Balfour Beatty
Rail

Mälardalens Högskola
Balfour Beatty Rail AB

Evaluation of Weak Relations in TracFeed

Master Degree Thesis
Electronics
Department of Computer Science and Electronics (IDE)

Supervisor: Mikael Ström
Examiner: Mikael Ekström
Author: Fredrik Gauffin

Abstract

This report is a master degree thesis in electronics that examines if weak relations can be used in TracFeed. TracFeed is a simulation tool designed to aid in the dimensioning of the power supply system for electrical railroads. The program was originally developed by Adtranz and is today further enhanced by Balfour Beatty Rail AB.

When using weak relations there will be fewer truncations of the steps and therefore the response time will decrease. The purpose of this thesis is to study how much time that can be gained in different types of simulations and in which way the result is effected because of weak relations.

The theoretical part explains the fundamentals concerning electrical trains and how trains are modelled in TracFeed. The calculation kernel used by TracFeed is called SIMPOW and it is described as well as the modelling language which is used to create the train models.

Weak relations are tested in two simulations. The result from those simulations shows that if a displacement in simulated time of the output is acceptable and there are many trains active simultaneously weak relations are an interesting alternative.

Terminology

Term	Description
Asynchronous motor	Also called induction motor. An AC motor where the rotor rotates slower than the magnetic field from the stator.
AT system	Auto Transformer system. A type of feeding system that use a negative feeder and transformers connected in such a way that the distance between converter stations can be increased compared to a BT system. [1]
BT system	Booster Transformer system. A type of feeding system used to control the path that the return current takes. [2]
Catenary	A line that transmits electrical energy to trains.
El18	A Norwegian locomotive built by Adtranz with an asynchronous motor. [3]
EMC	Electromagnetic Compatibility. EMC is the ability of different items of electrical equipment to work together without suffering the effects of interference. [4]
GTO	Gate Turn-Off Thyristor
IGBT	Insulated Gate Bipolar Transistor
IORE	A Swedish locomotive built by Bombardier with an asynchronous motor. The locomotive hauls iron ore on Malmbanan. [5]
Malmbanan	A railroad section between Luleå and Narvik.
Pantograph	A device on top of trains that collect the current from the catenary.
pu	Per unit. Quantities are expressed as a fraction of a base value.
RC4	A Swedish locomotive built by ASEA with a thyristor converter and a DC motor. The locomotive was designed to pull both passenger and freight trains but is today only used in cargo trains. [6]
RC6	A development of the RC4.

Term	**Description**
REGINA	A passenger train developed by Adtranz with an asynchronous motor.[7]
Regenerative braking	The trains motor functions as a generator during braking and produces power for the feeding system.
Thyristor converter	A device that converts AC to DC where the output voltage can be controlled.
Tractive effort	The pulling force generated by a locomotive
VCR stations	Voltage Controlled Rectifier stations. AC to DC converter stations with a thyristor converter.

Table of contents

1 Introduction

1.1 Balfour Beatty Rail presentation

Balfour Beatty Rail is an international company with 5500 employees that focuses on infrastructure for railroads. Balfour Beatty Rail AB (henceforth known as BBrail), the Swedish part of the company, is situated in Västerås. The primary markets are the Nordic and Baltic countries but the company is also involved in projects in Malaysia, Turkey and Korea.

BBrail is one of the leading suppliers in the Nordic countries in areas such as power supplies, operation, maintenance and telecommunications for railroads.

More information about BBrail is available at http://www.bbrail.se

1.2 TracFeed presentation

In the beginning of 1990 Adtranz and the Swedish railroad administration Banverket started to discuss a simulation tool for electrical railways that could be used to dimension the power supply system. The program was originally named SimTrack but the name was later changed to TracFeed Simulation (henceforth known as TracFeed). It was developed with the power system analysis program SIMPOW as a calculation kernel. The first TracFeed version was released in 1995 for UNIX and in 2000 a Windows version was available. Today the program is being further enhanced by BBrail.

Currently TracFeed is used by BBrail, the Norweigian railroad administration Jernbanverket and Banverket.

Jernbanverket has made a large set of studies from 1995 and onwards. TracFeed has also been used in the following projects by BBrail [8]:

- Malaysia: Kuala Lumpur Upgrade, 50Hz transformer stations, included verification
- Malaysia: Rawang Ipoh, 50Hz transformer stations
- Korea: Pusan subway, DC VCR stations
- Turkey: Light rail and subways, DC diode stations
- Norway: Bergen Bybane, DC diode and VCR stations
- Sweden: Malmbanan bergslagsstråket, 16 2/3Hz converter and transformer station

Banverket recently became a user.

TracFeed can be used to simulate the following [8]:

- AC and DC systems
- Arbitrary feeding and distribution grid
- Free placement of stations
- Arbitrary traffic situation
- Regeneration of power
- Reaction to low and high voltage
- The influence of curves, hills, tunnels and speed limits
- AT systems
- DC train with an induction motor
- AC train with an induction motor
- AC train with a DC motor and a thyristor converter

1.3 Background

TracFeed is event-driven, meaning that the behaviour of the trains is decided by events. An event is caused by a change in the truth value of an if-statement. If for example the train has reached a curve or there should be a decrease in the tractive effort caused by a low voltage this will lead to a change in the truth value of a relation causing an event. When an event happens the time step will be truncated at the time of the event and the changes caused by the event are applied before simulated time is allowed to advance. In large simulations events happens frequently which leads to long response times.

To remedy this weak relations can be used. When a relation such as *if (a > b)* is made weak it will not truncate the step when it changes truth value, instead the truth value will be changed at the end of a step. This gives the possibility to gather events at the end of steps.

1.4 Purpose

The object of this thesis is to thoroughly examine if weak relations can be used in TracFeed. This involves looking into which relations that can be made weak without altering the out data in any significant way. The thesis should present conclusions concerning how much time that can be gained in different types of simulations and in which way the result is effected because of weak relations.

1.5 Limitations

TracFeed includes three train models, a train with an induction motor in an AC feeding system, a train with an induction motor in a DC feeding system and a train with a DC motor in an AC feeding system. The two models with induction motors are practically identical; therefore weak relations will only be tested with the AC version.

This is a commercial program so therefore only parts of the code can be shown in this work.

2 Theory

2.1 Traction basics

The facts in this section is based on [2] and the formulas and the units presented here are the same as in TracFeed. A train's tractive effort must be enough to overcome the total train resistance and deliver the desired acceleration. The braking force together with the total train resistance has to give the desired retardation. The total tractive effort or the braking force, F (kN), can be expressed with the following force equation:

$$F = m_a \cdot a + F_r \tag{1}$$

where m_a is the trains dynamic mass (tons), a is the acceleration (m/s) and F_r (kN) is the total train resistance.

In TracFeed the total train resistance is divided into three parts:

- Running resistance
- Gradient resistance
- Curve resistance

2.1.1 Train resistances

2.1.1.1 Running resistance

The running resistance, F_{rr} (kN), is the resistance the train meets when it is on a horizontal track with no curves, TracFeed also includes the resistance from possible tunnels. The running resistance is caused by for example friction in the wheel bearings, unevenness on the wheels, energy losses in the dampers and air resistance. It is calculated in the following way:

$$F_{rr} = RRA + RRB \cdot v + TUNNEL \cdot RRC \cdot v^2 \tag{2}$$

The parameters RRA, RRB and RRC are vehicle specific constants. The variable v is the speed of the train in km/h. $RRC \cdot v^2$ corresponds to the air resistance which increases when the train is in a tunnel, hence $TUNNEL$ is larger than one when the train drives through a tunnel otherwise it is one.

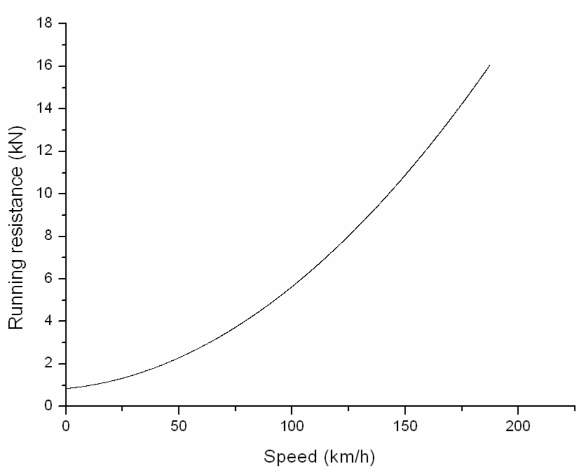

Figure 1: Running resistance as a function of the speed for a Regina train

2.1.1.2 Gradient resistance

The slope of the track is of big importance when it comes to the resistance the train has to overcome. The resistance from a slope, F_{gr} (kN), is expressed in the following way:

$$F_{gr} = m \cdot g \cdot \frac{S}{1000} \tag{3}$$

The constant m is the weight of the train (tons), g is the acceleration of gravity (m/s^2) and S is the gradient in permillage.

2.1.1.3 Curve resistance

When a vehicle passes through a curve there will be friction and sliding between the track and the wheels. The force due to the curve resistance, F_{cr} (kN), is calculated in the following manner:

$$F_{cr} = m \cdot \frac{CR_0}{r - CR_1} \tag{4}$$

The parameters CR_0 (kNm/tons) and CR_1 (m) are train specific parameters, m is the weight of the train (tons) and r is the radius of the curve (m).

2.1.1.4 Total train resistance

The total tractive effort needed to keep a certain speed is given by the total train resistance F_r which is the sum of the previous resistances.

$$F_r = F_{rr} + F_{gr} + F_{cr} \tag{5}$$

2.1.2 Acceleration/ Retardation

The tractive effort F_a (kN) needed for acceleration or retardation is calculated as:

$$F_a = m_a \cdot a \tag{6}$$

where m_a is the dynamic mass (tons) and a is the acceleration or retardation (m/s^2). The dynamic mass includes the static mass (weight of the train) and the effect of the moment of inertia.

If eq. 5 is added to eq. 6 the sum is the total tractive effort, eq. 1.

2.1.3 Limitations to the tractive effort

A trains tractive effort is typically limited by the motors maximum moment up to base speed. Above base speed the engine produces constant power. The power, P (W), is given by the following formula:

$$P = F \cdot v \tag{7}$$

where F is the tractive effort (N) and v is the speed (m/s). This limitation is represented in TracFeed as the maximum value of $F \cdot v$ called *TRAINLIMIT1* $(kN \cdot km/h)$ where the unit for F is kN and v is km/h (since the unit for v is km/h and not m/s, $F \cdot v$ will not have the unit Watt). This gives the maximum tractive effort above base speed, *FLIMIT*1 (kN), as:

$$FLIMIT1 = \frac{TRAINLIMIT1}{v} \tag{8}$$

At even higher engine speeds the tractive effort is usually limited by for example requirements of good commutation for a DC-engine and the margin to the pitching moment for an asynchronous motor. Here the tractive effort decreases more than inversely proportional to the speed. This is modeled in TracFeed in the following way:

$$FLIMIT2 = TRAINLIMIT2 \cdot \left(\frac{MAXSPEED}{v} \right)^2 \tag{9}$$

where *TRAINLIMIT2* (kN) is the tractive effort at the trains max speed and *FLIMIT*2 is the limitation in the force it causes.

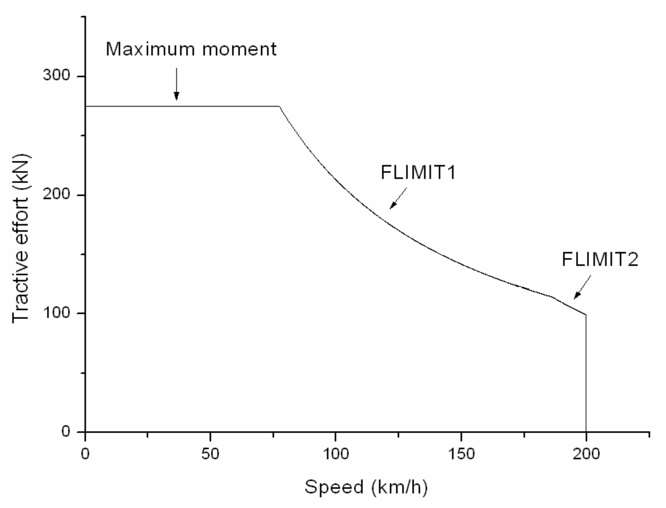

Figure 2: Tractive effort as a function of the speed for an El18 train with the limiting factors shown

The above limits are valid at nominal voltage. If the voltage drops below nominal voltage because of for example a heavily trafficked section that can lead to a reduction of the tractive effort.

If the force from the engine is going to develop a tractive effort there has to be some friction between the wheels and the railroad track, the term that describes this is called adhesion. The maximum force the train can develop because of adhesion, $F_{adhesion}$ (kN), is calucluated in the following way:

$$F_{adhesion} = ADHMASS \cdot g \cdot \left(ADHCOEFF + \frac{ADH1}{ADH2 + v} \right) \tag{10}$$

ADHMASS is the adhesion mass which is the total mass on the trains driving axles (tons), *g* is the acceleration of gravity (m/s^2), *ADHCOEFF* is a track dependant parameter, *ADH*1 and *ADH*2 are train dependant constants (km/h) and *v* is the speed of the train (km/h). If the adhesion force is smaller than the desired tractive effort the tractive effort will be reduced.

2.2 Simulation of power electronic systems

According to [9] there are two types of computer programs to choose from when analyzing a power electronics circuit: circuit oriented simulators and equation solvers. In a circuit oriented simulator the user specifies the circuit topology and the component values. The circuit equations are totally transparent to the user. In an equation solver the user must specify the equations for all possible states of the circuit.

SIMPOW is a circuit oriented simulator, another example is SPICE. An equation solver could be any high-level programming language such as C or a program such as MATLAB.

If the voltages and currents in a circuit are to be calculated a number of differential equations as functions of time needs to be solved. This applies both to circuit oriented simulators and equation solvers. To do this a system of equations is created consisting of so called state variables, often the voltages and currents (the train models in TracFeed has many more state variables). They are called state variables since they describe the state of the system. The equation system is then solved at discrete points in time, at the end of steps.

2.3 SIMPOW

SIMPOW is an abbreviation of “digital SIMulation and analysis of electrical POWer systems” and it is developed by STRI. The program is the calculation kernel of TracFeed. It is designed to simulate electrical power systems of any size. AC voltages and currents are represented as phasors or by their instantaneous values and DC voltages and currents are represented by their mean values [10].

SIMPOW can use a variable step size where the step size is completely decided by the program. Alternatively the user can specify a fixed step size or a basic and maximum step size. TracFeed is usually run with the latter option where a basic step size H and a maximum step size HMAX is set. When there are few events the step size will be increased from H to HMAX.

The electrical system consists of nodes which are interconnected by lines with arbitrary impedance. To the nodes are then added for example generators and motors and SIMPOW will calculate the currents and power flows for a given amount of time. The result will be given as lists of events and diagrams of variables as a function of time or as a function of another variable.

2.3.1 OPTPOW

OPTPOW is a program within SIMPOW that calculates the initial power flows. When a system is going to be simulated this has to be run first. Input data is given in an OPTPOW file (see Appendix A: Input data for a small simulation, *OPTPOW file*).

The solution procedure for a new case starts with the system deenergized, with zero currents i.e. with a known solution. Then the reactive power sources are energized and the voltages are established, after which the active power is taken up. [10]

A static model of the electrical network, valid for symmetrical, steady state conditions with the flow of active and reactive power, losses, the voltage profile etc is computed by OPTPOW. This will establish the topology and branches of the network. [10]

The steady state solution of the system is used for the subsequent dynamic analysis (see 2.3.2).

2.3.2 DYNPOW

To dynamically simulate a system DYNPOW is used which is another program within SIMPOW. The system is here represented by dynamic models such as transfer functions and data is given in a DYNPOW file (see Appendix A: Input data for a small simulation, *DYNPOW file*). From the data given in the DYNPOW file and the OPTPOW run the initial values of the state variables are calculated. After this initialization the dynamic model of the system can be used for time simulations, such as calculation of machine transients and calculation of short-circuit currents. [10]

2.3.3 DSL

DSL, Dynamic System Language, is a modeling language used by SIMPOW. It gives the user the possibility to create own models of electrical or mechanical components. The syntax is similar to high-level programming languages such as C and FORTRAN. A process is created where a set of variables that constitute the state is declared, the previously mentioned state variables. A program code then defines how the state should change when the simulation proceeds by the use of if-statements. Several processes can be united to create a system.

There are two types of state variables, those that changes continuously in time and those that only changes at discrete points in time. Variables in the first category are called real state variables and have floating point numbers as values. Real state variables can be assigned a value in implicit assignments such as

x: .d/dt. x + x = y

where *x* is a real state variable and *y* can be a variable of any type. With the above assignment the variable *x* will be given a value so that the left side equals the right side of the equation. Real state variables can also be assigned a value in a "normal", explicit way. State variables that only change at discrete points in time are either integers or changeable constants where the latter have floating point numbers as values.

Non state variables can be declared and they function as intermediate variables. They can be of the real or integer type. If no assignment is done to a state variable the value is kept unchanged. Non state variables on the other hand are undefined under such circumstances.

In DSL cubic spline functions can be used. When spline is used on a matrix with two columns the second column is considered to be a function of the first. This means that there will be no discontinuities when using such a function.

2.3.3.1 Events and their effect on the time step

Relations that contain real variables are part of the state of a process so the program continuously checks the truth value of those relations. If, after the system of equations has been solved a truth value has changed, SIMPOW will calculate at what time that happened. At that time the step will be truncated, an event has happened, and the program code will be run again and again until the flow through the program does not change. In other words until no relation changes its truth value.

The possible logical relations in DSL are shown in table 1.

Logical relation	Explanation
.eq.	Equal to
.ne.	Not equal to
.gt.	Greater than
.ge.	Greater than or equal to
.lt.	Less than
.le.	Less than or equal to

Table 1: Logical relations

At an event there will be a discontinuity. An example of that follows in the syntax used in DSL, with the real variables *a, b* and *c*

```
If (a .gt. b) then
  c = b
else
  c = a
endif
```

When the above statement changes its truth value *c* will change discontinuously.

At an event the step will be truncated and a new step of the basic step size H will be taken from the truncation. This is illustrated in the picture below.

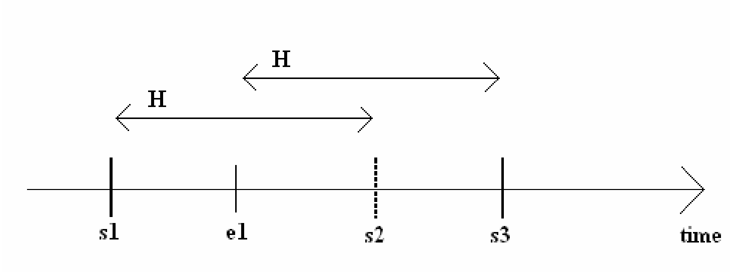

Figure 3: An event truncates the step

When t equals s1 a new step is taken and if there is no event there will be a new step after H seconds has passed, this is when t equals s2. If instead there is an event when t equals e1 then there will be a new step at s3, after H seconds has passed since e1. This means that in a simulation with many events there will be many steps; hence it will take a long time to run the simulation.

2.3.3.2 Deadband

In a relation involving real variables (state or non state) or changeable constants there is a deadband. The deadband is inserted to prevent the logical value of the relation from switching back and forth between false and true without any advancement of time, this is called “ticking”. The default size of the deadband is 0.001 but can be multiplied with an arbitrary number to make it larger or smaller. With the code example from 2.3.3.1 the concept of deadbands will be explained further (here the deadband is of the default size):

```
If (a .gt. b) then
  c = b
else
  c = a
endif
```

(a .gt. b) will be true if *a* > (*b* + 0.001) and false if *a* <= *b*. In the gap where the truth value is not clearly defined the truth value will stay the same. If *a* = 9 and *b* = 10 the relation will be false, suppose *a* starts to increase and *b* is kept constant then the relation will stay false until *a* > 10.001.

If instead *a* = 10 and *b* = 9 the relation will be true. If *a* starts to decrease and *b* is kept constant the relation will be true until *a* = 9 when it will change to false.

2.3.3.3 Weak relations

By using weak relations the goal is to gather events that cause discontinuities at the end of steps leading to fewer steps and thereby decrease the computation times of the simulations.

When a relation involving variables that change continuously in time, real or real state variables, is weak the truth value of the relation will be changed only at the end of a step and there is no deadband. In other words a weak relation will not truncate the step. If on the other hand an ordinary relation truncates the step weak relations can change their truth values. This applies when the step size equals the basic step size H. If the step size is larger than H, because of a lack of events, weak relations behave as ordinary relations with the default deadband.

It is pointless to make relations that only contain integers or changeable constants weak, since such relations never truncates the step; such relations can only change truth value when another relation have caused a truncation.

Below is a table with the ordinary relations that can be made weak and their weak counterparts:

Ordinary relation	Weak relation
.gt.	.wgt.
.ge.	.wge.
.lt.	.wlt.
.le.	.wle.

Table 2: Ordinary and weak relations

2.3.3.3.1 Weak relations example

Here follows an example of how weak relations behave.

A DSL-model contains the following relations

```
If ( a .wgt. b ) then
  ...
endif
if ( c .wgt. d ) then
  ...
endif
if (e .wgt. f ) then
  ...
endif
```

where all of the variables are of the real type.

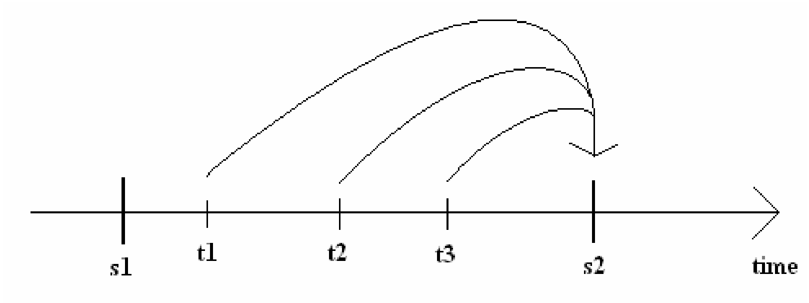

Figure 4: Changes of truth values with weak relations

When the time equals s1 a step of the basic step size H is taken and s2 = s1 + H (i.e. the time where a step would be taken if there is no event during the time between s1 and s2). At t1 *a* becomes larger than *b*, at t2 *c* becomes larger than *d* and at t3 *e* becomes larger than *f*. Since the three relations are weak their respective truth values will be changed simultaneously when the time equals s2.

Weak relations have the greatest impact on the time it takes to run a simulation when there are frequent events and the step size is large. Under those circumstances there are many events between the steps that can be gathered.

Had the relations instead been ordinary relations there would have been three truncations: when $a > (b + 0.001)$, $c > (d + 0.001)$ and $e > (f + 0.001)$ if the standard size of the deadband is used.

2.4 TracFeed

In TracFeed the trains are coded as DSL models and there are three types of train models but only two will be used for this thesis. One of the used models describes a train with a three phase induction machine with a GTO or IGBT converter in an AC feeding system. The second model depicts a train with a DC engine and a thyristor converter in an AC feeding system, called a thyristor train. The models are very similar, the main differences are that the train model with the induction motor is capable of regenerative braking which the thyristor train is not and also that the calculation of the reactive power consumption is different.

TracFeed connects each train in the simulation to a node which moves within the electrical system. The process controlling this movement is called the nodemover but that process is not of interest when implementing weak relations.

From the perspective of the rest of the simulated electrical system the trains are moving loads. The DSL code will compute the speed the trains move with in the system and the currents they consume.

The user inputs the data for the simulation in the DYNPOW file. In addition to the input SIMPOW needs about the feeding system TracFeed needs four categories of data:

- Trains
- Track
- Drive path
- Timetable

2.4.1 User input to TracFeed

The categories of data will be explained further below. For a complete example of a DYNPOW file see Appendix A: Input data for a small simulation, *DYNPOW file*.

2.4.1.1 Trains

Contains train data. This data does not include the inner structure of the train, only how it interacts with the feeding system. The data can theoretically be obtained through measurements without knowing anything about its inner structure. Below are examples of the input data.

- Mass (static-, dynamic- and adhesion mass, tons)
- Maximum speed (km/h)
- Signal length (m)
- Distance between front and mass centre (m)
- No load losses (MW)
- Auxiliary power from bridge and/or transformer (MW)
- TRAINLIMIT1 (maximum value of $F \cdot v$, $kN \cdot km / h$)
- TRAINLIMIT2 (tractive effort at maximum speed, kN)
- Reference values for acceleration and retardation (m/s^2)
- Lists with maximum tractive effort (kN) and power consumption (MW) as a function of the catenary voltage (kV)

2.4.1.2 Track

In this group track sections are created. A section has one node in each end. Each section needs data about the starting and ending position, the impedance of the catenary and track details. The track details contain lists with the track layout (information about curves, gradients, adhesion and tunnels), speed profile (allowed speeds) and the position of stations. The speed profile and stations can have several lists. In the stations list data is given concerning waiting times at stations and earliest departure from stations.

2.4.1.3 Drive path

The different sections are combined into a path. A speed and stations list is chosen for each section and the traveling direction is set (up track or down track).

2.4.1.4 Timetable

This section describes how the trains traffic the paths. Trains and paths are connected and departure times are given and there is also a possibility to set an initial speed of the train.

2.4.2 Description of the train models

The important concepts in the train models will be explained here. When nothing else is written the facts given apply to both train models.

2.4.2.1 Speed reference

The speed profile in the DYNPOW file function as a speed reference to the train. If the allowed speed is higher than the trains maximum speed the reference is decreased to the max speed. When the train enters a section where the allowed speed increases, the train starts to accelerate when the front of the train plus the signal length has passed the position where the speed profile increased. When the speed profile is decreased, the train brakes in such a way that its speed will equal the lower allowed speed when the front has reached the position where the reduced allowed speed starts.

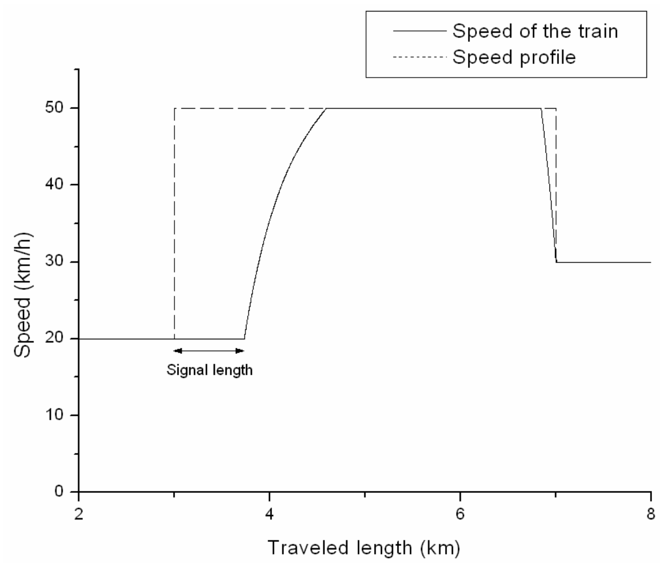

Figure 5: Example of a speedprofile with an IORE train with a signal length of 730m.

When the train should stop at a station that of course takes precedence over the speed profile. The train will brake to zero and wait according to the stations list before going back to the speed profile.

If the speed of the train is equal to the speed reference the acceleration will be set to zero. Under some circumstances the train might not be able to keep the speed constant. For example in a large uphill the speed of the train decreases if the desired tractive effort is smaller than the available.

When the speed of the train is lower than the speed reference the train will try to accelerate with the acceleration given in the train data. This will give the desired tractive effort as eq. 1. If any of the limits to the tractive effort is lower than the desired tractive effort the acceleration, a (m/s^2), will be set as

$$a = \frac{F_{\lim} - F_r}{m_a} \tag{11}$$

where $F_{\lim}$ is the lowest of the limitations to the tractive effort (kN), F_r is the total running resistance (kN) and m_a is the dynamic mass of the train (tons).

If the speed of the train is higher than the speed reference the train will brake with the retardation given in the train data. This retardation can always be kept. Trains that have the possibility to use regenerative braking will do so and if this is not enough to keep the desired retardation mechanical brakes will be used as well. Trains that are not capable of using regenerative braking will only use mechanical braking.

To calculate the speed of the train the acceleration is integrated and to calculate the travelled length the speed is integrated. When an initial speed is set the speed is increased from zero to the initial speed much faster than in a normal acceleration. If the train started at the initial speed it would lead to convergence problems.

2.4.2.2 Impact of the track layout

When the train enters a curve, a change in the adhesion or a tunnel the new values are chosen when the travelled length plus the distance between the front of the train and the mass centre are greater than the position specified in the input data for the change. The required tractive effort to overcome these obstacles is calculated as in section 2.1.

When calculating the gradient the mass of the train is distributed over two times the distance between the front of the train and the mass centre. When the distributed mass is on two or more different gradients at the same time the resulting gradient, *grad*, is calculated as

$$grad = \frac{grad_1 - grad_2}{3.6 \cdot 2 \cdot mcDist} \int v \ dt \tag{12}$$

where $grad_1$ is the gradient at the front of the train and $grad_2$ is the gradient at the rear of the train where both are measured in permillage. *mcDist* is the distance between the front of the train and the mass centre in meters and v is the speed of the train in km/h. The resulting gradient will gradually change from $grad_2$ to $grad_1$. If the front and rear of the train is on the same gradient eq.12 is not used, instead *grad* is simply set to the value of $grad_1$ and $grad_2$. The tractive effort needed to overcome the slope is calculated as in section 2.1.1.2.

2.4.2.3 Electrical calculations

The real, $u_{re,pu}$ and imaginary, $u_{im,pu}$, parts of the pantograph voltage are not calculated in the train models. Those calculations are part of the main feeding system and are handled by SIMPOW but they are available in the DSL-process. To calculate the magnitude of the voltage, *utrain* (kA), the following equation is used:

$$utrain = \sqrt{u_{re,pu}^2 + u_{im,pu}^2} \cdot U_{base} \tag{13}$$

where U_{base} is the base voltage in kA.

The active power , $P_{traction}$ (MW), is calculated from the tractive effort, speed, efficiency and no load losses in the following way

$$P_{traction} = \left(\frac{F \cdot v / 3.6}{1000 \cdot eff / 100} \right) + p0 \tag{14}$$

where F is the tractive effort in kN, v is the speed in km/h, *eff* is the efficiency in percent at the current speed given in the DYNPOW file and $p0$ is the no load losses in MW also given in the DYNPOW file. This will give $P_{traction}$ in MW, if the voltage at the pantograph is below nominal voltage $P_{traction}$ might be reduced depending on the train data which will lead to a reduction of the acceleration. The same parameters are also used when calculating the regenerated power during braking for the trains with an induction motor. The thyristor train model is not capable of regenerative braking. The total power is the sum of the traction power and the auxiliary power specified in the train data. The auxiliary power is the power needed for air condition, lighting etc.

For trains with an asynchronous motor the reactive power at motoring is calculated from the active power and information given in the train data in the following manner:

$$Q = \left(P_{traction} + P_{AUXB}\right) \cdot \tan\varphi + P_{AUXT} \cdot \frac{\sqrt{1 - pF_{AUXT}^2}}{pF_{AUXT}} \tag{15}$$

where Q is the reactive power in MW, $P_{traction}$ is the active power used for traction in MW, φ is the phase angle given in the train data as a function of the voltage, P_{AUXB} is the auxiliary power taken from the bridge (MW), P_{AUXT} is the auxiliary power taken directly from the main transformer (MW) and pF_{AUXT} is the power factor for P_{AUXT}.

When the train has stopped, Q is calculated in the following way:

$$Q = P_{AUXB} \cdot \frac{\sqrt{1 - pF_{AUXB}^2}}{pF_{AUXB}} + P_{AUXT} \cdot \frac{\sqrt{1 - pF_{AUXT}^2}}{pF_{AUXT}} \tag{16}$$

where the new parameter pF_{AUXB} is the power factor for P_{AUXB} when the motors are not used.

When the thyristor train is above base speed the reactive power is calculated based on the active power and data given in the DYNPOW file in the following way:

$$Q = P_{traction} \cdot \frac{\sqrt{1 - pF_{traction}^2}}{pF_{traction}} \tan\varphi + Q_{FILTER} + P_{AUXT} \cdot \frac{\sqrt{1 - pF_{AUXT}^2}}{pF_{AUXT}} \tag{17}$$

where $pF_{traction}$ is the power factor for $P_{traction}$ when the train is above base speed, Q_{FILTER} is the reactive power consumption of the train filters (MW) and the other variables are the same as above. When the speed is lower than the base speed the number of bridges has to be taken in to account resulting in more complicated calculations.

The real, $i_{re,pu}$, and imaginary, $i_{im,pu}$, parts of the current are solved by using the calculated values of P and Q and the real, $u_{re,pu}$, and imaginary, $u_{im,pu}$, parts of the pantograph voltage. The equations use implicit assignments for the currents and are shown below.

$$i_{re,pu} : u_{re,pu}(tC \cdot \frac{d}{dt} i_{re,pu} + i_{re,pu}) + u_{im,pu}(tC \cdot \frac{d}{dt} i_{im,pu} + i_{im,pu}) = -\frac{P}{S_{base}} \tag{18}$$

$$i_{im,pu} : u_{re,pu}(tC \cdot \frac{d}{dt} i_{im,pu} + i_{im,pu}) + u_{im,pu}(tC \cdot \frac{d}{dt} i_{re,pu} + i_{re,pu}) = \frac{Q}{S_{base}} \tag{19}$$

The equations are written in the above way, as differential equations, because it will make changes in the current slower to avoid convergence problems. tC is a time constant stated in the train data. Since the voltage and current are in pu:s the active and reactive power has to be divided by the base power S_{base} (MW). The signs have been changed in the equation because of the way SIMPOW defines current directions. The calculated current is then set in the feeding system.

To calculate the magnitude of the current, *itrain* (kA), the following expression is used:

$$itrain = \sqrt{i_{re,pu}^2 + i_{im,pu}^2} \cdot \frac{S_{base}}{U_{base}} \qquad (20)$$

2.4.2.4 Structure of the code

A general outline of the structure of the code where the most important parts are explained follows. This applies to both train models when nothing else is written.

Variable declarations

Variables are declared and the initial values of integer state variables are set.

Initialization

The start values of real state variables and changeable constants are set. Calculates the gradient at the trains starting position

Track changes

Controls changes of the allowed speed, the gradient, the radiuses of curves, the tunnelfactor and the adhesion coefficient. All the changes are applied when the very important relation *if (tlength .gt/5/. tlengthNext)* changes its truth value. The variable *tlengthNext* has the value of the position of the next track change and *tlength* is the travelled length. The deadband is multiplied by 5 to gather track changes that are close to each other.

Precalculation of the braking curve

The braking curve for the next lower speed is precalculated. The code is run when the train starts and after a brake is completed if the train is not braking for a station. If it is, then the calculation is done when the train leaves the station.

Get values from tables and calculate the pantograph voltage

Several variables get their value from tables where the variable is a function of the speed or the pantograph voltage. Examples of the variables are the efficiency and the motoring effort as a function of the speed and the allowed tractive effort as a function of the voltage. This is done with spline if the speed or voltage is inside the boundaries of the table, otherwise it is set as the lowest or highest value. The code for these routines are similar to the code shown in Appendix B:changes to the code, *Spline in the braking algorithm.*

If the square of the magnitude of the voltage in pu, $u_{re,pu}^2 + u_{im,pu}^2$, is less than or equal to zero *utrain* is set to zero otherwise it is calculated as in eq. 13. During iterations variables can get strange values, therefore it is necessary to do this check.

Calculate train resistances

The gradient is calculated (see 2.4.2.2) as well as the gradient resistance (see 2.1.1.2), curve resistance (see 2.1.1.3) and the running resistance (see 2.1.1.1).

Braking algorithm

Checks if the train has reached the precalculated braking curve. If it has the speed reference changes from the allowed speed to the new lower speed. Also controls when the train should leave a station it has stopped at (see 2.4.1.2).

Check if a speed change is necessary

The speed is compared to the speed reference and the train accelerates, brakes or keeps a constant speed accordingly (see 2.4.2.1).

Limitations and active power

Computes limitations and compares the desired tractive effort with the limitations. The limitations are *FLIMIT1*, *FLIMIT2*, limitations because of the voltage level, adhesion and the maximum force from the motor. The tractive effort is decreased to the lowest of the limits. See 2.1.3 for more information about limitations and Appendix B: Changes to the code, *Limitiations to the tractive effort*. Also computes the active power (see 2.4.2.3).

Calculate speed and travelled length

This is calculated as in section 2.4.2.1 when the train is moving. If the train has stopped the acceleration and speed are set to zero and the travelled length is kept constant.

Calculate the power angle and the reactive power

See 2.4.2.3

Calculate the currents

Uses eq.18 and 19 for the real and imaginary parts of the current.

The code for the calculation of the magnitude of the current is similar to the calculation of the pantograph voltage with the exception that eq. 20 is used.

3 Analysis

3.1 Method

Some small changes were done to the original models before weak relations were implemented. When implementing weak relations several small simulations were run to see the effect. The final weak relations models and ordinary relations models were then used in two simulations and the result was compared. The first simulation was created by BBrail to study possible changes to the power supply system for a part of Malmbanan. In the second simulation more trains were added to the first simulation to see how weak relations perform in a simulation with more events.

Two step sizes were used; 0.1s which is normally used in TracFeed simulations and 1s which is a large step size.

The data that was used when comparing the results was the magnitude of the current. Since it is a parameter that changes relatively fast and it is through the calculated value of the current the trains have an effect on the rest of the simulated system this is the most critical parameter.

3.2 Initial Changes to the code

Before implementing weak relations some changes were done to the code. After a train had finished braking to a lower speed it would accelerate for a short while and then brake to the lower speed again as shown in figure 6. This causes errors and unnecessary events.

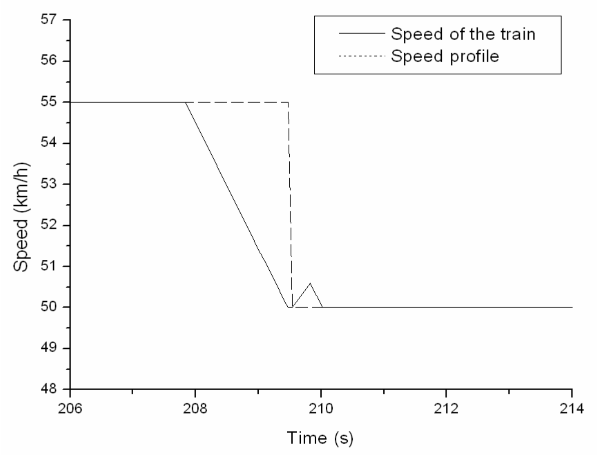

Figure 6: Speed bug

This happened because of an inconsistency in the deadband of two relations. The first relation is *if (tlength .gt/5/. tlengthNext)* controlling the track changes (see 2.4.2.4) and the second is *if (tlength .gt. brakeToS)* in the braking algorithm (see 2.4.2.4) where both should change to true at the same time when the train has finished braking. When the relations change to true the speed reference should change from being set in the braking algorithm to being set in the track changes block. Since the deadband is multiplied by five in the first relation this does not happen. To solve this problem the deadband is multiplied by five in the second relation as well.

The code that sets the limitations to the tractive effort functioned in the following way:

```
If (limA < fdesir)
  fdesir = limA
if (limB < fdesir)
  fdesir = limB
and so forth…
```

where *limA* and *limB* are limits to the tractive effort and *fdesir* is the desired tractive effort. If for example *limA* becomes lower than *fdesir* but *limB* is lower than *limA* the step would be truncated. This is unnecessary since only the lowest limit is interesting. This code was rewritten so that only a change of the lowest limiting factor will lead to a truncation (see Appendix B: Changes to the code, *Limitations to the tractive effort*). This removes some truncations and leads to slightly shorter simulation times.

In the braking algorithm linear interpolation was used to calculate the retardation used in the precalculation of the braking curve. This was changed so that spline is used instead (see appendix B: Changes to the code, *Spline in the braking algorithm*).

In the thyristor train model the regenerative force at braking was calculated but not used. The code had simply been copied from the asynchronous train model. This was removed because the thyristor train does not have the ability to use regenerative braking.

3.3 Implementing weak relations

When implementing weak relations only relations containing real variables are of interest (see 2.3.3.3). Relations that check if a variable is eligible for a mathematical function such as square root should not be made weak.

It is interesting to note which relations change their truth values the most times and thereby causes the most truncations. This of course depends on the simulation but one general trend is noticeable. Since most simulations have a lot of track details the statement that leads to far more truncations than any other is the relation *if (tlength .gt/5/. tlengthNext)*. Other relations tend to cause at most one tenth as many truncations.

The structure of the code shown in 2.4.2.4 is used in this section and the most important relations will be mentioned. The impact because of weak relations were first tested in the small simulation shown in Appendix A: Input data for a small simulation (that specific setup was used to test the train model with an asynchronous motor).

Variable declarations

Contains no real relations.

Initialization

Contains no real relations.

Track changes

If the relation *if (tlength .gt/5/. tlengthNext)* is changed to a weak relation it means that all track changes will be applied at the end of steps. Since most simulations have a very detailed gradient data, a train will always be on two or more different gradients at the same time and the gradient will be calculated as eq.12. The weak relations changes the point in time when the gradients change and this will create a constant error in the resulting gradient. Because of that two sets of weak relations models were used, one where *if (tlength .gt/5/. tlengthNext)* was made weak and one where the code was rewritten so that a change in the gradient truncates the step.

Precalculation of the braking curve

No weak relations were implemented here since it can lead to that trains miss stations and also runs beyond the specified track.

Get values from tables and calculate the pantograph voltage

Weak relations were implemented in the relations that check if the speed or voltage is within the boundaries of the tables. Usually the most common speeds and voltages are defined in the tables so those relations don't cause many truncations. Weak relations were not implemented in the calculation of the voltage since the relation checks if square root can be used.

Calculate train resistances

Contains no real relations.

Braking algorithm

No weak relations were implemented here for the same reasons as in *Precalculation of the braking curve*.

Check if a speed change is necessary

Weak relations were not implemented because it can lead to the train switching back and forth between acceleration and braking as shown in figure 7. Simulations have also shown that trains can miss stations.

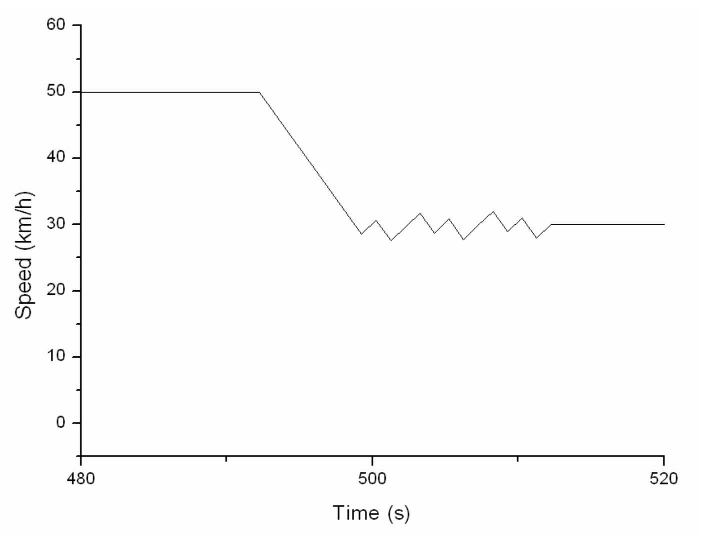

Figure 7: Speed problems because of weak relations

Limitations and active power

This part of the code contains three relations (two for a thyristor train) that usually cause a lot of truncations. The first checks if the total tractive effort is larger than or equal to zero which is necessary when applying the limitations. The second finds the smallest limitation when the total tractive effort is larger than or equal to zero (acceleration or a constant speed). See Appendix B: Changes to the code, *Limitations to the tractive effort* for this relation. For the train with the induction motor there is also a relation that finds the smallest limititation when the total tractive effort is smaller than zero (braking/regeneration). There are also some real relations in the calculations of the limitations but they usually cause a lot less truncations. All of the relations in *Limitations and active power* were made weak.

Calculate speed and travelled length

Has a relation that checks if the speed is greater than zero or if the acceleration is greater than or equal to zero where especially the acceleration part leads to many truncations. It was made weak. Weak relations were not implemented in a relation checking if the initial speed has been reached, since that can make the speed too high during the initial very fast acceleration.

Calculate the power angle and the reactive power

For the train with an asynchronous motor weak relations were implemented in relations similar to those in *Get values from tables*. The tables have the phase angle as a function of the voltage. For the thyristor train, several relations were made weak in the calculation of the reactive power when the speed is below base speed, except for a relation checking if a variable is negative before a square root function.

Calculate the currents

Weak relations were not implemented in the calculation of the magnitude of the current because the relation checks if square root can be used.

3.4 Test simulations

Two simulations were run to compare weak relations to ordinary relations. The first simulation is a simulation BBrail did for Banverket. In this thesis it will be called Malmbanan. The same track data was used in the second simulation but the number of trains was increased.

3.4.1 Malmbanan

Banverket asked BBrail to examine an alternative feeding system for the part of Malmbanan going between Gällivare and Riksgränsen. The converter stations in Kiruna and Stenbacken are closed and there is a new AT system between Kiruna and Tornedalen as well as a new converter station in Råtsi[11].

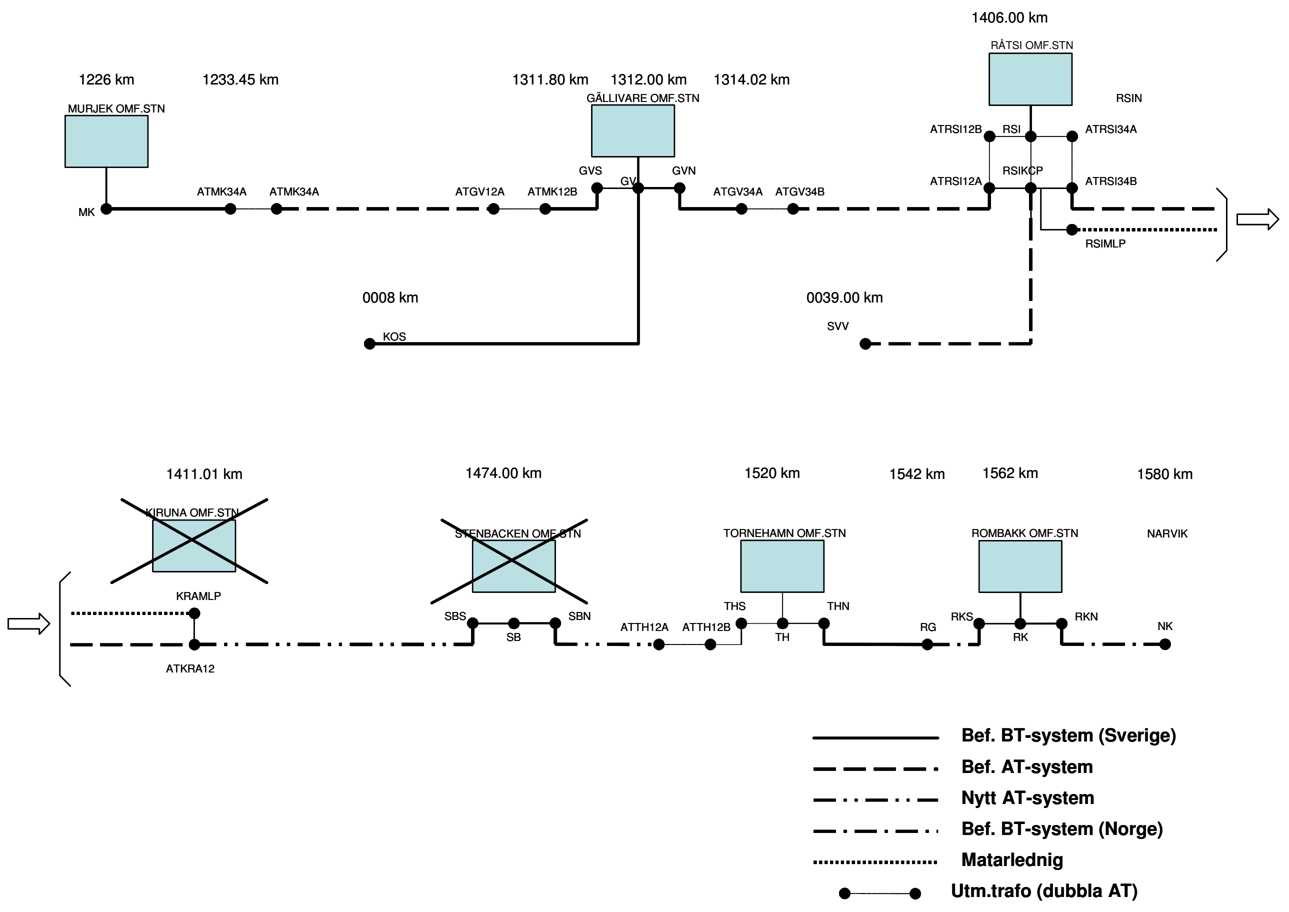

Figure 8: Schematic picture of the Malmbanan simulation[11]

The figure above shows the name and position of converter stations and the nodes in the simulation. Also shown is the type of feeding system between the nodes (in Swedish).

The simulation starts at 00:00 and ends at approximately 18:00. Trains are active between 12:00 and the end of the simulation which corresponds to the daily period when the traffic is heaviest. There are a total of 22 trains with 9 trains running simultaneously at most.

In the simulation there are eight loaded and seven empty IORE trains, three RC4 and four RC6 trains. The trains run according to a timetable supplied by Banverket who also supplied the input data regarding the track, the power supply system and some of the train data. Information regarding the IORE train came from Bombardier.

This simulation was chosen because it is not so large so that an overall picture is hard to form and it is not so small so that weak relations won't have an impact on the response time.

3.4.2 Modification of Malmbanan

The track data used in the Malmbanan simulation were also used in this simulation. The number of trains was increased to 66 and they start every 10 seconds with the first train starting at 100 seconds. There are three times as many trains for all train types in this simulation. Since the trains start with such a short interval there are 66 trains running simultaneously at most. The trains use the same drive path as before but the converter stations were replaced by infinite busses because of convergence problems. The trains pass each other on the same track in this simulation, but that will only generate a warning.

By having this large number of trains the time between events decreases which give the possibility for more events to be gathered at the end of steps when using weak relations.

3.5 Simulation Results

The simulations that was run with ordinary relations is called OR and the simulations with weak relations is called WR1 and WR2. WR1 uses the models where a change in the gradient causes a truncation and WR2 uses the models where it doesn't.

The time it took to run the simulations and the total number of steps was recorded. The time is the most important parameter but it is interesting to note how the number of steps decreases by using weak relations.

To compare the currents a C program (see Appendix C: C program) was written that uses two sets of output files from SIMPOW, one set of files from OR simulations and one from WR1 or WR2 simulations. In the output files the current is a function of the time and the program computes the average difference in the magnitude of the current between the first train from set one and the first train from set two, then the second trains and so forth. This is done when the trains are active and in two different ways.

The first algorithm finds matching current values (y-values) by using linear interpolation, this is necessary because different simulations have output data at different points in time. Figure 9 shows an example of this. The average difference between the current values for one train from OR and the corresponding train from WR1 or WR2 is calculated, this is called the LI current error. The program also adds the LI current errors from all the trains in the set and computes the average error, this is called the Total LI current error.

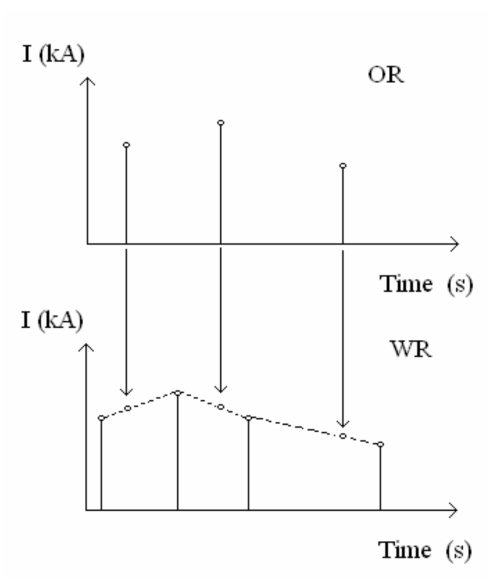

Figure 9: Example of linear interpolation

The second calculation is done by simply calculating the average value of the current for a train from OR and subtracting the average value of the current from the corresponding train from WR1 or WR2 and the absolute value of this difference is called the AVG current error. The AVG current errors are added for all trains in the set and the average value is calculated which is called the Total AVG current error. Since the main problem with the output data is a displacement in time (see later in the report) it is interesting to see how much smaller the AVG current error is compared to the LI current error.

Because of the displacement in time the program also keeps track of the time when trains have finished their paths and computes the difference between the trains in OR and WR1 or WR2, this is called the Time error. A train has finished its path when the last sequence of zeros starts in the output file. The average value of this difference for all trains is computed as well, this is called the Total time error.

The basic step sizes chosen for the simulations was 0.1s and 1s. The reason for choosing the smaller step size was that it is the step size most often used in TracFeed simulations. In a simulation with ordinary relations and a very complex feeding system as well as many trains the response time is usually smaller with a step size of 0.1s instead of 1s [12]. The larger step size combined with weak relations is the most interesting alternative because it gives the possibility to gather more events at the end of steps when using weak relations, leading to a larger reduction in the response time. The maximum step size was set to 50s in all simulations.

3.5.1 Malmbanan results

There are no trains running during two thirds of this simulation but the step size increases to the maximum step size in that part of the simulation so the last third of the simulation is reached fast.

	H=0.1			H=1		
	OR	WR1	WR2	OR	WR1	WR2
Steps	182 457	180 134	178 631	39 579	34 808	26 593
Time (s)	1 754.10	1 716.01	1 705.44	489.62	433.52	373.73
ΔTime (%)	-	2.17	2.77	-	11.46	23.67
Largest LI current error (A)	-	28.46	23.49	-	59.57	68.85
Total LI current error (A)	-	6.04	6.89	-	9.79	18.97
Largest AVG current error (A)	-	5.31	4.81	-	7.91	18.81
Total AVG current error (A)	-	1.07	1.40	-	1.55	3.84
Largest time error (s)	-	13.86	7.87	-	10.43	37.73
Total time error (s)	-	3.12	2.85	-	4.27	9.49

Table 3: Malmbanan results

$\Delta Time$ is the gain in time in percent when comparing weak relations simulations to ordinary relations simulations with the same step size. This is computed as

$$\Delta Time = \frac{Time_{OR} - Time_{WRx}}{Time_{OR}} \cdot 100$$

where $Time_{WRx}$ is the time for WR1 or WR2.

The largest LI, AVG and time errors are the largest error one train had with WR1 and WR2 compared to OR.

There are too few events in this simulation for there to be any significant decrease in time when using a step size of 0.1s.

When using the larger step there are naturally more events that can be gathered at the end of steps leading to a larger decrease in time for especially WR2 but the cost for this is rather high. The output data have been displaced a lot in time which figure 10 illustrates. The figure shows the train that had the largest time error with WR2 and H=1 and also the corresponding trains from OR and WR1. When the current reaches 0 kA at around 64450s the train has finished its path.

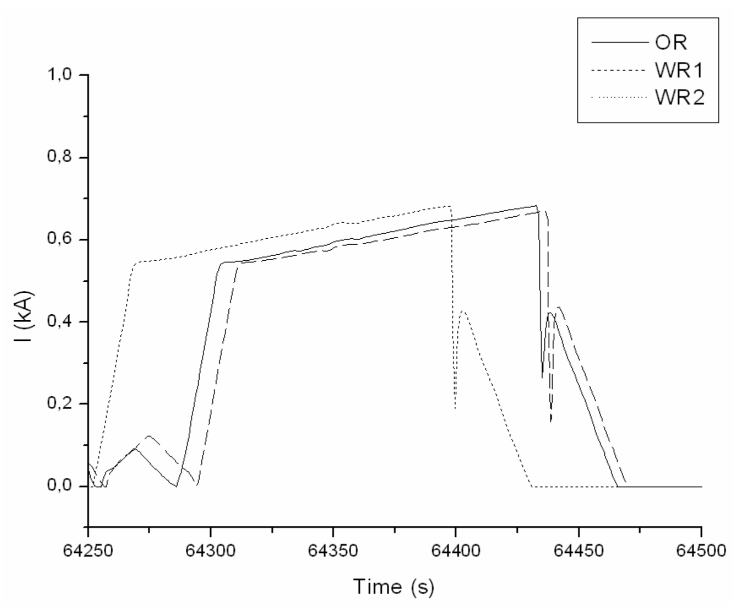

Figure 10: Example of a displacement in time because of weak relations.

For the smaller step size it was pointless to make a change in the gradient truncate the step since the errors was very similar to WR2 when using the same step size. However with the larger step it has paid off when comparing the current errors and time displacement but the gain in response time is halved. The gradient table is detailed therefore changes in the gradient causes many truncations.

The time gained for the different setups are not very impressive, especially if the decrease in response time is weighed against the deviation of the output.

3.5.2 Modified Malmbanan results

	H=0.1			H=1		
	OR	WR1	WR2	OR	WR1	WR2
Steps	184 248	174 986	156 813	51 163	41 437	25 762
Time (s)	4 635.63	4 436.21	3979.35	2 035.90	1 695.39	1 143.70
ΔTime (%)	-	4.30	14.16	-	16.73	43.82
Largest LI current error (A)	-	23.37	23.01	-	52.79	90.78
Total LI current error (A)	-	4.58	5.06	-	11.76	17.43
Largest AVG current error (A)	-	4.83	4.91	-	14.51	23.76
Total AVG current error (A)	-	0.59	0.58	-	1.68	3.17
Largest time error (s)	-	7.05	10.76	-	32.70	52.03
Total time error (s)	-	2.02	2.00	-	6.56	9.47

Table 4: Modified Malmbanan results

Since there were more events in this simulation the gain in response time is larger for all different simulation setups. As in the Malmbanan simulation, the decrease in response time when using WR1 is smaller than when using WR2 and with the larger step size WR1 is more accurate than WR2.

Figure 11 shows the train that had the largest time error with WR2 and H=1s and also the corresponding train from OR. The curves have been adjusted in time so they overlap each other and a smoothing of 1 minute has been done to each curve. The error in the current is now rather small so the main problem with weak relations lies in the displacement in time. The LI current error has decreased from 66.79 A to 15.94 A. By visually inspecting the curves from other trains it is obvious that the displacement in time is by far the worst problem. It should be noted that the train in this example operates for around 11 000 seconds so in comparison a time error of 52 seconds is not very much.

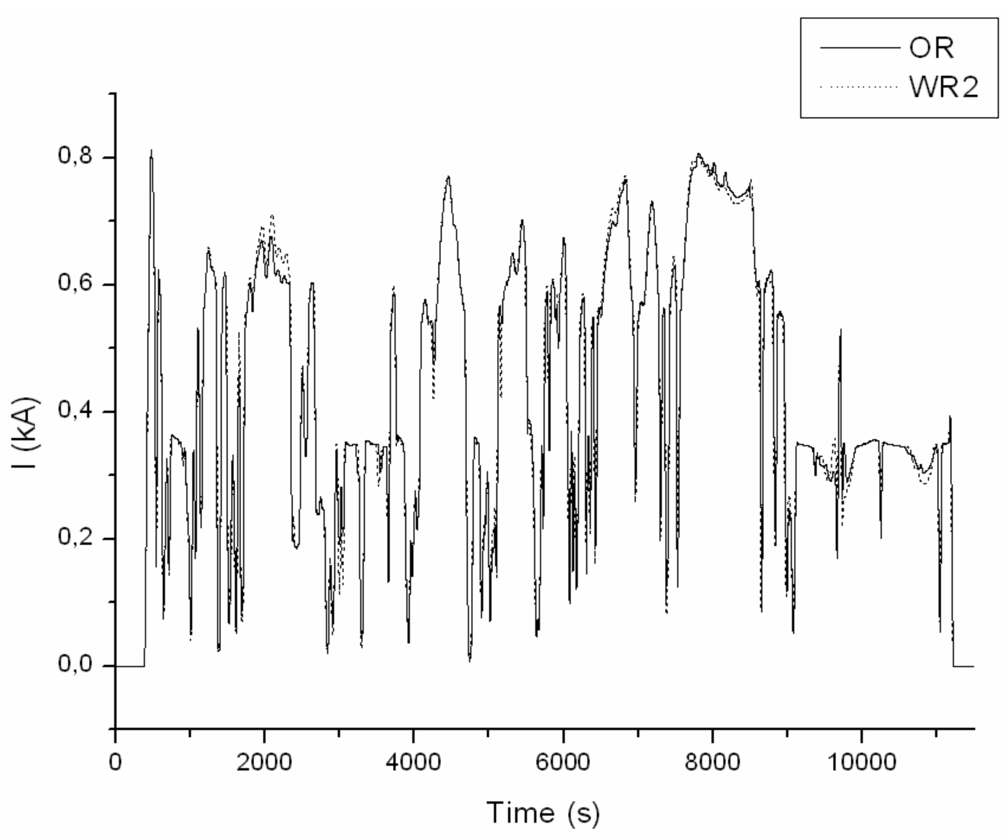

Figure 11: Adjusted curves

If the displacement in time of the output is acceptable then WR2 together with H=1s is an alternative in a simulation with this many trains. For the other setups the gain in time is too small to warrant the use of weak relations.

3.6 Conclusions and Recommendations

The problem with the displacement in time arises because weak relations alter the point in time for truth value changes which means the length of certain intervals changes. If for example intervals with a high allowed speed are made longer and intervals with a large gradient are made shorter the train will go faster and things will happen too soon. These small errors are added and for some trains there will be a major displacement in simulated time. This is most obvious when using a larger step size as the change in the truth value of relations can move more in simulated time. When using a smaller step size the error tends to be more constant which leads to a smaller deviation of the output.

When using a step size of 0.1s the gain in time is so small with weak relations that ordinary relations might as well be used.

Even when the step size is as large as 1s a lot of trains are necessary for weak relations to cause a significant decrease in the response time but as previously stated there will be problems with the output. The deviation in the output decreases when using WR1 instead of WR2, but the gain in time is so severely decreased that there is no point in using WR1.

WR2 should be used with caution since it leads to a rather large error in the output. The error is mainly caused by a displacement in time. If the curves are adjusted so that the time displacement disappears it is obvious that the error in the current is small. This means that most things happen as they should, but at the wrong time.

If the displacement in time is acceptable to the user then weak relations are an interesting alternative in large simulations.

4 Thesis summary

This thesis has presented theory concerning electrical traction basics, the resistance trains meet and the limitations to the tractive effort. The simulation program SIMPOW has been described and how TracFeed uses it to simulate trains coded as DSL models. The concept of weak relations has been explained as well as other important features in the dynamic system language. The input data required for a simulation has been mentioned and how TracFeed models trains.

By making relations weak their truth value can only change at the end of steps. This will decrease the number of steps and thus decrease the computation times of simulations. The objective of this thesis was to examine what impact weak relations have on the output from a simulation and how much the response time could be decreased by using weak relations in different simulations.

Some changes were done to the code before implementing weak relations. When implementing weak relations a small simulation were run to see if any major problems were encountered. The final weak relations models were then tested in two larger simulations. The first simulation was a simulation BBrail did for Banverket. The same track layout was also used in the second simulation but a lot more trains were added to create more events and thereby see how much time that could be gained in a larger simulation with weak relations. The step sizes chosen for the simulations was 0.1s and 1s.

The results from the first simulation show that the time gained when using weak relations and the smaller step size is negligible. When the larger step size was used more time was gained at the cost of a rather large displacement of the output for some trains.

The decrease in response time when using weak relations in the second simulation was larger as it should be because of more frequent events but the time gained with the smaller step size was still not very impressive. With the larger step size weak relations can decrease the time it takes to run this simulation by about 40% but this leads to a large displacement in time of the output for some trains. If this is acceptable to the user then weak relations are an interesting alternative.

5 Further Work

In addition to the simulations run for this thesis it would be interesting to test weak relations in a simulation with a highly complex feeding system and a large number of trains. Such simulations often take days to run so a decrease in the response time would be very positive.

A decision has to be made concerning what an acceptable time displacement is.

6 References

[1] SETRA/ROD (1997) *Adtranz teknikutbildning Stationära utrustningar*, Adtranz AB

[2] Östlund, Stefan (1996) *Elektrisk traktionsteknik,* Kungliga Tekniska Högskolan

[3] **http://www.jernbane.net/norge/el/el18/index.asp**, 2007-12-11

[4] **http://www.twi.co.uk/j32k/protected/band_3/ksgbm001.html**, 2007-12-11

[5] **http://www.jarnvag.net/lokguide/Iore.asp**, 2007-11-22

[6] **http://www.jarnvag.net/lokguide/Rc4.asp**, 2007-11-22

[7] **http://www.jarnvag.net/vagnguide/X50.asp**, 2007-11-22

[8] Ström, Mikael (2006) *TracFeed Education*, Balfour Beatty Rail AB

[9] Mohan, Ned et al. (2003) *Power Electronics 3rd edition*, John Wiley & Sons Inc.

[10] STRI (2004) *SIMPOW User Manual*, STRI AB

[11] Stern, Jari (2004) *Systemstudie Kraftförsörjning Gällivare-Riksgränsen,*
Balfour Beatty Rail AB

[12] Discussions with Mikael Ström, 2008-01-09

Appendix A: Input data for a small simulation

OPTPOW file

```
CONTROL DATA
  PHASE=1                                                        Number of phases
END

GENERAL
  SN=5                                                           Base power in MVA
END

NODES
  A           UB=15                                              Node name and base voltage in kV
  C           UB=15                                                          -||-
END

POWER CONTROL
  A           TYPE=NODE    RTYPE=SW     U=16.5      FI=0         Source type, initial voltage and phase angle
  C           TYPE=NODE    RTYPE=UFI    U=16.5      FI=0                       -||-
END
END
```

DYNPOW file

```
                                                    ***SIMPOW data***
CONTROL DATA
  TEND=930                                          End time of simulation
  H=1                                               Step size
  HMAX=50                                           Maximum step size
END

GENERAL
  NREF=1                                            Nr. of reference nodes
  REF=A                                             Name of reference node
  FN=16.666667                                      Frequency of the system
END

NODES
  A TYPE=1                                          Type of node (1 -> constant voltage and phase)
  C TYPE=1                                                  -||-
END
                                                    ***TracFeed data***
TRAINS
ASYNCTRAIN IORE_TOMT                                Model of an empty IORE asynchronous train
{
   DYNMASS      1750                                Dynamic mass (tons)
   MASS         1720                                Static Mass (tons)
   ADHMASS       360                                Adhesion mass /tons)
   MAXSPEED       70                                (km/h)
   SLENGTH       730                                Signal length (m)
   RRA       38.788                                 Const.  used when calc. the running resistance
   RRB        0.03889                                       -||-
   RRC        0.00774                                       -||-
   ADH1        7.5                                  Const. used when calculating adhesion
   ADH2       44                                            -||-
   TC          1.0                                  Time constant used when calculating current
   UNOM       15                                    Nominal voltage of the train (kV)
   PAUXB       0.24                                 Auxiliary power from the bridge (MW)
   PFAUXB      0.8                                  Phase angle of PAUXB
   PAUXT       0                                    Auxiliary power from the transformer (MW)
   TRAINLIMIT1 39600                                Max. value  of F*v (kN*km/h)
   TRAINLIMIT2   433                                F at max speed (kN)
```

In the lists the independent var. is on the left

```
ACCREFV
{
  0 A 1.0 R 0.4 ;
 70 A 1.0 R 0.4 ;
}
```

Acc. and retardation reference(m/s^2) as a func. of the speed (km/h)

```
FIU
{
 12           FIMOT  -5    FIBRAKE  6     ;
 12.061       FIMOT  -5    FIBRAKE  6.03   ;
 12.121       FIMOT  -5    FIBRAKE  6.061  ;
 …
 17.879       FIMOT  0     FIBRAKE  0     ;
 17.939       FIMOT  0     FIBRAKE  0     ;
 18           FIMOT  0     FIBRAKE  0     ;
}
```

The phase angle (degrees) of the train as a func. of the catenary voltage (kV) at motoring and braking. Only parts of the list shown.

```
FVTRAINMAX
{
 0            FMOT  1350     FELBRAKE  750   EFF  40     ;
 0.808        FMOT  1350     FELBRAKE  750   EFF  42.667  ;
 1.616         FMOT  1350     FELBRAKE  750   EFF  45.333  ;
 …
 78.384       FMOT  1200     FELBRAKE  750   EFF  85     ;
 79.192       FMOT  1200     FELBRAKE  750   EFF  85     ;
 80           FMOT  1200     FELBRAKE  750   EFF  85     ;
}
```

Max. tractive effort, braking force (kN) and efficiency as a func. of the speed (km/h). Only parts of the list shown.

```
 FUSUPPLYLIM
 {
  11.0        FMOT  1350  ;
  18.0        FMOT  1350  ;
 }
```

Max. tractive effort (kN) as a func. of the catenary voltage (kN)

```
 PUSUPPLYLIM
{
  9           PMOT  0     PELBRAKE  0       ;
 9.086        PMOT  0.279  PELBRAKE  0.197    ;
 9.172        PMOT  0.558  PELBRAKE  0.395    ;
 …
 17.328       PMOT  13     PELBRAKE  2.347    ;
 17.414       PMOT  13     PELBRAKE  1.173    ;
 17.5         PMOT  13     PELBRAKE  0       ;
}
}
END
```

Max. power consumption/regeneration (MW) as a function of the catenary voltage (kV). Only parts of the list shown.

TRACK

```
A  C    SECTION  AtoC                                      Nodes at the  start and end of the section
{                                                          and the name of the section
 STARTPOS        0                                         Start position of the section
 ENDPOS          13                                        End position of the section
 UB              15                                        Base voltage of the section
 ROVER           0.2                                       Resistance of the catenary (ohm/km)
 XOVER           0.2                                       Inductance of the catenary (ohm/km)

LAYOUT                                                     Layout of the track
{                                                          GRAD -> gradient (permillage)
0  GRAD=0   CURVE=0   ADHCOEFF=0.3   TUNNEL=1  ;           CURVE -> curve radius (m)
3  GRAD=2                                      ;           ADHCOEFF -> Adhesion coeff.
4               CURVE=5000                     ;           TUNNEL -> Tunnel factor
5  GRAD=6                            TUNNEL=1.8 ;          All of the above as a func. of the position (km)
6              CURVE=0                         ;
7  GRAD=0                             TUNNEL=1  ;
11             CURVE=2000                      ;
12             CURVE=0                         ;
}

SPEEDPROFILE    AtoCspeed                                  Name of the speed profile
{                                                          Allowed speeds (km/h) as a func. of
0  U   D   SPEED=60  ;                                     the position (km)
3  U   D   SPEED=50  ;                                     U -> Up track
7  U   D   SPEED=70  ;                                     D -> Down track
}

STATIONS   AtoCstations                                    Name of the stations list
{                                                          Position of the stations (km)
0  U   D   WAIT=0   ;                                      WAIT -> Amount of time the train should wait
8  U   D   WAIT=50 ;                                       at the station (seconds)
13 U   D   WAIT=0   ;                                      U and D as in SPEEDPROFILE
}
}
END
```

DRIVE PATH

```
PATH AtoCpath                                                                  Name of the drive path
{
 1  SECTION = AtoC   U   SPEED = AtoCspeed   STATIONS = AtoCstations ;         Adds sections, speed profile and stations list to
}                                                                              the path
END
```

TIMETABLE

```
TRAFFIC AtoCtraffic                Name of timetable
{
 TRAIN IORE_tomt                   Type of train
 PATH  AtoCpath                    Sets path
 DEPARTURES
 {
  TRUP1 AT 0 ;                     Departure time
 }
}
END

END
```

Appendix B: Changes to the code

Limitations to the tractive effort (for the thyristor train)

If the train is in a motoring state it is either accelerating or keeping the speed constant. If standing equals one the train has stopped at a station. *min_acc* is a state variable so it keeps its value if unchanged and *Fr* is the total train resistance.

```
…
!! CHECK FOR RESTRICTIONS IN THE FORCE AND POWER:


 if ( (trainState .eq. motoringState) .and. (standing .eq. 0) ) then
   F_arr(1) = acc*dynMass + Fr                          !!desired acceleration
   F_arr(2) = fLimit1
   F_arr(3) = fLimit2
   F_arr(4) = trainFMot                                 !!maximum moment
   F_arr(5) = adhesionMass*9.81*adhesion                !! adhesion limitation
   F_arr(6) = supplyFMot                                !! force limit because of voltage level
   F_arr(7) = ((pMot-p0)*eff*36/(speed+0.001)           !! power limit because of voltage level

   for(i = 1 , 7) do                                    !!find the smallest
     if (F_arr(i) .lt. F_arr(min_acc)) min_acc = i
   endfor

   acc = (F_arr(min_acc) -  Fr)/dynMass                 !!set the acceleration

   fSum = F_arr(min_acc)
   pTmp = fSum*speed/(3600*eff/100) + p0

else !! ((trainState .eq. brakeState) .or. (standing .eq. 1))
  fSum = 0
  pTmp = p0

endif
…
```

Spline in the braking algorithm

tempV is the calculated speed (km/h) and *retRef* is the retardation (m/s^2) during braking precalculation. *retRefTab* is a table containing the retardation as a function of the speed.

```
…
if (tempV.lt.retRefTab(1,1)) then
  tempRetRef=retRefTab(1,2)
elseif (tempV.gt.retRefTab(nrow(retRefTab),1)) then
  tempRetRef=retRefTab(nrow(retRefTab),2)
else
  tempRetRef=spretRefTab(tempV)
endif
…
```

Appendix C: C program

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>

//This program uses two sets of SIMPOW data table files containing x and y values
//to compute an average and maximum difference in the y values of the
//files. If the x values do not match linear interpolation will be used.
//The difference of the start of the last zero sequence is calculated.
//The difference of the average values are computed.

int main(void)
{

  char name1[100]="table_", name2[100]="table_", name11[100], name22[100], jobid1[100], jobid2[100];
  FILE *file1, *file2, *output;
  int i=0, j=0, lenarr1, lenarr2, c, k, nrpos1, nrpos2, nr, diffzeronr=0, maxdiffzeronr=0;



  double *arr1, *arr2;

  double x1, x2, y1, y2, x, y;

  double li_diffavg, li_diffsum, li_diff, li_sumdiffavg = 0, li_maxdiffavg=0, li_totdiffavg = 0;
  int li_maxdiffavgnr=0;

  double avg_sum1 = 0, avg_sum2 = 0, avg_diffavg = 0, avg_maxdiffavg = 0, avg_sumdiffavg = 0, avg_totdiffavg = 0;
  int avg_maxdiffavgnr = 0;

  double lastzero1=-1, lastzero2=-1;
  double diffzero=0, sumofdiffzero=0, maxdiffzero=0, avgdiffzero = 0;

  int start, inactive_start, inactive_end;



  output = fopen("Results", "w");                //open/create output file

  printf("Enter the job ID for file 1: ");
  scanf("%s", jobid1);
  printf("Enter the job ID for file 2: ");
  scanf("%s", jobid2);

  printf("Enter the nr of files to compare: ");
  scanf("%d", &nr);
```

```
strcat(name1, jobid1);
strcat(name2, jobid2);

nrpos1 = strlen(name1)+3;
nrpos2 = strlen(name2)+3;

strcpy(name11, name1);
strcpy(name22, name2);
strcat(name1, "_c_x.txt");
strcat(name2, "_c_x.txt");
strcat(name11, "_c_xx.txt");
strcat(name22, "_c_xx.txt");

for(k=1 ; k <= nr ; k++ )                //increases the file number
{
 i=0;
 j=0;
 li_diffsum=0;
 avg_sum1 = 0;
 avg_sum2 = 0;
 start = 0;
 inactive_start = 0;
 inactive_end = 0;

 if(k < 10 )                       //If the file number is smaller than 10
 {
   name1[nrpos1] = k + 48;
   name2[nrpos2] = k + 48;
   printf("\nfile1: %s\nfile2: %s\n", name1, name2);
   if((file1 = fopen(name1, "r")) == NULL)
   {
     printf("\nError: Cannot open %s\n", name1);
     system("PAUSE");
     return 0;
   }
   if((file2 = fopen(name2, "r")) == NULL)
   {
     printf("\nError: Cannot open %s\n", name2);
     system("PAUSE");
     return 0;
   }
 }

 else                              //If the file number is larger than 9
 {
   name11[nrpos1] = (k - k%10)/10 + 48;
```

```
  name11[nrpos1+1] = (k%10) + 48;
  name22[nrpos2] = (k - k%10)/10 + 48;
  name22[nrpos2+1] = (k%10) + 48;
  printf("\nfile1: %s\nfile2: %s\n", name11, name22);

  if((file1 = fopen(name11, "r")) == NULL)
  {
    printf("\nError: Cannot open %s\n", name11);
    system("PAUSE");
    return 0;
  }
  if((file2 = fopen(name22, "r")) == NULL)
  {
    printf("\nError: Cannot open %s\n", name22);
    system("PAUSE");
    return 0;
  }
}

while( (c=getc(file1)) != EOF )
{
  if(c == '\n')
    i++;                          //counts nr of rows in file1
}
lenarr1 = i;

while( (c=getc(file2)) != EOF )
{
  if(c == '\n')
    j++;                          //counts nr of rows in file2
}
lenarr2 = j;

arr1 = calloc(lenarr1, 2 * sizeof(double));  //allocates memory for file1
arr2 = calloc(lenarr2, 2 * sizeof(double));  //allocates memory for file2

fseek(file1, 0, 0);                   //sets the file position indicator for
                                      //file1 to the beginning
fseek(file2, 0, 0);                   //sets the file position indicator  for
                                      //file2 to the beginning

for(i=0 ; i < lenarr1 ; i++)
  fscanf(file1, "%lf%lf", &arr1[2*i], &arr1[2*i+1]);  //create arr1



for(i=0 ; i < lenarr2 ; i++)
```

```
  fscanf(file2, "%lf%lf", &arr2[2*i], &arr2[2*i+1]);  //create arr2


 for(i=0, j=0 ; i<lenarr1 ; i++)
 {

  if( (i > 0) && (arr1[2*i-1] != 0) && (arr1[2*i+1] == 0) )    //find start of last zero sequence for file1
    lastzero1 = arr1[2*i];

  while( (arr2[2*j] < arr1[2*i]) && (j < lenarr2) )
  {
    if( (j > 0) && (arr2[2*j-1] != 0) && (arr2[2*j+1] == 0)  ) //find start of last zero sequence for file2
      lastzero2 = arr2[2*j];
    j++;
  }


  if(j == lenarr2)                              //the end of arr2 has been reached
    break;                                      //and no more comparisons should be done


  else if(arr2[2*j] == arr1[2*i])                  //if x in file2 equals x in file1
    y = arr2[2*j+1];                               //difference equals the absolute value of
                                                   //y in file1 - y in file2

  else
  {
    x1 = arr2[2*(j-1)];
    y1 = arr2[2*j-1];
    x2 = arr2[2*j];
    y2 = arr2[2*j+1];
    x = arr1[2*i];
    y = ( (x2 - x)*y1 + (x - x1)*y2 ) / (x2 - x1);       //linear interpolation to get a y-value
                                                         //in file2 that corresponds to the current
                                                         //y-value in file1
  }


  avg_sum1 += arr1[2*i+1];        //sums the y-values in arr1
  avg_sum2 += y;                  //sums the y-values in arr2
  li_diff = fabs(arr1[2*i+1] - y );
  li_diffsum += li_diff;


  if( (arr1[2*i+1] == 0) && (y == 0) )  //Find start and end
  {
    inactive_end++;
  if(start == 0)
    inactive_start++;
  }
  else
  {
    start = 1;
    inactive_end = 0;
  }
}

avg_diffavg = fabs(avg_sum1 - avg_sum2) / (lenarr1 - (inactive_start + inactive_end));
                                            //computes the difference between the average values
avg_sumdiffavg += avg_diffavg;

if( avg_diffavg > avg_maxdiffavg )
{
  avg_maxdiffavg = avg_diffavg;
  avg_maxdiffavgnr = k;
}




li_diffavg = li_diffsum / (lenarr1 - (inactive_start + inactive_end));   //compute the current error
li_sumdiffavg += li_diffavg;

if(li_diffavg > li_maxdiffavg)       //find the largest current error
{
   li_maxdiffavg = li_diffavg;
   li_maxdiffavgnr = k;
}

fprintf(output, "%d:\nCurrent error: %lf\n",
       k, li_diffavg);

if( (arr1[2*lenarr1-1] == 0) && (arr2[2*lenarr2-1] == 0) && (lastzero1 != -1) && (lastzero2 != -1) )
{
  diffzero = lastzero1 - lastzero2;               //compute the time error
  fprintf(output, "Time error: %lf\n", diffzero);
  sumofdiffzero += fabs(diffzero);
  diffzeronr++;
  if((fabs(diffzero)) > maxdiffzero)              //find the largest time error
  {
    maxdiffzero = fabs(diffzero);
    maxdiffzeronr = k;
  }
}
else
```

```
  fprintf(output, "Error computing time error\n");

  fprintf(output, "Difference of the average values: %lf\n\n", avg_diffavg);

  free(arr1);
  free(arr2);
  fclose(file1);
  fclose(file2);
 }

 avg_totdiffavg = avg_sumdiffavg/nr;                    //compute the total average error

 li_totdiffavg = li_sumdiffavg/nr;                      //compute the total current error

 fprintf(output, "\n****************************************************************************************\n");

 fprintf(output, "Largest current error: %lf for comparison nr: %d\nTotal current error: %lf\n\n",
        li_maxdiffavg, li_maxdiffavgnr, li_totdiffavg);
 if(diffzeronr != 0)
 {
  avgdiffzero = sumofdiffzero/diffzeronr;              //compute the total time error
  fprintf(output, "Largest time error: %lf for comparison nr: %d\nTotal time error: %lf\n\n",
         maxdiffzero, maxdiffzeronr, avgdiffzero);
 }
 else
  fprintf(output, "Error computing total time error\n\n");

 fprintf(output, "Largest average difference: %lf for comparison nr: %d\nTotal average difference: %lf\n",
        avg_maxdiffavg, avg_maxdiffavgnr, avg_totdiffavg);

 fprintf(output, "\n****************************************************************************************\n");

 fclose(output);
 system("PAUSE");
 return 0;
}
```

Revisions

Rev	**Date**	**Description of revision**	**Approved**
-	2008-01-16	Issued	Mikael Ström