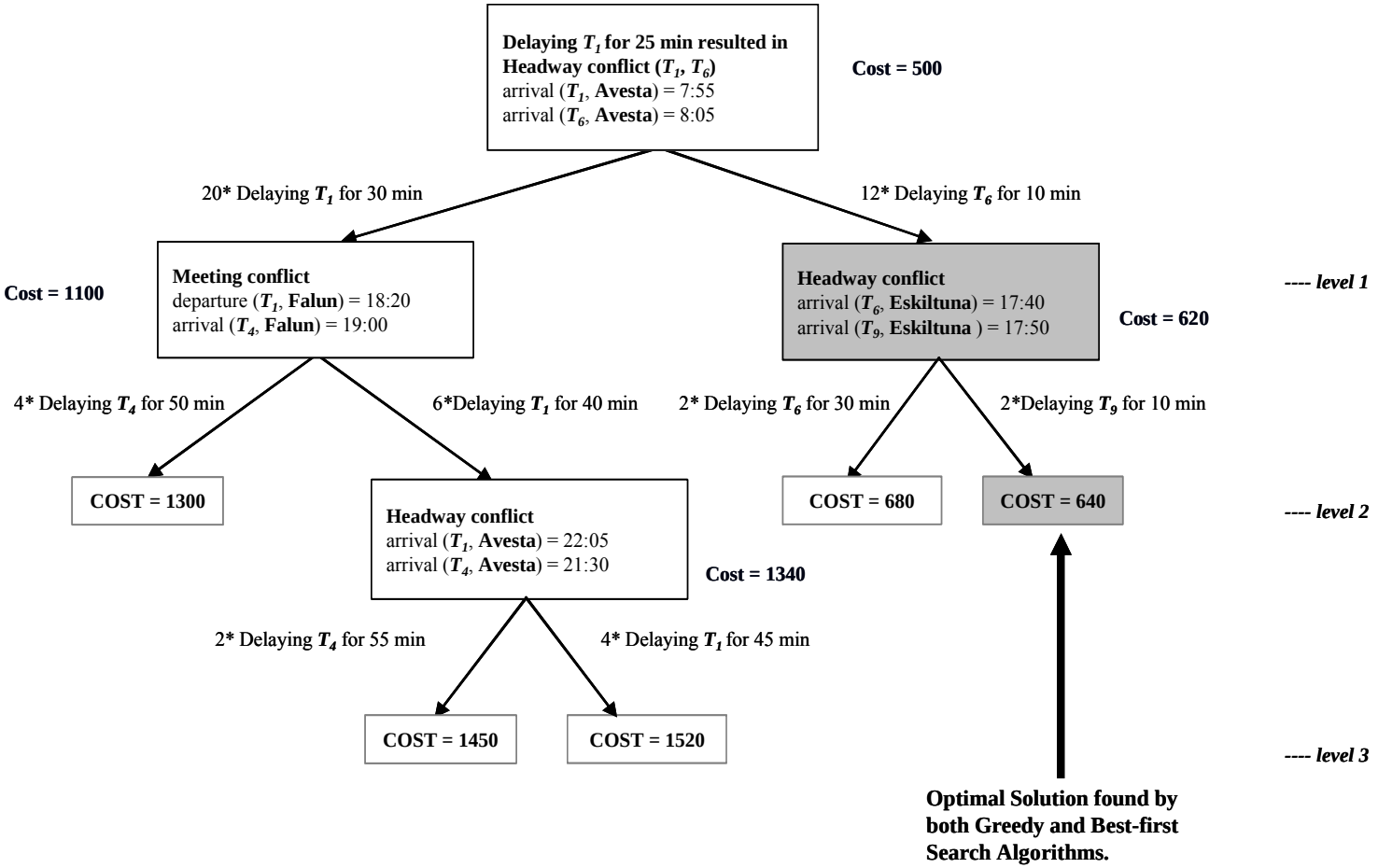

Figure 5.5: Showing possible solutions to the problem 3 on Sample Network 2.

Sample Network 3:

Sample network 3 considered is the network given in figure 5.6 below. As explained previously, the **headway time** between trains is 20 minutes, while **departure arrival interval** between trains is 5 minutes. The trains involved in the network are T_1, T_2, T_3 and T_4. The speed of trains T_2 and T_4 are the same and are faster than for T_1 and T_4 which also having the same speed.

Assuming there is a problem with train T_1 after departing from **Uppsala** and needs to be delayed for 30 minutes to repair the train. The algorithm will delay the train T_1 at time 10:10 for 30 minutes and then have abnormal departure at 10:40, and then all its successive trips will be delay for 30 minutes. The solutions to the problem as achieved by the algorithms are given in figure 5.7 and also worked out solutions from the algorithms are illustrated below:

WORKED EXAMPLE

Using the ***Algorithm 1*** to find solution to this problem:

The algorithm will immediately delay the departure time of T_1 and all its successive trips with 30 minutes:

arrival (T_1, **Mora**) = 10:45 + 30 = 11:15

departure (T_1, **Mora**) = 11:15 + 30 = 11:45

arrival (T_1, **Uppsala**) = 12:00 + 30 = 12:30

departure (T_1, **Uppsala**) = 12:45 + 30 = 13:15

arrival (T_1, **Ludvika**) = 15:00 + 30 = 15:30

departure (T_1, **Ludvika**) = 15:10 + 30 = 15:40

arrival (T_1, **Stockholm**) = 16:40 + 30 = 17:10

Cost of delay = 7 * 30 minutes = 210.

From these changes the algorithm will encounter the first conflict between T_1 and T_3:

Headway Conflict:

arrival (T_1, **Uppsala**) = 12:30

arrival (T_3, **Uppsala**) = 12:35

The algorithm will then check for the train that will give less cost when delayed, it will decide to delay T_3 because it will produced less cost.

Sample Network 3:

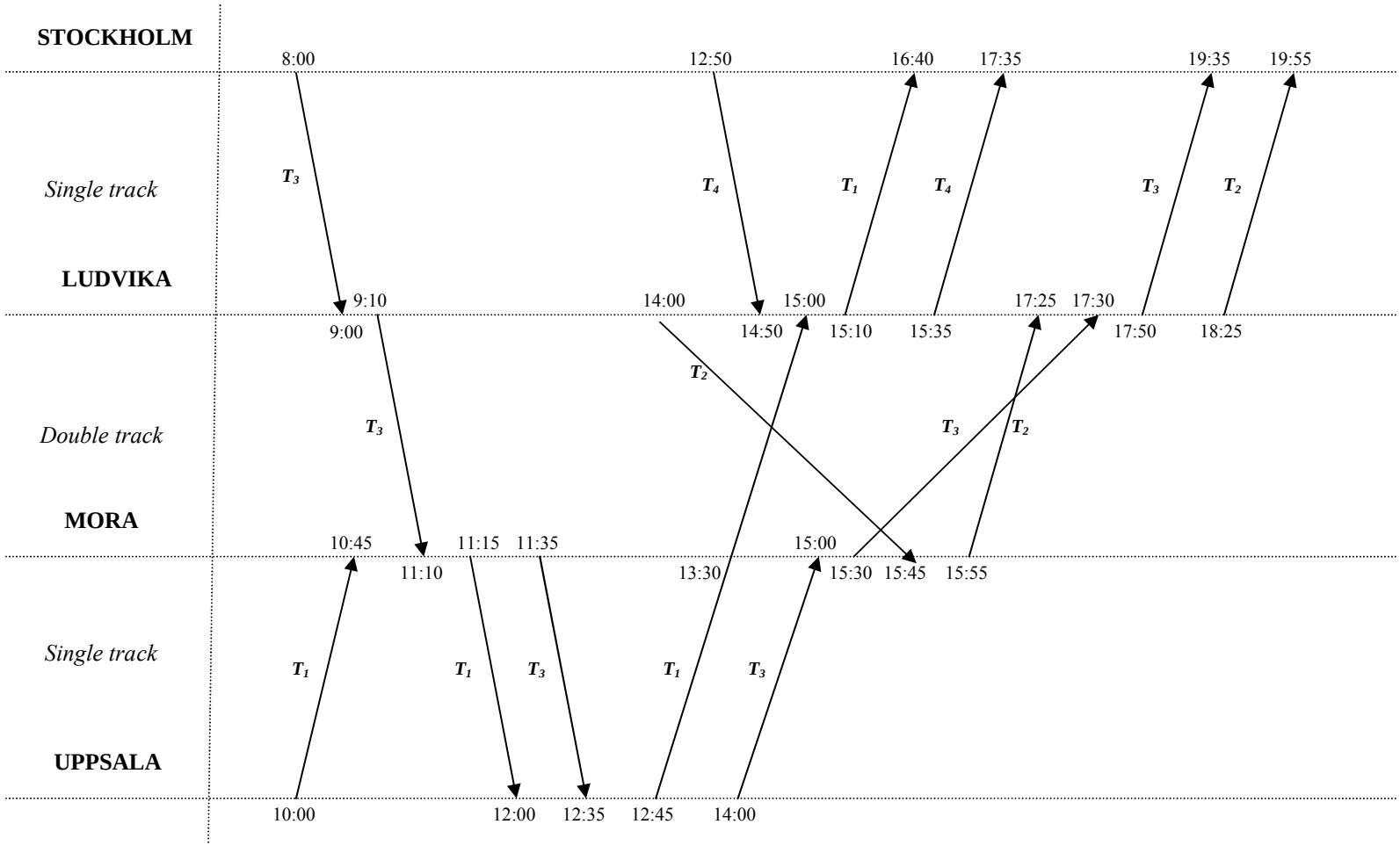

Figure 5.6: Sample Network 3.

Delaying T_3 and all its successive trips for 15 minutes, we have:

departure (T_3, **Mora**) = 11:35 + 15 = 11:50

arrival (T_3, **Uppsala**) = 12:35 + 15 = 12:50

departure (T_3, **Uppsala**) = 14.00 + 15 = 14:15

arrival (T_3, **Mora**) = 15:00 + 15 = 15:15

departure (T_3, **Mora**) = 15:30 + 15 = 15:45

arrival (T_3, **Ludvika**) = 17:30 + 15 = 17:45

departure (T_3, **Ludvika**) = 17:50 + 15 = 18:05

arrival (T_3, **Stockholm**) = 19:35 + 15 = 19:50

Cost of delay = 8 * 15 minutes = 120

Total delay = 210 min for previous delays + 120 min = 330.

The algorithm will make the changes and detect a meeting conflict, and then will save it in the conflict list:

Headway conflict (1b):

arrival (T_3, **Stockholm**) = 19:50

arrival (T_2, **Stockholm**) = 19:55

As previously done, it will delay the train with less cost, therefore it will delay T_2.

Delaying T_2 for 15 minutes, we have:

departure (T_2, **Ludvika**) = 18:25 + 15 = 18:40

arrival (T_2, **Stockholm**) = 19:55 + 15 = 20:10

Cost of delay = 2 * 15 minutes = 30

From these, the algorithm will declare a path to the solution has been found. The total cost of delay is:

Total Cost of Solution using Greedy algorithm:

Final Cost = 210 min for the departure delay of T_1 + 120 min for the delay of T_3 + 30 min for the delay of T_2 = 360.

Note: Comparing this with all possible solutions to the problem shows that the result achieved is an optimal solution.

Using the ***Algorithm 2*** to find solution to this problem:

The algorithm will immediately delay the departure of T_1 and all its successive trips with 30 minutes, we have:

arrival (T_1, **Mora**) = 10:45 + 30 = 11:15

departure (T_1, **Mora**) = 11:15 + 30 = 11:45

arrival (T_1, **Uppsala**) = 12:00 + 30 = 12:30

departure (T_1, **Uppsala**) = 12:45 + 30 = 13:15

arrival (T_1, **Ludvika**) = 15:00 + 30 = 15:30

departure (T_1, **Ludvika**) = 15:10 + 30 = 15:40

arrival (T_1, **Stockholm**) = 16:40 + 30 = 17:10

Cost of delay = 7 * 30 minutes = 210.

From these changes the algorithm will detect a conflict between T_1 and T_3:

Headway Conflict:

arrival (T_1, **Uppsala**) = 12:30

arrival (T_3, **Uppsala**) = 12:35

The algorithm will then store and compare the current cost of delaying each. The algorithm will pick the one with less cost and delay the train involved. Note each path cost is stored and sorted with respect to the cost of delaying the trains involved. Then trying to solve the problem for better solution, we have:

1. The algorithm will detect the headway conflict, and then save it in the conflict list. Then uses possible delay time for each option of *level 1*:

(a) Delaying T_1 for 25 minutes and its successive trips, we have:

departure (T_1, **Mora**) = 11:45 + 25 = 12:10

arrival (T_1, **Uppsala**) = 12:30 + 25 = 12:55

departure (T_1, **Uppsala**) = 13:15 + 25 = 13:40

arrival (T_1, **Ludvika**) = 15:30 + 25 = 15:55

departure (T_1, **Ludvika**) = 15:40 + 25 = 16:05

arrival (T_1, **Stockholm**) = 17:10 + 25 = 17:35

Cost of delay = 6 * 25 minutes = 150

Total delay = 210 min for previous delays + 150 min = 360.

Headway conflict (1a):

arrival (***T4***, **Stockholm**) = 17:35

arrival (***T1***, **Stockholm**) = 17:35

(b) Delaying ***T3*** and all its successive trips, we have:

departure (***T3***, **Mora**) = 11:35 + 15 = 11:50

arrival (***T3***, **Uppsala**) = 12:35 + 15 = 12:50

departure (***T3***, **Uppsala**) = 14.00 + 15 = 14:15

arrival (***T3***, **Mora**) = 15:00 + 15 = 15:15

departure (***T3***, **Mora**) = 15:30 + 15 = 15:45

arrival (***T3***, **Ludvika**) = 17:30 + 15 = 17:45

departure (***T3***, **Ludvika**) = 17:50 + 15 = 18:05

arrival (***T3***, **Stockholm**) = 19:35 + 15 = 19:50

Cost of delay = 8 * 15 minutes = 120

Total delay = 210 min for previous delays + 120 min = 330.

Headway conflict (1b):

arrival (***T3***, **Stockholm**) = 19:50

arrival (***T2***, **Stockholm**) = 19:55

The algorithm will compare these costs and store the cost of each train involved and then pick the one with less cost as the required path to explore first, and we have:

2. The algorithm will now pick the current path with less cost and explore the conflict encountered from it and uses possible delay time for each option:

From conflict at **1b**, we have:

(i) Then delaying ***T2*** and all its successive trips for 15 minutes, we have:

departure (***T2***, **Ludvika**) = 18:25 + 15 = 18:40

arrival (***T2***, **Stockholm**) = 19:55 + 15 = 20:10

Cost of delay = 2 * 15 minutes = 30

Total delay = 330 min for previous delays + 30 min = 360.

The algorithm will detect that a solution has been found, then compare the cost with other paths still needed to be explore and also with any solution found previously if any.

(ii) Delaying T_3 and all its successive trips with 25 minutes, the algorithm will detect a solution has been found (see figure 5.4). The algorithm will then compare the cost of solution with the solution achieved from 1b(i). Since the cost of this solution is greater than the cost of solution from 1b (i), the algorithm will delete this path of solution found.

From conflict at **1a**, we have:

(i) Delaying T_1 for 20 minutes, this will result in the cost that will be greater than solution from 1b (i), then the algorithm will delete this path of the solution found.

(ii) Also, delaying T_4 for 20 minutes, this will result in the cost that will be greater than solution from 1b (i), then the algorithm will delete this path of the solution found.

At this point, the algorithm have been able to find the path that has less cost of delaying trains involved in conflicts, this solution found gives the optimal solution path to the problem.

Total Cost of Solution using Best-first search algorithm:

Final Total Cost = 210 min for the departure delay of T_1 + 120 min for the delay of T_3 + 30 min for the delay of T_2 = **360**.

Note: Comparing this with all possible solutions to the problem shows that the result achieved is an optimal solution one.

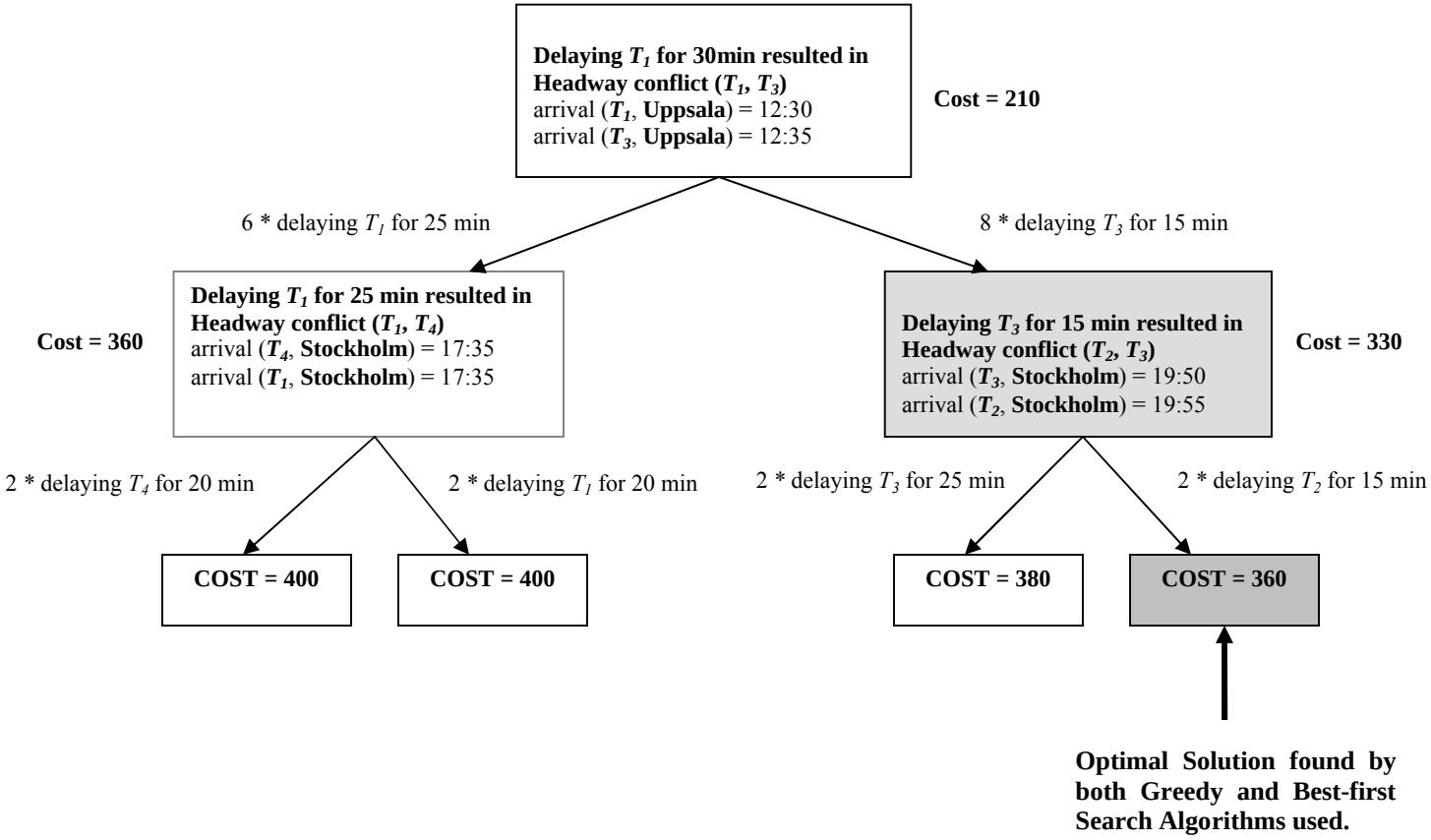

Figure 5.7: Showing possible solutions to the problem on sample network 3.

5.2 ANALYSIS

The networks considered are in forms of real life rail network which helped in testing the heuristics approaches proposed to solve the problems considered on them. These heuristics methods proposed solves the conflicts as they occur and looks ahead for possible conflicts as a result of solving the previous ones, therefore the selection criteria used in selecting the train to be delayed for the others needs to be carefully chosen to have feasible and reliable solution because it's the aim of this thesis work. From the sample networks considered, these heuristics methods solves the problems encountered and favorably gives better or optimal solutions as compared to other possible ones available, but the time and memory space used in finding the solutions differs.

The Greedy heuristic approach (*algorithm 1*) uses the selection criteria which is the cost of delaying each trains involved in the conflict. This selection rule used makes the algorithm to be faster in finding a possible solution as we can see from the sample problems considered on the networks and results in better or optimal solutions to some of the problems. This selection rules can easily miss a better or optimal solution if a more busy or complex timetable is considered but the steps used will always be closer to the best solution if not the best as we can see from the *sample network 2* when *sample problem 2* was considered on it. The result achieved from the problem resulted in not the optimal solution but a much better solution when compared to available possible ones.

With the Best-first search approach (*algorithm 2*) used, the only deciding factor is exploring the path with less cost first as long as the path is lesser than any available one. Since the searching steps used is more of searching along each possible levels of conflicts encountered, the time involved in searching for a better solution is greater than for the Greedy heuristic approach and definitely can give better or optimal solutions to more complex problems when used. All the sample problems considered in testing the Best-first search method solves the problems and resulted in producing optimal paths of solution to the problems. This approach to solve dispatching problems can most of the time leads to optimal solution since it considers and explore available paths with less cost first, it will backtrack if the current cost of path taken is greater than any other path available. From *sample problem 2* considered on *network 2* shows the ability of this heuristics, it produces a solution which is optimal which the Greedy approach could not achieve. The solutions from the sample problems shows the effectiveness

of this methods of solving problems as it will result in giving optimal solution as shown from results achieved from sample problems.

Both the Greedy and Best-first search heuristics algorithms used on the sample problems gave better or optimal solution to the problems but the Greedy approach found the solutions faster than the Best-first search. The execution time of achieving results from both algorithms differs as a result of the approaches or methods used in finding reliable solutions by each algorithm. The execution time of the Greedy method will always be lesser than that of the Best-first search methods due to their steps used in finding solutions. The Best-first searches across each level of the search space looking for paths with less cost and results in backtracking, while the Greedy method only move along the path with less cost when a conflict is encountered without backtracking (only focuses on current conflicts encountered). The search space and memory space used by Best-first search will always be greater than that of the Greedy heuristic approach due to the methods explained above. Both algorithms are reliable as they both solves dispatching problems and can help the dispatchers in making reliable and cost effective decisions in solving problems. The Best-first search approach is more reliable in achieving optimal solution to problems as a result of its ability to search extensively through the search space available.

CHAPTER 6

RECOMMENDATION

The rail network considered are in form of a real life network but are very much smaller due to number of trains, stations and other infrastructures used. The heuristics used in solving the dispatching problems on the networks were able to solve the problems but when considering a more complex network, the result may not be easily found as a result of the search space involve. The Greedy heuristic uses some selection rules in deciding which train to be delay, these rules may not always give the right decision when considering a much more complex network as the selection rules used may not be satisfactory in decision making. The Best-first search heuristic used will always give optimal solution but the search space it always considered will be much when considered on a much more complex rail network. Reduction of the search space and possible satisfactory selection rules which can help in finding better solution for stochastic problems are required to have a much more reliable and cost effective dispatching tools. For better solutions for dispatching problems, further researches that I think can help in solving dispatching problems is given below:

- Having a selection criterion that can give satisfactory decision when there are conflicts between trains by deciding which train to be delay for the other to continue its journey. A possible priority between trains can really help, but needs to be carefully used as certain trains needs to be given higher priority than the others.

- The use of genetic algorithm can be use to solve the dispatching problems as it is flexible and does not require derivative information of the objective function.

- Also a more research work needs to be done on using branch and bound algorithms in finding solutions to dispatching problems.

CHAPTER 7

CONCLUSIONS

Careful study on dispatching problems were done and that helped in the choice of heuristics algorithms been proposed. The file format used by the tool considered are in xml file format which are easily accessible easily via the World Wide Web. The proposed heuristics algorithms considered on the rail network were able to solve the problems but they also have some short comings when considering a much more complex rail network. The Greedy heuristics can solve dispatchers' problems but can easily miss the optimal solution, but the Best-first search approach will always find the optimal solution but the time of operation will always be greater than that of the Greedy method. Also the memory space used by the Best-first search method is always greater than that of the Greedy approach because of the search space involved in its operations. The Best-first search approach is more reliable in achieving optimal solution to problems as a result of its ability to search extensively through the search space available.

For better solution the following recommendations in chapter 6 needs to be carefully considered. The aims and objectives of this work was achieved as it helps the dispatchers in making decisions in solving dispatching problems they faces by finding much more better or even optimal solution.

REFERENCES

1. Michiel J.C.M. Vromans, Rommert Dekker, and Leo G. Kroon
"Reliability and Heterogeneity of Railway Services"
Erasmus Research Institute of Management (Report Series *Research in Management*), pp. 1-4.

2. M.A. Salido, M. Abril, F. Barber, L. Ingolotti, P. Tormos, A. Lova
"Topological Constraints in Periodic Train Scheduling", pp. 1-4.

3. Kurdi Mohamed (2004), Masters Thesis Project:
"Simulation of Train Dispatching Problem", pp. 1-40.
Department of Computer Science, Hogskolan Dalarna University, Sweden.

4. Güvenç Şahin, Ravindra K. Ahuja, Claudio B. Cunha
"New Approaches for the Train Dispatching Problem", pp. 1-14.

5. C.K. Chinu, C. MChou, J. HM. Lee, Leung, and Y. W. Leung
"A Constraint Based Interactive Train Rescheduling Tool"
Department of Computer Science University of Hong Kong, Shatin, N. T,
Hong Kong, pp. 1-9.

6. Chris George (December 1995).
"A Theory of Distributing Train Rescheduling", pp. 1-18.
UNU/IIST (International Institute for Software Technology)

7. Yu Cheng (November 1997).
"Hybrid Simulation for Resolving Resource Conflict in Train Traffic Rescheduling".
ILOG Co. Ltd, Ys Sanbancho Building 24-14 Sanbancho, Chiyoda-Ku,
Tokyo 102, Japan, pp. 1-3.

8. Ismail Sahin (1999)
"Railway Traffic Control and Train Scheduling Based on Inter-Train Conflict Management", *Transportation Research B* 33, pp. 511-534 (1999).

9. R. L. Sauder and W. Westerman (1983),

"Computer Aided Train Dispatching Decision Support Through Optimization" *Interfaces* 13(6), pp. 24-37.

10. Szpigel B. (1972), "Optimal Train Scheduling on a Single Track Railway" In Roos M., editor, Proceeding of IFORS Conference on Operational Research", pp. 324-343.

11. A. Higgins, E. Kozan, and L. Ferreira (1997), "Heuristic Techniques for Single Line Train Scheduling", *Journal of Heuristics 3*, pp. 43-62.

12. X. Cai, C. J Goh, and A. I. Mees (1998), "Greedy Heuristic for Rapid Scheduling of Train on a Single Track", *IIE Transactions 30*, pp. 481-493.

13. http://en.wikipedia.org/wiki/Rail_transport

APPENDIX

APPENDIX A: LIST OF FIGURES

1)

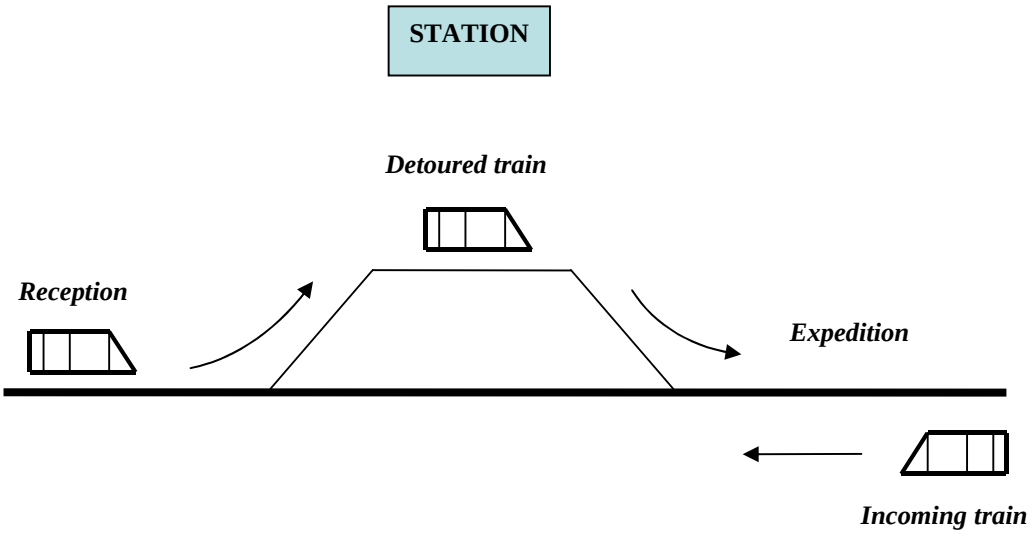

Figure 2.1: Showing train crossing with related constraints

2)

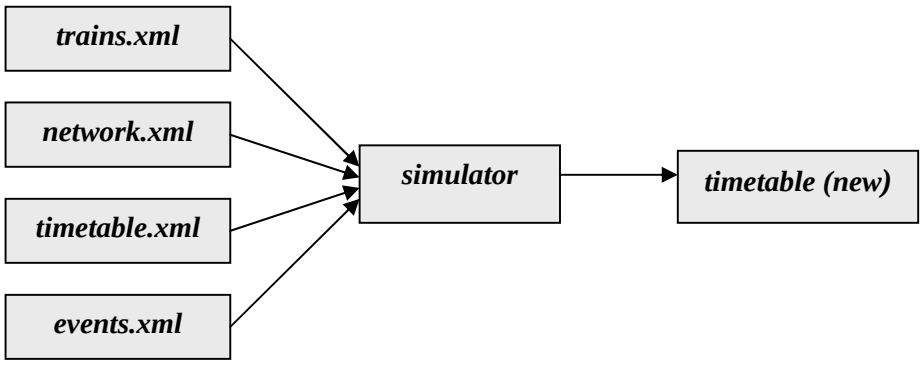

Figure 4.1: Shows the links between the simulator and the input files.

3)

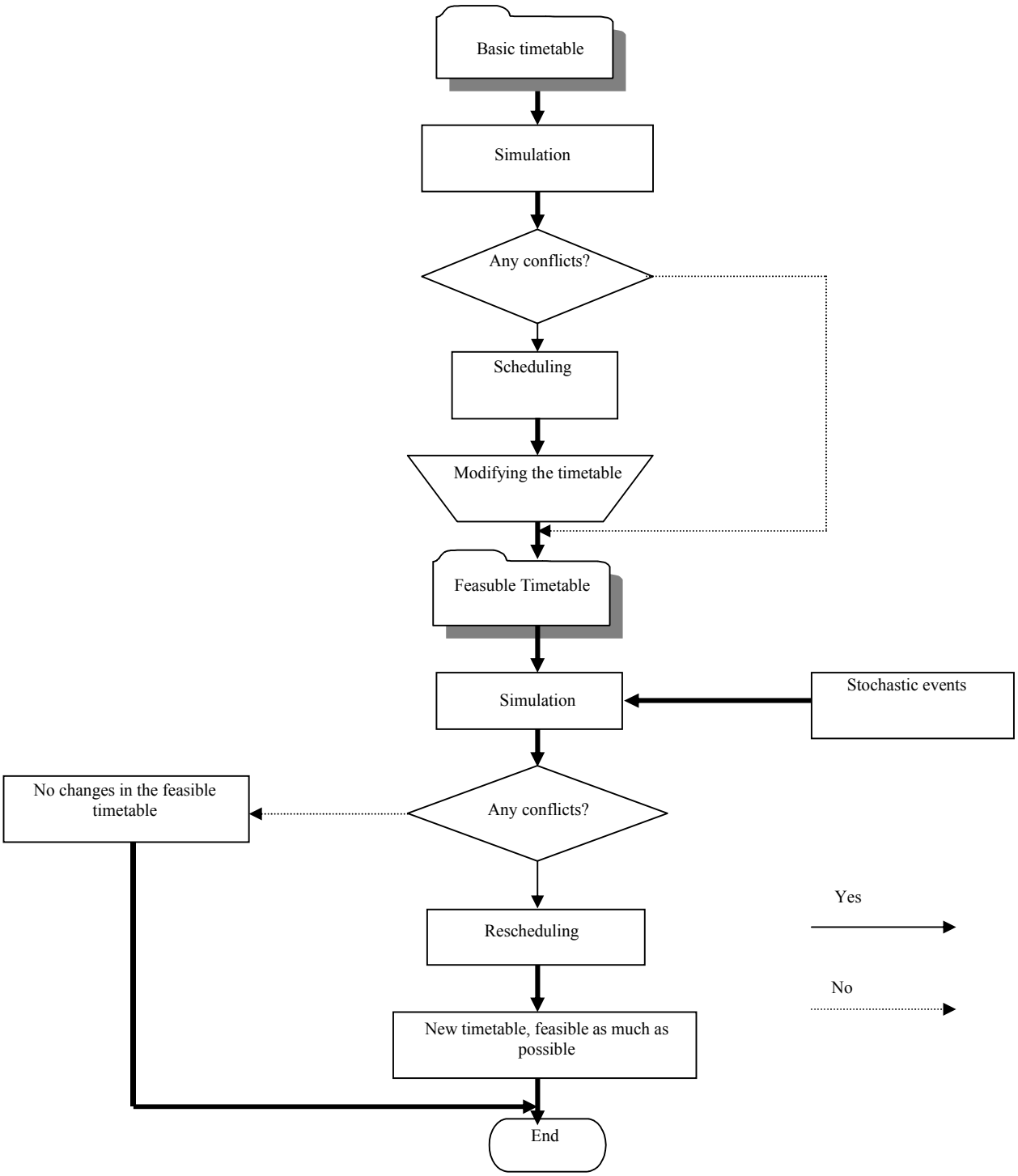

Figure 4.2: Simulation process with scheduling and rescheduling

4)

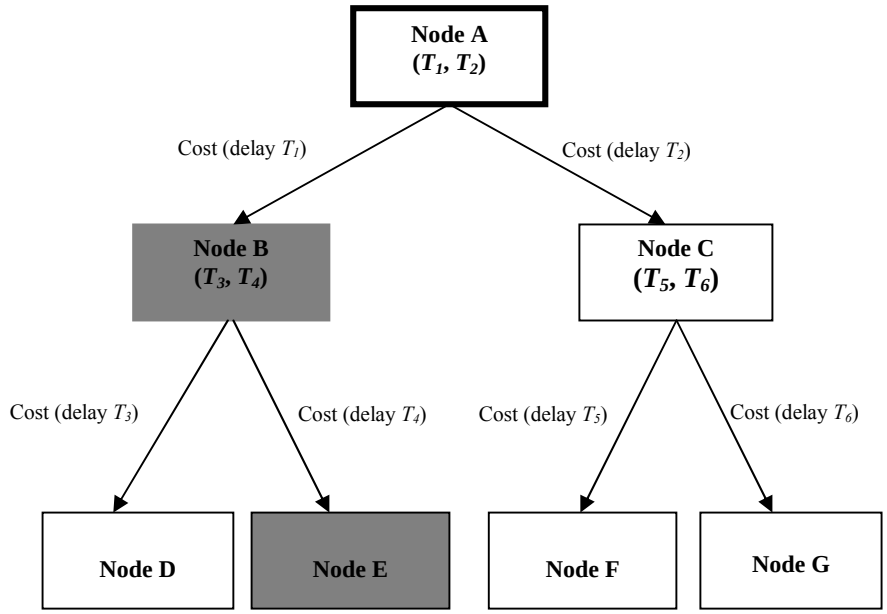

Figure 4.3: Shows the nodes and the costs of delaying the trains involves along the path.

5)

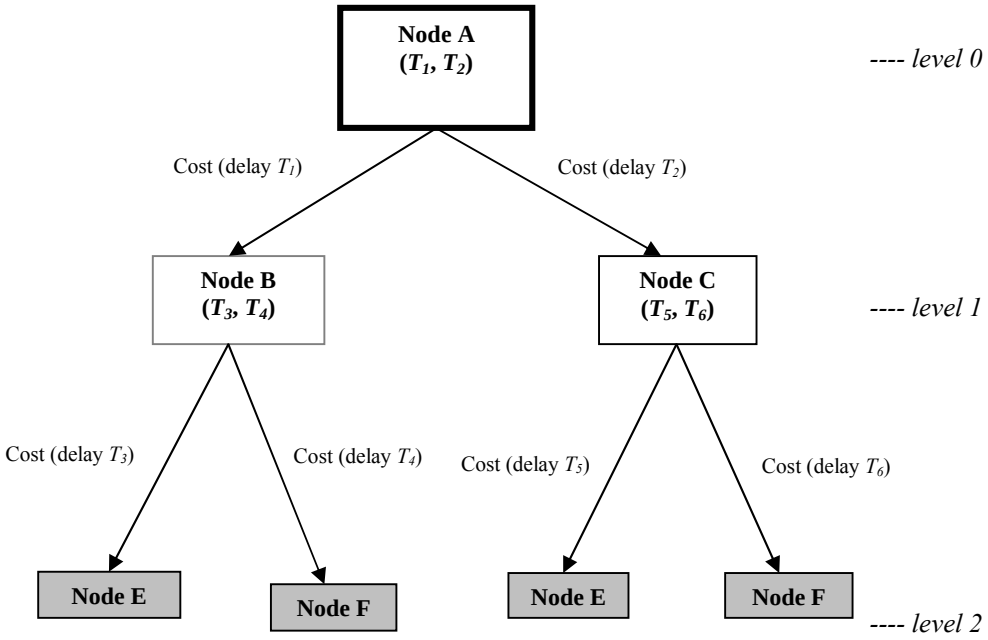

Figure 4.5: Shows the nodes and the costs of delaying the trains involves along the path.

6)

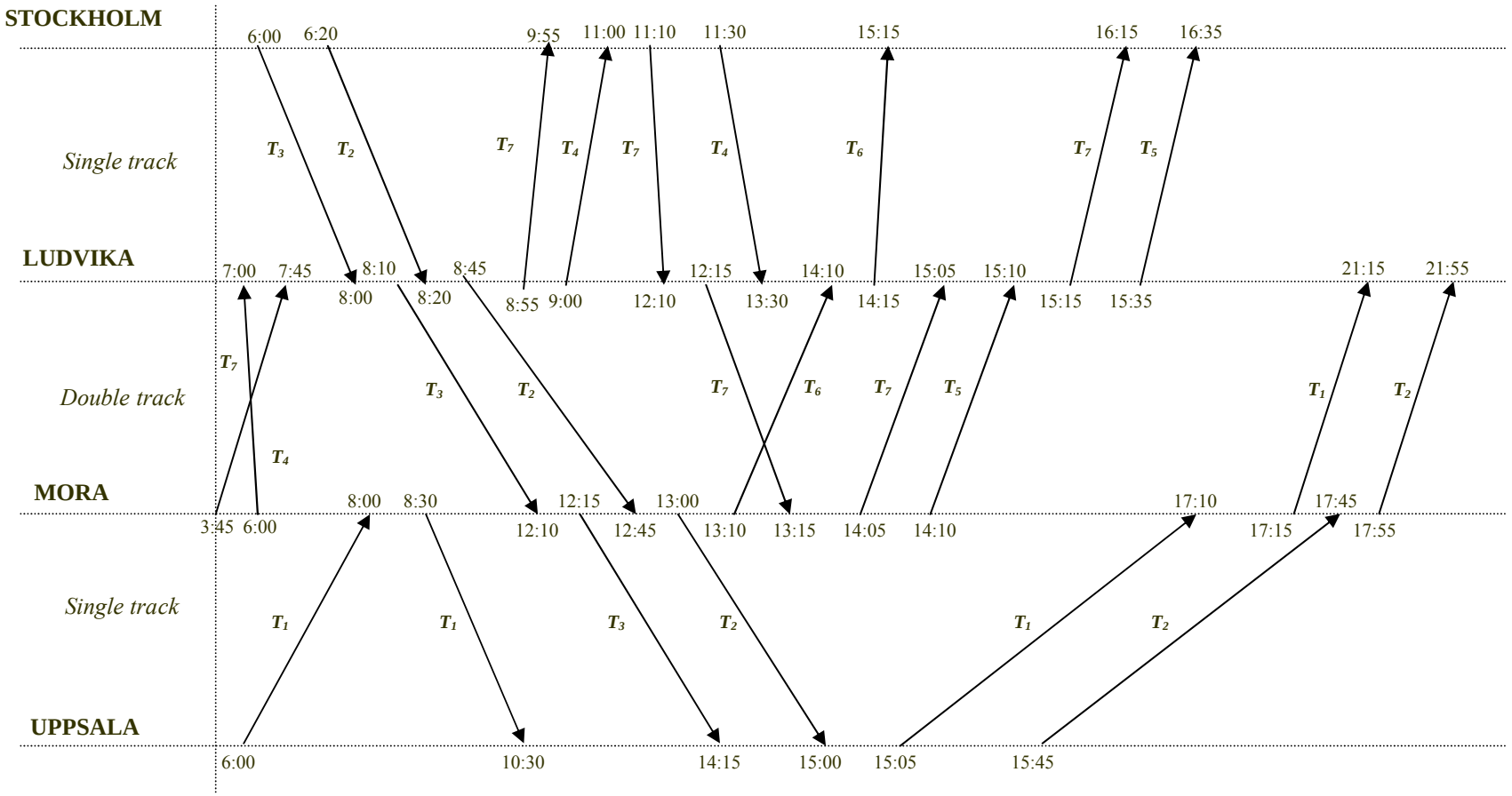

Figure 5.1: Sample Network 1

7)

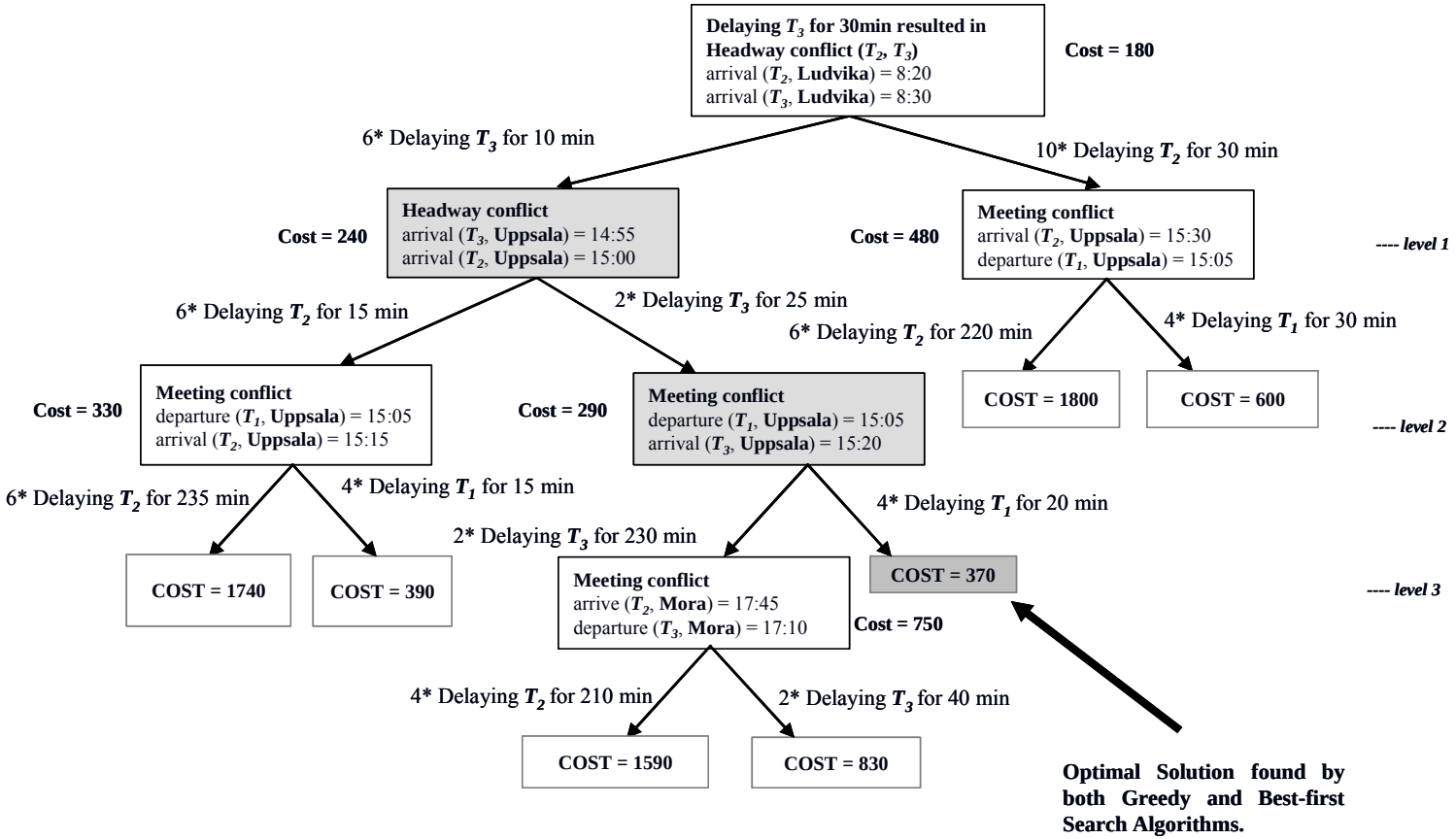

Figure 5.2: Showing possible solutions to problem 1 of Sample Network 1.

8)

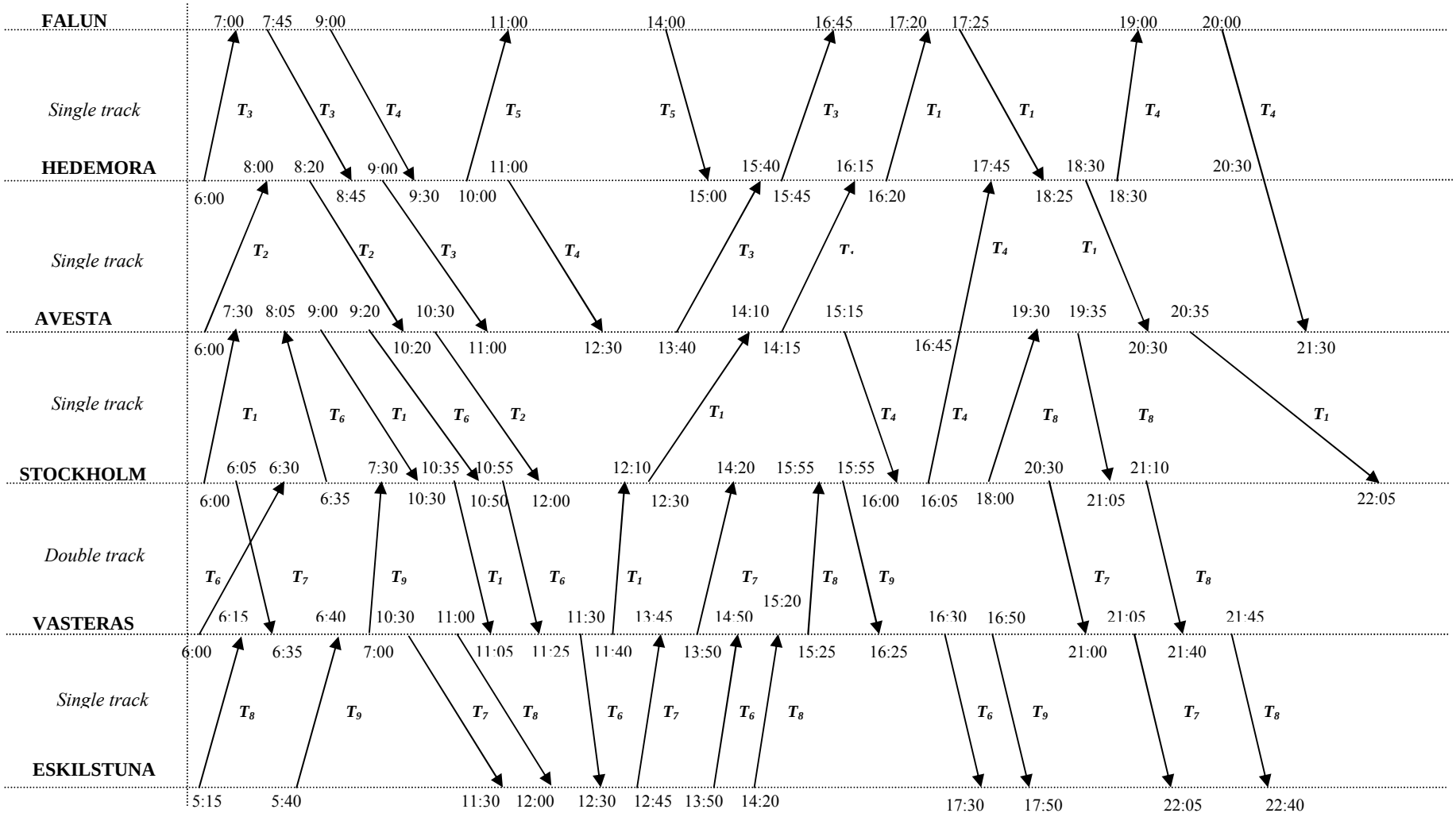

Figure 5.3: Sample Network 2.

9)

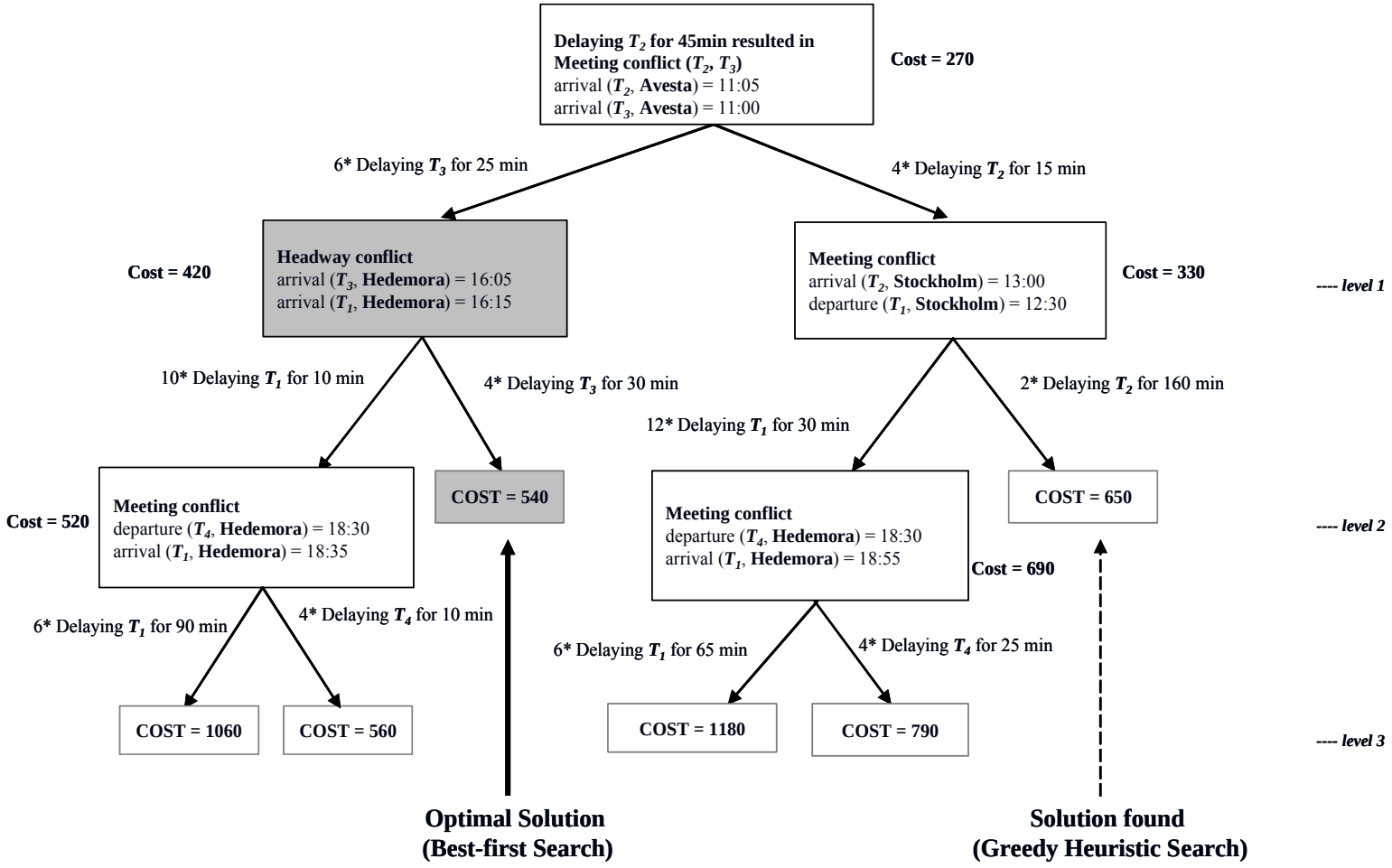

Figure 5.4: Showing possible solutions to the problem 2 on Sample Network 2.

10)

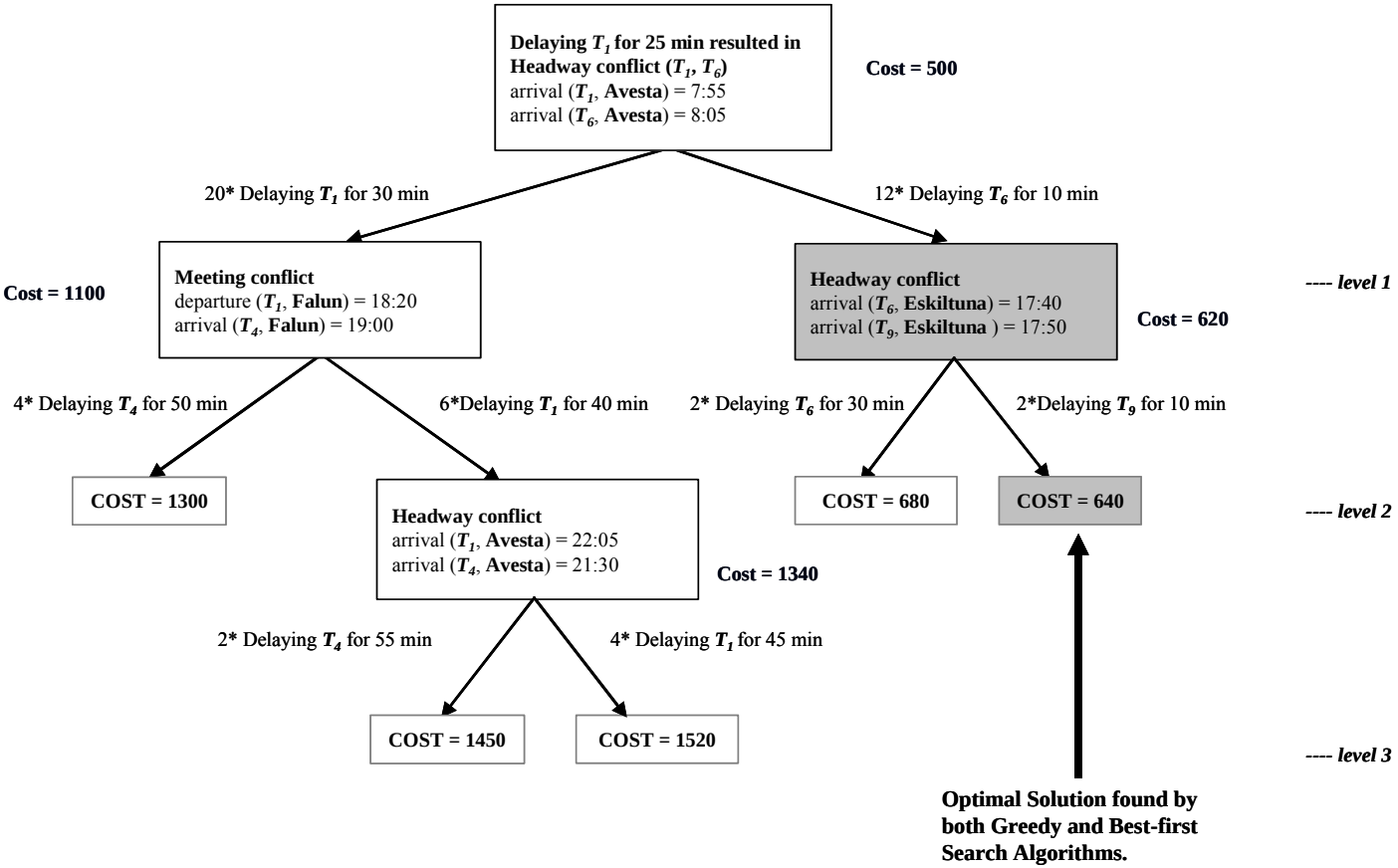

Figure 5.5: Showing possible solutions to the problem 3 on Sample Network 2.

11)

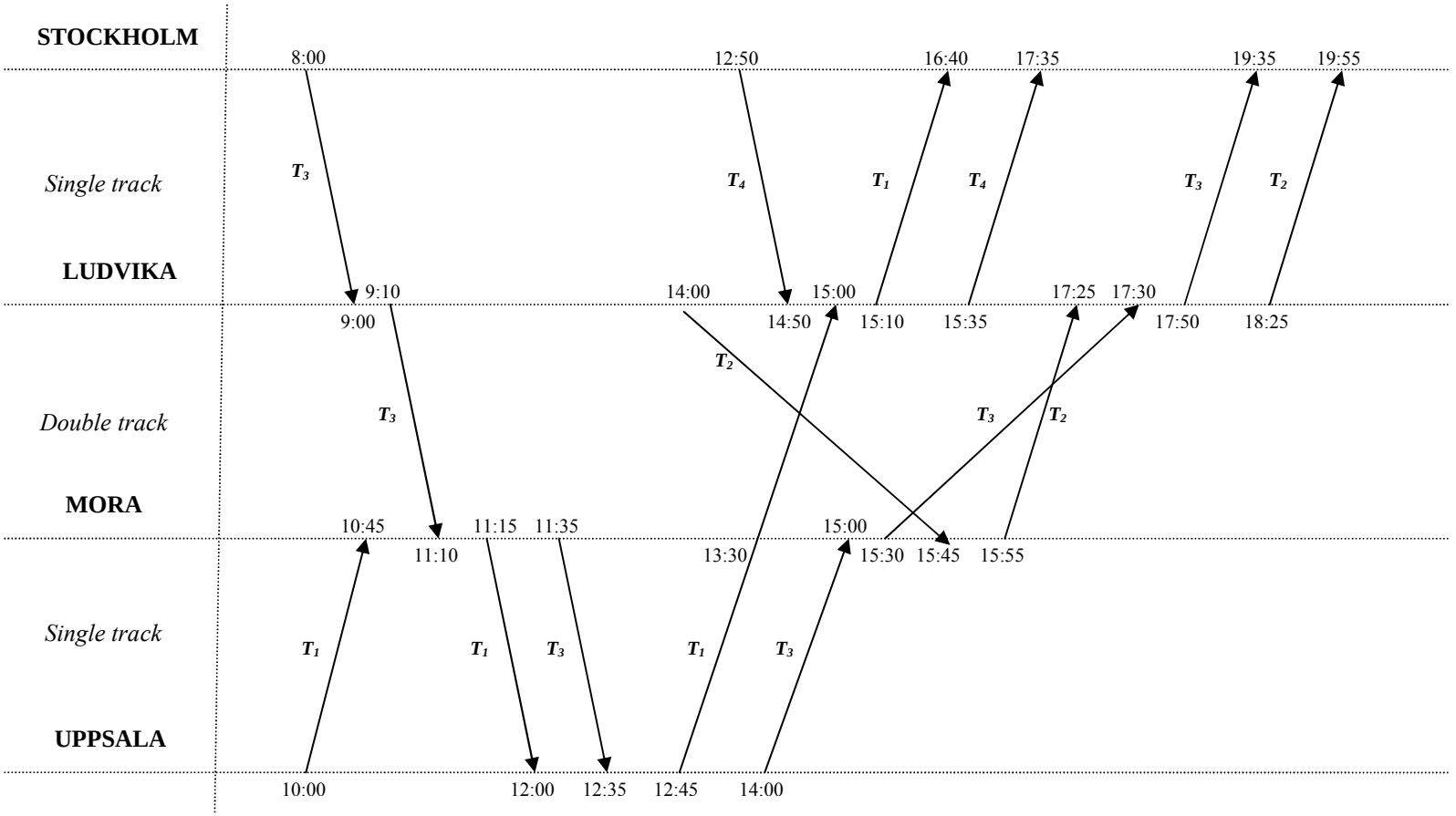

Figure 5.6: Sample Network 3.

12)

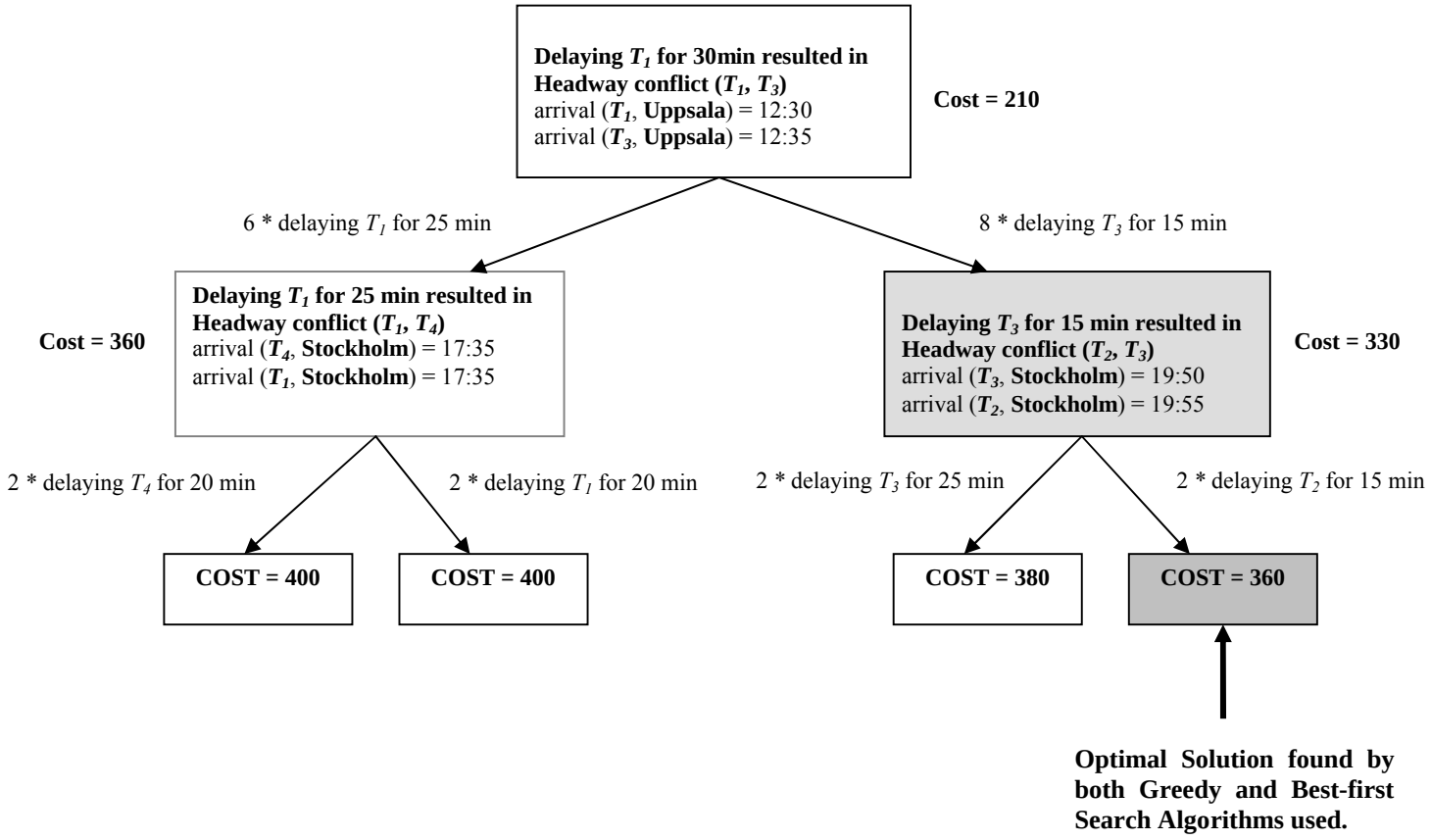

Figure 5.7: Showing possible solutions to problem 4 on Sample Network 3.

APPENDIX B

The syntax of the ***trains.xml*** is given below:

```
<?xml version="1.0" ?>
<!ELEMENT trains (traintype+)>
 <!ELEMENT traintype (maxspeed, weight, length, power)>
  <!ATTLIST traintype type CDATA #REQUIRED>
  <!ELEMENT maxspeed (#PCDATA)>
  <!ELEMENT weight (#PCDATA)>
  <!ELEMENT length (#PCDATA)>
  <!ELEMENT power (#PCDATA)>
```

APPENDIX C

The syntax of the ***events.xml*** is given below:

```
<!ELEMENT events (event+)>
  <!ELEMENT event (type, train_ID?, location, Dis_to_start?, Origin_Hour, Origin_Minute,
                   Delay Time, Time_To_SURE)>
     <!ATTLIST event ID CDATA #REQUIRED>
     <!ELEMENT type (#PCDATA)>
     <!ELEMENT train_ID (#PCDATA)>
     <!ELEMENT location (#PCDATA)>
       <!ATTLIST location type CDATA #REQUIRED>
     <!ELEMENT Dis_to_start (#PCDATA)>
     <!ELEMENT Origin_Hour (#PCDATA)>
     <!ELEMENT Origin_Minute (#PCDATA)>
     <!ELEMENT Delay Time (#PCDATA)>
     <!ELEMENT Time_To_SURE (#PCDATA)>
```

APPENDIX D:

The syntax of the ***network.xml*** is given below:

```
<?xml version="1.0" ?>
<!ELEMENT network (station+, connection+)>
 <!ELEMENT station (yard_capacity)>
  <!ATTLIST station name
            (Stockholm|Ludvika|Mora|Uppsala|Falun|Borlange|Linkoping|Nykoping|Kiruna
              |Leksand|Ratvik) #REQUIRED>
  <!ELEMENT yard_capacity (#PCDATA)>
 <!ELEMENT connection (station1, station2, type, track+)>
  <!ELEMENT station1 (#PCDATA)>
  <!ELEMENT station2 (#PCDATA)>
  <!ELEMENT type (#PCDATA)>
  <!ELEMENT track (track_length, maximum-weight, maximum-speed)>
   <!ATTLIST track name ID #REQUIRED>
   <!ELEMENT track_length (#PCDATA)>
   <!ELEMENT maximum-weight (#PCDATA)>
   <!ELEMENT maximum-speed (#PCDATA)>
```

APPENDIX E:

The syntax of the ***timetable.xml*** is given below:

```
<?xml version="1.0" ?>
<!ELEMENT timetable (train+)>
 <!ELEMENT train (train_type, departure_station, departure_hour, departure_minute,
                  track_id, station*, destination_station, arrive_hour, arrive_minute)>
 <!ATTLIST train number CDATA #REQUIRED>
 <!ELEMENT train_type (#PCDATA)>
 <!ELEMENT departure_station (#PCDATA)>
 <!ELEMENT departure_hour (#PCDATA)>
```

<!ELEMENT departure_minute (#PCDATA)>

<!ELEMENT track_id (#PCDATA)>

<!ELEMENT station (name, arrive_hour, arrive_minute, departure_hour, departure_minute, track_id)>

<!ELEMENT name (#PCDATA)>

<!ELEMENT arrive_hour (#PCDATA)>

<!ELEMENT arrive_minute (#PCDATA)>

<!ELEMENT destination_station (#PCDATA)>