Aalborg University
Department of Computer Science
Fredrik Bajers Vej 7, building E
DK-9220 Aalborg Ø
http://www.cs.aau.dk/

Title:
Guidelines on Modelling a Train by use of Graph

Theme:
Algorithms and Data Structures

Project Unit:
EVU, autumn 2005

Student:
Niels Klitgaard Lassen

Supervisor:
Linas Bukauskas

Number printed:
3

Pages:
16

Date of Completion:
January 11, 2006

_____
Niels Klitgaard Lassen

Synopsis:

This report describes Guidelines on Modelling a Train by use of Graph. The proposed model considers Services, which are requested by the Customers, as Vertices and concrete solutions, which delivers the requested Services, are represented as Edges. The model is added information concerning Component properties at the Edges and information concerning Conditions and Constraints at the Vertices.
The Model describes the most important connections between Components and Subsystems in a Graph, which can be processed by an Algorithm.

Table of content

1 Problem Definition

This report describes guidelines on modelling a Train, which must be optimized by use of an Algorithm running on the Graph. The Graph is the basis for development of an Algorithm, which aids the Engineers during the Design Process of a Train. The Design Procedure of a Train is complex, since the Train consists of many different Parts, which must fit together. For instance, an IC3 Train – are driven by DSB - consists of around 8000 different Parts.

Beyond the Requirement that the Parts must fit together, everything must be designed in order to meet the maximum load in the working Environment. The maximum Torque Moment in the Gearboxes must meet the maximum Moment delivered by the Engines (This Moment occurs, when braking the Train by use of the Engines) and the Power supply must deliver Energy for the Air-conditioning System, the Interior light and so on …

Some general Requirements from the Customers are:

- X_1 numbers of Train sets of a low price.
- The Train must carry X_2 number of passengers.
- The maximum consumption of Fuel must be X_3 l/km.
- The Speed shall be X_4 km/hour.
- The minimum Acceleration shall be X_5 m/s^2.
- High Reliability (max. X_6 stopping errors per 1 million km).
- The Traction System must be supplied by Diesel – or by Electricity.

- The above Requirements are set up by every Customer.

Some Customers have further Requirements, which could be:

- There must be light at the passenger Seats.
- The Chairs shall be placed in Groups of four around a Table…
- The Train must be equipped with a Toilet.

Some of the requirements are counter clockwise, in the sense that they pull design decisions i different directions. For instance, reducing the Price can result in a heavy Train, which has poor effect on fuel consumption and reliability and vice versa. The big Puzzle to the design Engineers is to select the right Subsuppliers for every Part of the Train and to ensure that everything fits together. This is done by requesting offers from Subsuppliers on the different Parts and Components. There are two different types of Subsuppliers; Those who produces the Components/Parts for the Train as a secondary product of their Business, and those who produces Parts for Trains as their primary Business. Suppliers, who only have the delivery of components to Trains as a secondary production, often only are willing to make a few customizations of a standard Product. This group of suppliers is typically those, who deliver engines, Brakes, Air Conditioning and so on, whereas the group, who produces components for Trains as a part of their primary business are willing to deliver expensive but fully customized Components. No matter which Subsupplier you are dealing with, it is necessary to specify the working conditions for their Components, and since not all Interfaces can be specified from the very beginning, the process of designing the Train will run in more iterations. It will be attempted to lock the Design around the important and critical Parts at an early stage of the iterations, while less important and fully customized Components can be specified in a more precise way in later Iterations. However, the request for offer must always specify to the Subsuppliers, how their Subparts are loaded, when the Train is in operation, and specify some further requirements on how the Subpart must be constructed in order to fit with the rest of the Train (This means a specification of the Interfaces). When the Subsuppliers return with their Offers, the Group of Designers selects the best Offer, and the design of the Train is locked on this Part.

This report presents a model of the Train, which - given a set of Components or Subsystems - can

be used in an Algorithm designed with the purpose to help the designers to choose among many Components, which Components to use in the Design of a Train. The Design of the Graph Model proposed in this report, supports development of an optimization Algorithm, which is able to select between different Subsuppliers and to return a set of Components that can be joined into the globally optimized Design of the Train.

Some further and nice properties of the Graph and Algorithm could be that it would be easy to make "What if ...?" analysis on Constraints and Subsystems and an Analysis on, which Constraint or Resource is the critical Bottleneck on locally Subsystems. The “What if ...” analysis could, for instance, be used to set up some indication on how important, it is to reduce the Weight; Meaning “If the weight is reduced by 1000kg in the Interior-equipment, the price of the Equipment can raise 10000$ without influencing the total cost Price negatively, since the cost Price of Boogie and Power Supply is reduced more due to reduced requirements on these parts”. Information about the Bottleneck Constraint in a Subsystem can explain which Requirement is the most critical in a Subsystem. This makes it easier to the Design Engineers to focus on the most important Constraint during the Design and Production Process. The considerations on “What if...”-analysis and Bottleneck Constraints are not discussed further in this Report.

2 Design Issues

This chapter describes a Graph model, which is used in an Algorithm to find an optimum solution, for the Design Problem. The Graph is denoted, G, and consists of Edges, E, and Vertices, V, and these notions will be used in the following to relate the design of a Train consisting of many different parts and components into a Graph – see Figure 2.1.

The chapter takes its basis in a model of the boogie subsystem, since the sub graph at this level have all the properties of the global graph but is easier to survey, than the situation where the entire train i considered.

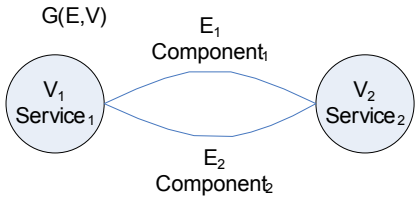

Figure 2.1: The Graph consist of vertices V1 and V2 connected by Edge E1 and Edge E2

2.1 Definition of Vertices

There can be set up more different approaches to model the Graph of a Train, but considering Vertices as services or functions and considering connections between these Vertices as Edges leads to a Graph, that represents the Train as a nearly complete Tree Structure. The Tree Structure is a logical way to consider the design of a Train, meant in the way, that a Train is equipped with the services: Interior, Front, Traction, Boogie, Airconditioning, Controlling System and Carbody. The Subsystem or Branch of the Boogie also have Branches, which is a branch for Wheel set and a Branch for Frame, where a Wheel Set is considered as a Subbranch with the Subbranches Wheels, Shaft and Brakes. The other main Branches in the Train can also be divided into Subbranches and Subbranches to Subbranches resulting in a Tree Structure – see Figure 2.2. In later sections it is discussed, that the Graph not always will be a complete Tree Structure.

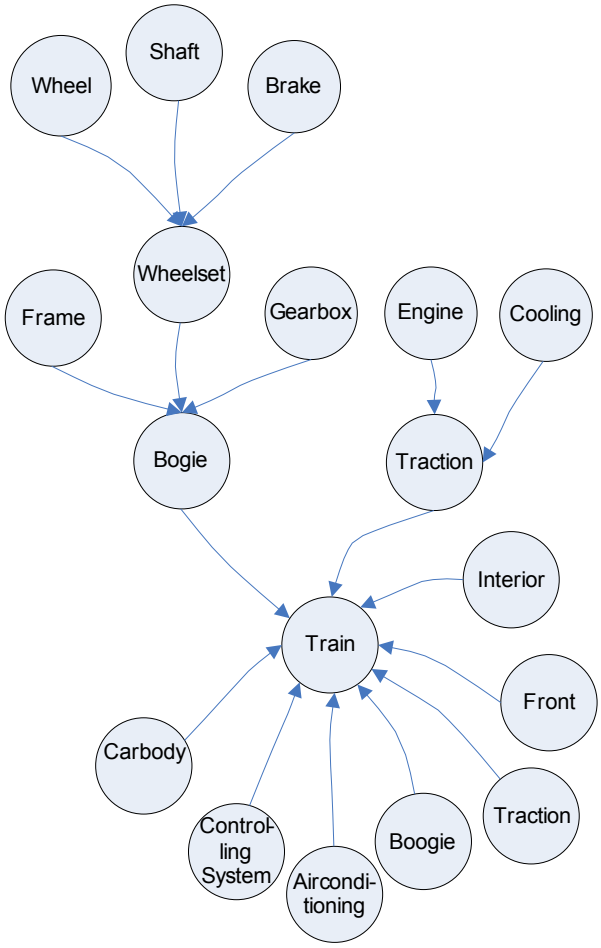

Figure 2.2: An incomplete Graph of a Train. The main Branch Boogie has been detailed, whereas the other main Branches are left back in a cause detailing Level.

The functionalities or services represented by each Vertex are requested by the Customers or Subsystems and Components. Customers can have Requirements leading directly to a Vertex for a particular Service. This could for instance be the situation, when the Customer request the Boogies to be equipped with pneumatic Brakes. This request results in a Subbranch to the Boogie Branch, and this Subbranch has at highest Level a Vertex specifying the request of pneumatic Brakes on the Boogie. The request on a Service can also be set up due to dependencies between Subsystems. This could be the requirement that the Boogie at least consists of a Frame and two Wheel sets - otherwise it is not denoted a Boogie and can not deliver the functionality, which it is expected to deliver, when it is denoted a Boogie. Both the Frame and the Wheel set are therefore modelled as Vertices in Subbranches to the Boogie-branch.

2.2 Definition of Edges

Edges are the concrete solutions to reach the requested requirements set up by defining the Vertices – see Figure 2.1. This means concrete Components and Subsystems assembled by concrete components are considered as Edges, which delivers a specific Functionality or Service. There can be some overlap in the conception of Vertices and Edges, because a Component often is denoted by the name of the service, it delivers. An example is the Brake-component, which actually has the property, that it can deliver the Service to brake the Train. However, in the description of Edges it is the specific Component or assembly of Components (sub System), which is denoted an Edge. This means, the Brakes from different Suppliers are presented in the Graph as different Edges with different Parameters, but the Subgraph for the Boogie only has one Vertex representing the Service delivered by the Brakes. Different Edges connecting Vertex Brake to the Vertex Boogie, therefore represents different possible choices to get the Service Brake - for instance respectively a German supplier and a Chinese supplier.

It has also been considered to model specific Components as Vertices, and then the connections between the vertices should carry information of how many vertices to be used for a feasible design of a Train. A Component could end up with no representation in the final Design of the Train, and the Edge to that Component therefore should act, as if the Component/Vertex not was represented in the Graph. A model of a Wheel set would for instance consist of the Vertices Wheels, with Part Number Pno_1, Brakes, with Part Number Pno_2 and Shaft, with Part Number Pno_3 and these Vertices should be connected by Edges, if the Components could be assembled into a feasible solution. The Edges should carry a parameter on the quantity of Components needed to design the Wheel set. If it was possible select between two different Brake Components, the Algorithm should handle to set the Quantity of the one Brake Component to zero and the Quantity of the other Brake Component to two. Dependencies between Brakes and Shaft could be expressed as directed Edges. The Train perhaps could be modelled in this way, but the Graph model would end up being very complex. Therefore this solution is not further discussed.

Directed or Undirected Edges

Edges in Graphs can be described as Directed or Undirected. The Edge from Wheelset to Boogie is directed, which means a Boogie has Wheelsets, and it make no sense to have a Wheelset without having a Boogie – see Figure 2.3. The Wheelset is incident to the Boogie. Since the Graph only consists of Vertices, which must be included in the final Design of the Train, all Vertices must be "visited" during the call of the optimization Procedure. In order to limit the size of the Problem to be handled in the Algorithm, all Edges are specified as directed and the Graph furthermore must be acyclic. This means that the Algorithm on the Boogie Subbranch performs a bottom up Analysis starting from Wheel, Brake and Shaft and towards the adjacent Boogie Vertex. Since the Wheel, Brake and Shaft vertices not are adjacent to the Boogie - due to the direction of the Edges - the directed Edges ensures a subbranch will not be considered over and over again.

Normally the in-degree (number of Edges entering the Vertex) is higher or equal than the out-degree, but there can be situations, where a Vertex has higher out-degree than in-degree, and this can cause problems to the Algorithm. This issue is discussed in the section” When Subbranches not are Independent“.

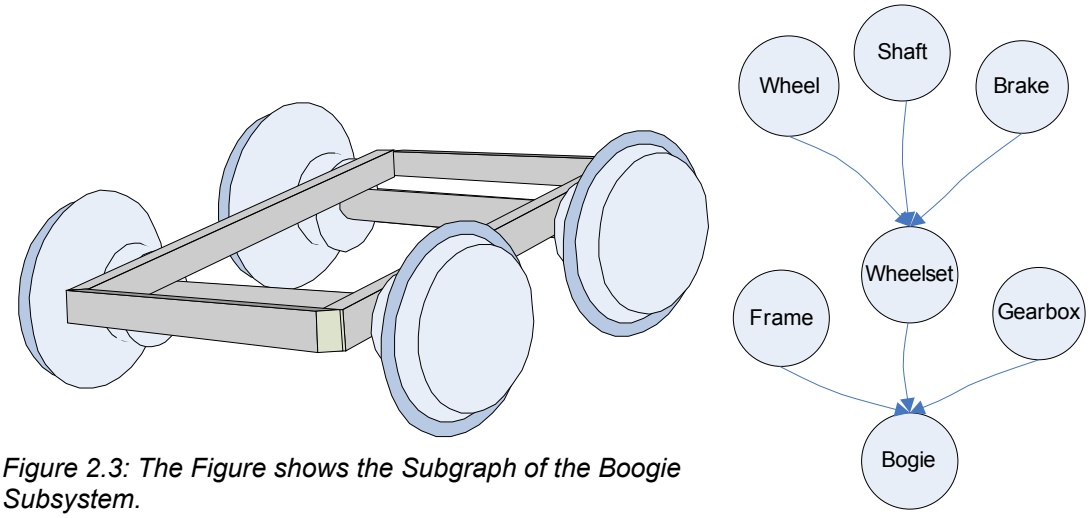

Figure 2.3: The Figure shows the Subgraph of the Boogie Subsystem.

2.3 Summary of the Design

The model of the Graph G has been described as consisting of Vertices, V, and Edges, E. Vertices represents the requested Functions or Services and the Edges represents concrete Components and Subsystems, which delivers the Service requested by the Vertices. Vertices can be set up by answering the question: "What functionality and properties must the solution have?", whereas Edges can be set up by answering "How can these functionalities and properties be reached?". Some typical requirements on the Services are, for instance, minimum Engine Power, maximum Torque moment on axles, Colours in the Interior or of electrical Parts and so on, and these requirements are met by the properties of the Components representing the Edges.

Modelling the graph, by specifying requirements to the solution though a set up of all requested services in Vertices and possible choices of Components in Edges, results in a Graph, where the optimum Solution, which could be minimum Price, can be found by an Algorithm minimizing the Price of visiting all Vertices considering no Constraints are violated.

3 Approach to Problem

This chapter describes things to be aware of when choosing an Algorithm - or Strategy - for the solution of the optimization Problem. Subsystems in the Graph of the train/boogie are not independent of each other. This is due to the fact, that for instance the weight of the interior effects the needed engine power. If the Price is considered as the cost function, it's therefore not for sure the cheapest (and heaviest) Interior, which results in the cheapest train, because there perhaps is needed a more powerfully and expensive Engine to carry a cheap and perhaps heavy interior. Because of this, shortest path Algorithms can not be applied to the overall solution of the system, since the shortest path algorithms are greedy, which means they presume, that no subproblems can be optimized. Therefore dynamic programming is considered as the alternative.

There are here described some techniques to contribute for a solution of the optimization problem. The technique described are constraint functions, setup of quantity of components and a description on how to model a Problem, where Subbranches not are independent of each other.

3.1 Conditions and Properties on Edges and Vertices

When the algorithm runs, it checks the different solutions for feasibility, ensuring that no component is overloaded or coupled to components, which it can not be joined together with in the real life solution. In order to handle these kinds of problems, it is necessary to place a Property Table on each Edge, carrying the main properties of the particular Component considered. The Property Table is connected to the Edge as Satellite data, and the Information inside the Table concerns the properties of the Component and the Quantity of the particular Component. The Constraint functions and Conditions are placed on the Vertices. These validates the properties of incoming Edges and propagate properties from incoming Edges to outgoing Edges, and the conditions are also represented as Satellite data placed at the Vertices corresponding to the data on the Edges – see Figure 3.1.

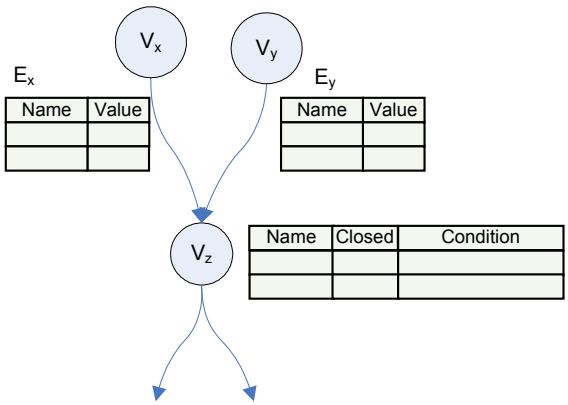

Figure 3.1:The Figure shows how Edges and Vertices are added respectively Property Tables and Constraint Tables.

The Conditions at the Vertices concerns:

- resource constraints expressed as combination of Algebraic functions and Logical expressions. These constraints concern some kind of resources, which must not be used up, when the system is designed. An example could be maximum weight shall not be more than 1000kg.
- violation constraints expressed as Logical expressions. These constraints expresses incompatibility between Components. An example could be Electrical equipment, where low Voltage Components not can be used in the same circuit as high Voltage Components.
- aggregations of the properties of incoming Edges. An example is accumulation of weight.
- regular expressions. Notation sum(E*.weight) implies the sum of the weight of all incoming Edges. An example could be that all Electrical Equipment must have the same operating voltage (220V).

The outgoing Edges are all considered to be generated runtime by the Algorithm performing the analysis, and only feasible solutions are generated. Properties on the outgoing Edges are all inherited or propagated from properties of incoming Edges. An example is later given in Figure 3.5.

3.2 Quantity of Components

Some components in the graph must be represented in a given proportion to each other. A

Wheelset with Brakes - consists of the components Wheels, Shaft and Brakes – and these Components are always represented with the following proportion to each other 2-1-2 – see Figure 3.2. A Wheelset consists of 2 Wheels 1 Shaft and 2 Brakes, and this is modelled in the Graph by use of the Property Table, which is added an attribute called Quantity.

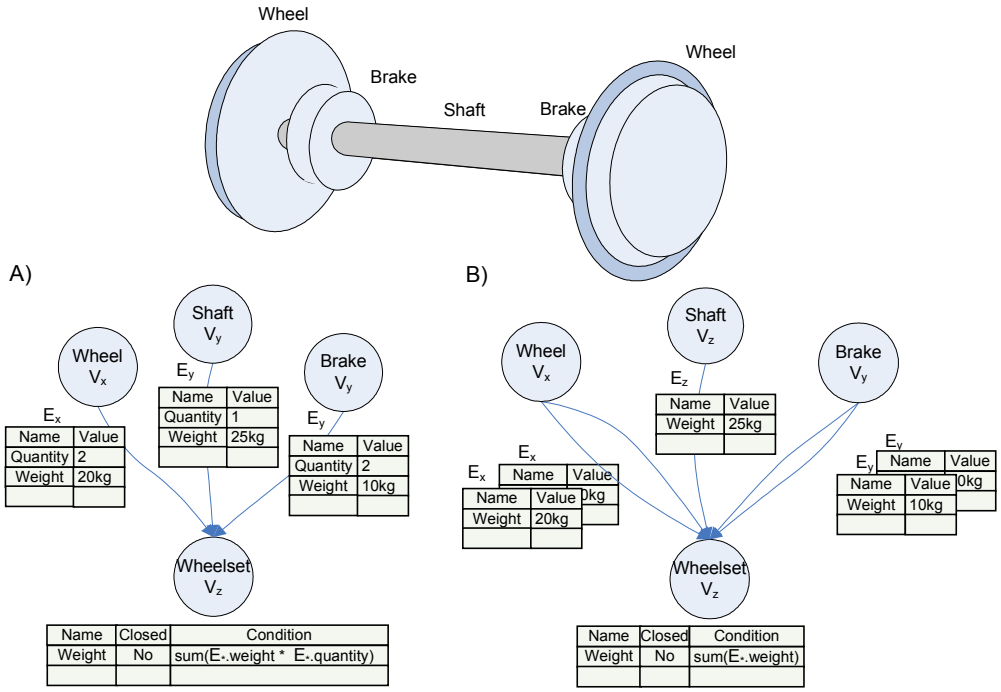

Figure 3.2: The Figure shows to approaches to represent Components in a given proportion relative to each other. Figure A shows how the number of Components represented by an Edge is expressed in an Attribute in the property Table, whereas there are added more Edges to the Model in the design showed in figure B.

An other solution could be to allow two Edges representing the same Component between Vertex Wheel and Vertex Wheelset and two Edges between Vertex Brakes and Vertex Wheelset the Brakes as depicted in the Figure 3.2. The chosen approach effects the Algorithm in a Complexity reducing direction, since it ensures, that the algorithm does not select different Components of the same Component Type, to set up the optimum solution.

Calculated Quantities

In some Situation, the algorithm must calculate how many components that are needed to fulfil a certain constraint. This can done by iterating on the Quantity Attribute places on the Property Table, and this Quantity can be increased until the Constraint is fulfilled or a given limit has been reached. It has been considered to allow cycles in the Graph, where Backward directed Edges without properties could be used by the Algorithm to move backwards in the Graph structure, applying yet another of the constraining Edge, and re-estimate the limiting constraint. This proposal has the poor characteristic that it is difficult to control that the same component is used on both Edges. For this reason, it has been decided to consider the quantity as an attribute in the property table. An example of how the calculated Quantities is presented in the Property Table respectively the Condition Table is given in Figure 3.5 in the Tables connected to E_8 and V_4.

Limitation of optimization problem

In order to limit the size of the Problem during the optimization process, only Property Parameters of locally interest are considered, when selecting the composition of Components and Systems. The Design parameters are encapsulated at the local level by use of the Condition Table on the Vertices, which is supplied with a Column called “closed”. The Column is set by the designers and tells, if the Design Parameter is encapsulated to the Subsystem. If the Attribute is set as “No”, the Parameter propagates to a Design consideration at higher level, whereas the Attribute Value “Yes” indicates that the Parameter is encapsulated at this Level. The encapsulation of parameters and

check for feasibility done by the conditions described in the Condition Tables are important tools to reduce the size of the problem. The difference between not to reduce the problem – shown in Figure 3.3 - and the Situation, where Condition Table has effected the amount of feasible solutions – see - is expected to be high.

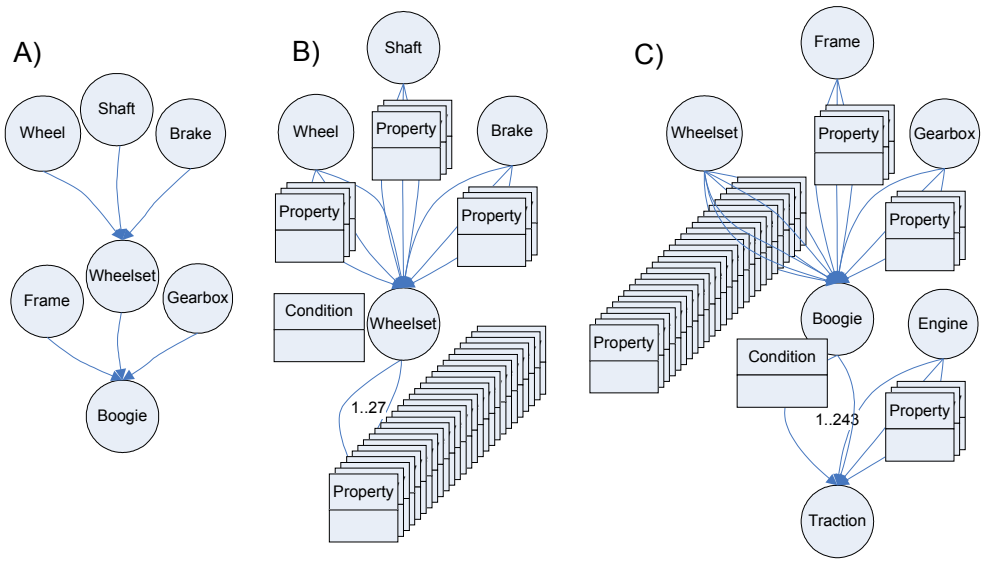

Figure 3.3:The Figure shows the expansion of Edges from Components being part of Wheelset and to the Boogie. Part A shows Wheelset can be configured in 27 different ways which propagates to 243 different configurations at Boogie Level, if the Boogie is assembled of 5 different service types each supported by 3 different choices of Components. Conditions are here expected not to effect the amount of possible solutions.

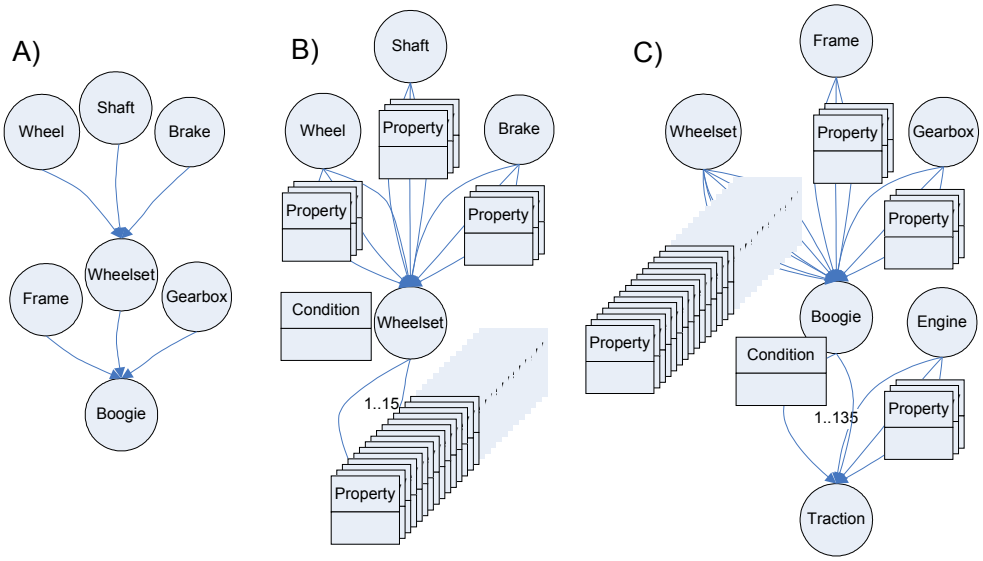

Figure 3.4:The Figure shows that the conditions at Wheelset Level reduces the amount of feasible Solutions from 27 to 15, which as shown in part B effects the amount of possible solutions for the Boogie very much.

3.3 An Example of Modelling

The Model in Figure 3.5 shows the Boogie Subsystem and in this, the Wheelset Subsystem consisting of Wheels, Brakes and Shaft. The Model includes Property Tables on the Edges and Condition Tables on the Vertices. The different characteristics of the Components are described in the Property Tables connected to the Edges, and Components delivered by different Suppliers has

different properties. Property Tables describes the important Design Parameters on the Subsystem or Component, and the Parameter price could be used by the Algorithm to find the overall optimum Solution. Other parameters (diameter, weight, length, and colour) are important in the sense to make the Parts fit together, since they describe Interface properties or other characteristics of the Components. The Interface description is handled by use of the Condition Tables describing the Constrains in the Graph.

Some comments to Notions used in Figure 3.5:

- A concrete Edge between two Vertices is denoted by name E_i and is expected to carry a Property Table. An Example could be the Edge E_1 between Vertex Wheel V_1 and Vertex Wheelset V_4. The Edge E_1 is related to a Property Table.
- An arbitrary Edge between Wheel V_x and Vertex Wheelset V_y is denoted (V_x,V_y). (V_1,V_4) Є $\{E_1,E_2\}$.
- There is used a regular expression for expressing the condition on price in Vertex V_5. E_* means all incoming Edges.
- A concrete parametre in the Property Table on an Edge is referenced by its Edge and Parametre name. (V_1,V_4).do references the value of Parametre “Diametre outer” on Edge E_1 or E_2 depending on the Edge chosen between the considered Vertices.
- Edges E_7, E_8 and E_9 are expected to be generated automatically by the Algorithm, and their Property Table are here set up by Propagating the Properties of incoming Edges to Vertex V_4 by use of Condition Table related to Vertex V_4.
- The Condition Table related to Vertex V_4 contains a Logic expression E_1 NOT E_3 meaning these two Edges excludes each other.
- The Condition Table related to Vertex V_4 propagates properties from incoming Edges to outgoing Edges. The propagated properties has attribute value “No” in column “Closed”. The Condition Table related to Vertex, V_4, ensures the property “weight”, is accumulated from the incoming Edges and set into the Property Table of generated outgoing Edges, E_7, E_8 and E_9.
- A Logical Constraint on inner Diameter for driving shaft in Wheels and Brakes respectively Outer Diameter of Shaft ensures that the parts can be assembled. The Diameter inner has the “closed” Attribute Value “Yes” at the Wheelset Vertex V_4, which means these properties are encapsulated and must not be considered, when setting up a composition on Boogie consisting of Frame and Wheelset.
- The property Quantity is locked for Wheels, Shaft and Brakes in the sence that the attribute values for “Quantity” on Edges E_1,... E_6 is set to “1” or “2”. On the Edge (V_4,V_5) the attribute is set to “1+” meaning the Algorithm can increase the quantity in order to reach a feasible Solution.
- E_{10}, which is not expected to be the only Edge leaving Vertex V_5 references the subpath $\{E_7, E_x, E_y, E_z\}$, which indicates all subgraphs to V_4 is out of interest at this Level.

Some comments to Figure 3.5, which makes it easier to read:

- E_6 is excluded from all feasible Solutions due to constraint on di.
- E_x, E_y and E_z are expected to enter V_5 and each Edge is expected to carry a property Table.

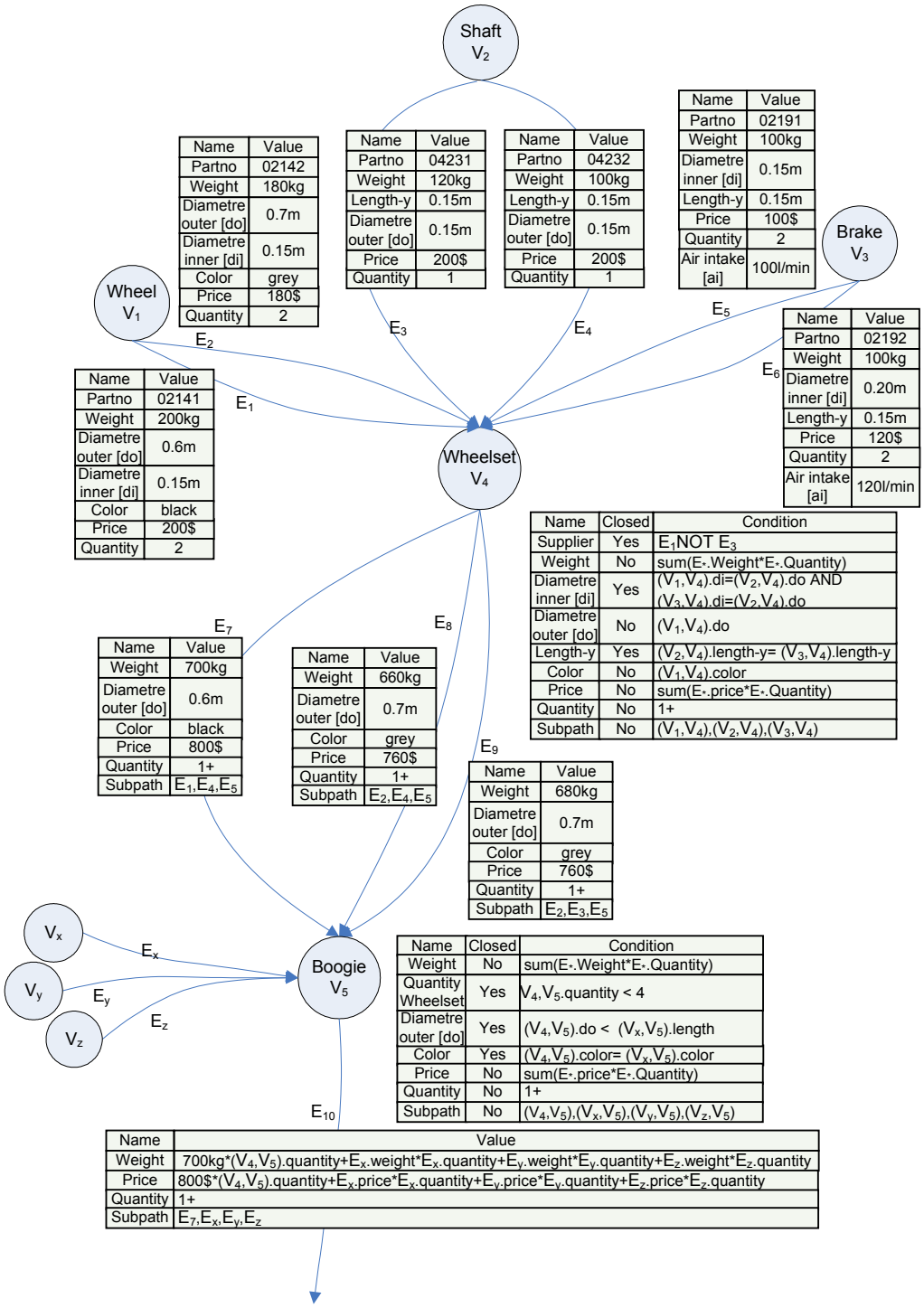

Figure 3.5:The Figure shows how the Condition Tables at the Vertices can be used to reduce the number of possible Solutions, as well as how the Subsystem properties are encapsulated.

3.4 When Subbranches not are Independent

There are basically two ways to model the issue, when more Subgraphs requests services from the same Service. In the one Model, the System, which deliver the requested resource, is connected directly to the root element in the Tree Structure – see Figure 3.6. By use of the Condition Table at Root Level it is possible to express which configurations of the Subsystems are feasible and which is not.

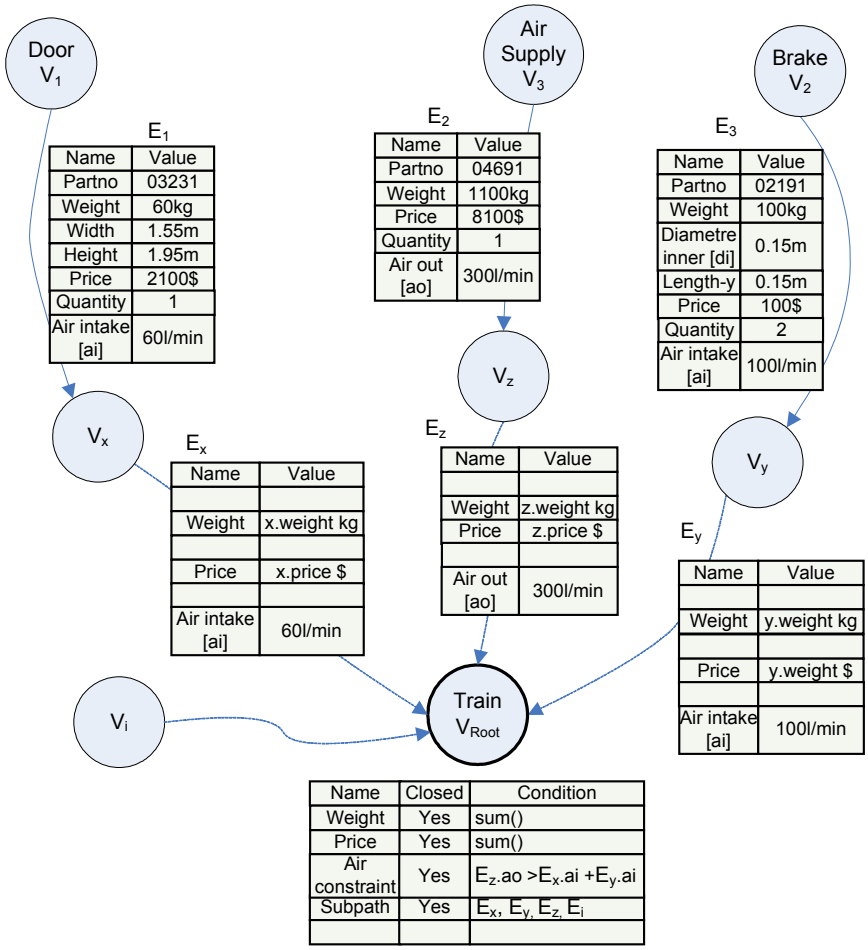

Figure 3.6:The Air Supply delivers compressed Air for more Systems - Doors and Brakes - and the Constraint on Air Supply is in the Model placed in the Constraint Table at the Root (Train).

The other Model also makes use of the Condition Tables at the Vertices for handling the Problem. In this Model the Constraint on Air Supply on delivery of compressed air to all depending Systems is placed higher in the Structure in order to reduce the amount of possible combinations to be violated on this constraint. In order to keep the graph directed an acyclic there has been introduced some dummy Vertices, and the out-degree of the Vertices are allowed to be higher than 1 – see Figure 3.7. Note in Figure 3.7 that the Edges E_7 and E_8 are not carrying any properties, but they have to exist in order to V_7 can produce E_{10} and V_9 can produce E_{11}. V_7 and V_9 are both denoted “dummy” because, they are introduced in order to avoid cycles. Edge E_{12} is related to an property Table containing the properties related to Air Supply, and these properties can be propagated to the Train Level.

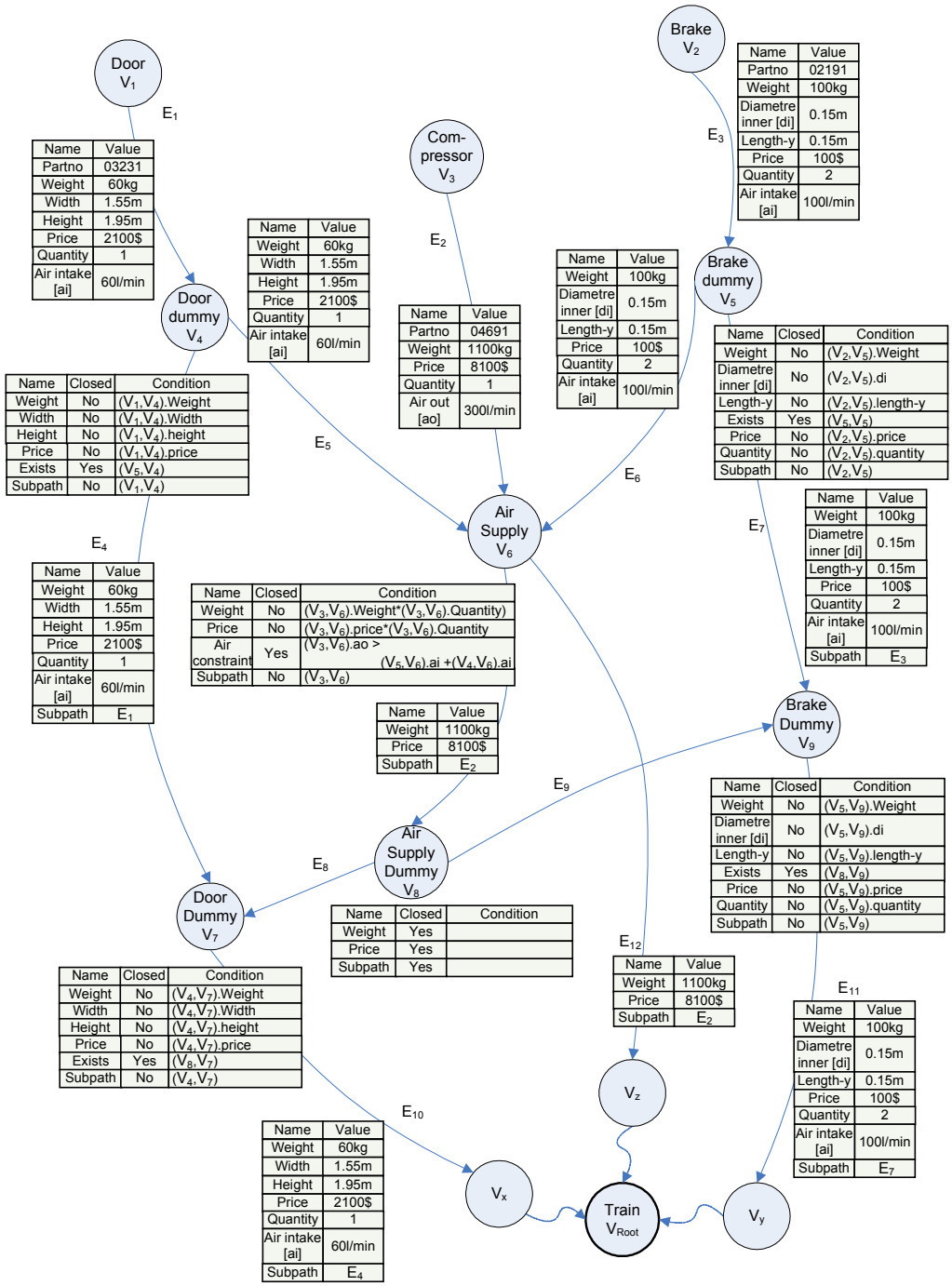

Figure 3.7: The situation, where subpaths in the Graph are not independent, is here modelled by use of dummy-vertices.

4 Future Work

The future work will be to set up a Cost function to be optimized, to develop an Algorithm, an Application and a Database Design for the technical part of the optimization, and to find a way to visualize the optimized Configuration.

Cost Function - or Object Function

The Cost function describes what to be optimized. When considering the Optimization of a Train or Boogie, there are at least two different concerns to be optimized. The price is obviously an Object for optimization, but the total Weight of the Train is an object as well, because the heavier the Train is, the bigger is the loss of energy during operation, when breaking and accelerating the Train. This means that the optimization problem is multiobjective [1] and this kind of problems can be handled in two different ways.

- One Method is to express a single Cost function as a combined sum of weighted factors to the different Cost functions.
- The other Method is to consider the most important factor of the Cost function as the overall Cost function in problem, and redefine other Cost functions as Constraints to the Optimization. The Cost functions defined as Constraints in the Optimization Process can be varied in order to find the overall optimal Solution.

Considering the optimization of a Boogie, the optimization could be done by expressing the Problem as: Minimize the total Price of the Boogie considering the total Weight of the Boogie must not exceed 4.5ton or 5 ton or 5.5ton.

Developing an Algorithm for the Optimization

The overall consideration when designing the Algorithm is, that the Algorithm performs a bottom-up analysis, and dynamic programming (fastest way through an Assembly Line [2 pg 324]) inspires the Design. The approach of dynamic programming is that results to Subproblems are stored and reused in every step of the Algorithm. It is also a Requirement, that all Vertices (in this report considered as Functions or Services) are visited in order to get at valid Solution of a Train. The Graph do not look like a well known Graph because, there are more starting points in the directed Graph, every subproblem must be checked for it´s feasibility and there are more Edges between the the same pair of Vertices.

Visualizing Results of the Optimization

Finally there should be done a lot of work in visualizing the results of the optimization, and it will in this context also be reasonable to consider, if it is possible to make “What if...”-analysis and to detect Bottleneck Constraints to optimal configurations.

5 Conclusions

This report presents a Graph Model, which can be used for modelling a Train. Vertices are considered to be Services og Functions, which the Customers or Subsystems request on the Train, and Edges are considered as concrete Solutions on how the requested service can be delivered. Edges are either considered as components delivered by Subsuppliers or as Subsystems set up by the algorithm during execution. Vertices carries Satelite data, here named Condition Tables, and these Tables are used for checking, if a given/proposed solution for a Subsystem is feasible or not. Edges also carries Satelite data named Property Tables, and these Tables specifies some important features on the Edges. The Condition Tables on the Vertices also helps the Algorithm to propagate properties from Property Tables placed on the incoming Edges to the Subsystem property Tables placed on outgoing Edges. This means the Subsystems inherits and propagates some of the properties of the Components, of which they consist.
The Graph is directed and acyclic, and there has been given examples on how these properties can be maintained, even when considering a complicated design issue, where more sub-paths in the Graph requests a service delivered by one component.

In the last chapter there is given an Example on how the directed and acyclic behavior of the Graph in conjunction with the use of Condition Tables, can be used to express how Component properties are inherited and propagated into Subsystem Properties. A Subgraph, representing a Subsystem in the Graph, only has to be revised once, and storing the Subsystem properties, leads to a Model of the Graph, which can be used in a more effectively Algorithm than the case where Sub-solution was not stored and infeasible solutions not excluded at an early stage.

Bibliography

1: Arora, Introduction to optimum design pg.466, McGraw Hill, 1989
2: Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein, Introduction to Algorithms, Second edition, MIT Press, 2003, ISBN 0-262-53196-8