Improving the Maintenance Planning of Heavy Trucks using Constraint Programming

Tony Lindgren[1,2], Håkan Warnquist[2], and Martin Eineborg[2]

[1] Department of Computer and System Sciences
Stockholm University, Stockholm, Sweden
[2] Scania CV, Service Support Solutions
Södertälje, Sweden

Abstract. Maintenance planning of heavy trucks at Scania is presently done using static cyclic plans where each maintenance occasion contains a fixed set of components. Using vehicle operational data gained from on-board sensors we will be able to predict at which intervals each component needs to be maintained. However, dynamic planning is needed to take this new knowledge into account. Another benefit using dynamic planning is that vehicle owners can influence maintenance plans with regard to their business. For this reason we have implemented a prototype of an automated maintenance planner based on constraint programming techniques. The planner has successfully been tested on vehicles belonging to Scania's internal haulage contractor. In this paper we will describe the planner and what we have learned using and developing it as well as ongoing work on how the planner will be developed further.

1 Introduction

Scania Commercial Vehicles (Scania) is a manufacturer of heavy trucks, coaches and engines for industrial and marine usage. This paper is concerned with Scania's ongoing effort of improving its maintenance service offer. Better maintenance planning is beneficial for the customers because they can utilize their Scania products in a more efficient manner and it makes Scania more competitive. To achieve better maintenance planning, we have developed a new maintenance planner that uses constraint programming techniques.

Customers are currently offered services such as Repair and Maintenance Contracts enabling the customer to fixate the operating costs of the vehicle [12]. When the vehicle manufacturer has full responsibility for vehicle repair and maintenance, the cost of the repair and maintenance contract can be reduced by customizing the maintenance planning for each individual vehicle. To achieve this customization, more information regarding the current and predicted status of the vehicle is needed. This information can be obtained by employing techniques for Integrated Vehicle Health Management (IVHM) which is an area interested in improving the safety, availability and reliability of vehicles [1, 7].

Using vehicle operational data gained from on-board sensors we can predict when and how often a component needs to be maintained. The components'

individual maintenance requirements make it possible to create more efficient maintenance schedules than using the present scheduling method where, three modules consisting of fixed sets of components are scheduled for maintenance with a preset periodicity [11]. The maintenance of an individual component is referred to as a maintenance point. When the maintenance points can be scheduled freely with irregular periodicity, the task of creating an efficient maintenance plan becomes too difficult for a human planner and there is need for an automated planner.

As a Proof of Concept (PoC), we have implemented a maintenance planner prototype using finite domain constraint programming techniques and evaluated it on the haulage contractor responsible for driving goods to Scania's factories with promising results. We will also report ongoing work with the next generation of the maintenance planner based on the lessons learned from the PoC with the haulage contractor.

2 The Problem

A customer that utilizes a Repair and Maintenance contract wants to maximize the availability of the vehicle and Scania as an issuer of the contract wants to minimize the maintenance costs. Downtime is when the vehicle is intended for use but not available. This time is costly for the customer because of loss of profit. We want a maintenance plan where each component is maintained sufficiently often to prevent components from breaking down and that has minimal interference with the vehicle's intended use. The customer cost of a maintenance plan is dependent on the downtime, the number of maintenance occasions, the part costs, and the time spent at the workshop.

The maintenance need of a component depends on how the vehicle is operated by the owner. A vehicle that is operating with heavy loads may need oil changes more frequently than one that is operating with lighter loads. The better we can predict wear, the more correct maintenance intervals we can use for each component. The vehicle has an internal network of connected computers for controlling functions in the vehicle. The computers collect data about the vehicle which can remotely be sent to a central server for further processing with the purpose of computing the required maintenance intervals.

The inputs to the maintenance planner are the maintenance point intervals of each component and optional user preferences in form of when maintenance can be done and when it cannot be done. The output is a maintenance schedule listing the dates for each maintenance occasion and, for each occasion, a list of components that should be maintained, see Figure 1.

3 Current Solution

Today the maintenance plan for a vehicle is set when the vehicle is sold. This is typically done by the seller together with the buyer by selecting one of a set of

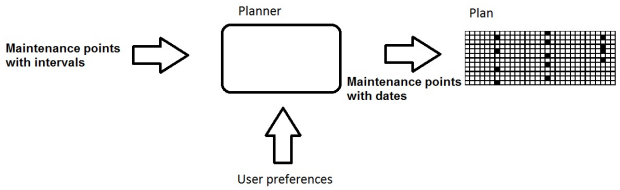


Fig. 1. Inputs and outputs of the maintenance planner.

predefined maintenance plans that best matches the vehicle specifications and the buyers intended usage.

The predefined maintenance plans are developed and maintained by skilled personnel having knowledge about both the products and the customer's usage. Vehicle usage is divided into six typical applications types. For each application type and vehicle specification, a cyclic maintenance plan is given as the number of kilometers between maintenance occasions with fixed maintenance protocols. Maintenance is always done in a cycle of S-M-S-L occasions, where S = Small, M = Medium, and L = Large are different maintenance modules for maintaining different sets of components.

There are a number of problems with the way maintenance plans are created today:

- Much responsibility is put on the salesperson to know the product as well as the customer's usage of the product.
- Once created the plans are seldom updated even if the application of the vehicle changes. Thus, it is possible that the maintenance a vehicle receives does not correspond to its needs.
- Although the fixed S, M, and L modules make it convenient to plan, they contain maintenance points that do not need to be grouped together with the effect that components are maintained more than necessary.
- The current maintenance plans are coarse in the sense that the precision in the type of application must be fitted into one of the six types of application. Therefore the experts dictating when maintenance ought to be done, use a safety margin given the uncertainty of the actual usage of a particular vehicle. This has the consequence that plans are not individualized to the degree that they could be.

4 An Automated Maintenance Planner

We have used Constraint Programming to create an automated maintenance planner which has been installed and used by two workshops servicing 20 ve-

hicles belonging to the Scania Transport Laboratory which is a Scania-owned transport company responsible for transporting goods between Scania's factories in Europe.

4.1 Motivation

The work load of the trucks is high and usage of around 16 hours a day is not unusual. To avoid interference with the daily operation of the trucks, a requirement from the fleet planner was that the trucks could only be maintained every forth week for a maximum of four hours. Such requirements together with previously mentioned goals of minimizing maintenance costs and offering better services to customers was the main driving force for developing this maintenance planner. The prototype was created to gain knowledge of how a solution could be implemented and what aspects are critical for Scanias customers.

The planning problem is too complex to be solved manually. We therefore chose to formulate the problem as a constraint satisfaction problem because many of the requirements on the plan are naturally translated into constraints and also because Constraint Programming techniques has historically been successful for applications similar to this. For example see [10, 8, 5, 2, 6].

4.2 Formulation of the Constraint Satisfaction Problem

The maintenance planning of a single vehicle is formulated as an independent Constraint Satisfaction Problem (CSP). A solution is a plan for all maintenance points with a resolution of one week and a limited horizon. In the Scania Transport Laboratory PoC each vehicle had around 80 maintenance points that needed to be scheduled 52 weeks ahead.

Each variable in the CSP corresponds to a maintenance point that needs to be scheduled in time. Where the Dm_i refers to the domain of the i:th maintenance point. The latest completion time (lct) of a maintenance point refers to its calculated maintenance interval. To reduce the solution space, a constraint is added that dictates the minimum maintenance interval of each maintenance point, i.e. earliest start time, (est). The est is user defined and typically between one third to half of the calculated maintenance interval as shown in equation 1. This also ensures an offset between two maintenance occasions are no closer than the est. In all equations, I refer to the set of maintenance point variable indexes.

$$EarliestStartTime : \forall_i \in I : est_i = lct_i - \frac{lct_i}{offsetParam \in [2,3]} \tag{1}$$

$$Domain : \forall_i \in I : Dm_i = [est_i, lct_i] \tag{2}$$

The usage of est_i affects the i:th maintenance points domains as in equation 2. Maintenance point dependency chains are defined by assigning a *starting variable* which has a domain value between est_i and lct_i. Each variable in these dependency chains corresponds to an occasion of a maintenance point and the

value corresponds to the time when the maintenance should occur. Successive maintenance points are then created until the planning horizon is reached. In the dependency chains each new variable gets a domain with a earliest plan starting time mpv_j that is equal to or greater than the preceding maintenance point variable $mpv_i + est_j$ and a latest plan completion time for mpv_j that is less than or equal to $mpv_i + lct_j$, shown in equation 3 and equation 4.

$$EarliestPlanStartTime : \exists_{i,j} \in I : mpv_j \geq mpv_i + est_j \tag{3}$$

$$LatestPlanCompletionTime : \exists_{i,j} \in I : mpv_j \leq mpv_i + lct_j \tag{4}$$

After the maintenance point dependency chains have been created, then, if the user has defined certain periods when maintenance can be done (*cbd*), or when it cannot be done (*cnbd*), these periods are handled as show below.

$$CanPeriod : \forall_i \in I : Dm_i = Dm_i \cap cbd \tag{5}$$

$$CannotPeriod : \forall_i \in I : Dm_i = Dm_i \setminus (Dm_i \cap cnbd) \tag{6}$$

In practice this means that *cnbd* periods are excluded from the variable domains. If *cbd* is defined, then these periods constitute the variables domain. Each dependency chain is related to one maintenance point which means that for our PoC there are around 80 dependency chains.

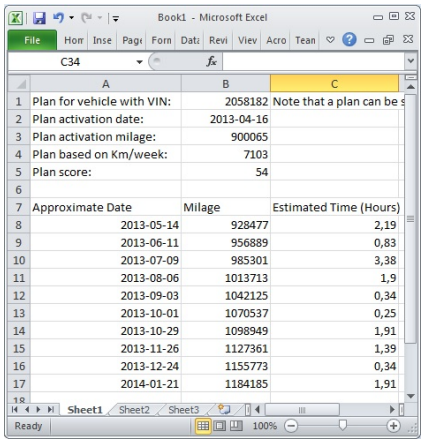
Input and Output. The input consists of the periodicity of each maintenance point expressed in kilometers and the times when each maintenance point was last maintained. The periodicity is then converted into weeks based on the expected number of kilometers the vehicle will be used per week which can either be set by the user or be learned from previous vehicle behavior.

If it exists, the solver finds a maintenance plan that satisfies all the constraints. This plan is then presented to the user as an Excel worksheet listing the dates, expected mileages, and durations of all the scheduled maintenance occasions, see Figure 2. The dates are approximate because the planner only plans with a resolution of one week. The exact date within that week must be set in dialogue between the fleet owner and workshop. The maintenance planner can also output the maintenance protocols that shall be used for each occasion.

Heuristics and Propagators. The solver in the clp(FD) library is set such that it will return the first maintenance plan it finds that satisfies all the constraints. This means that the search heuristics are important for the behavior of the planner. The user can select between two search heuristics.

The first heuristic uses the built in parameters of the clp(FD) library so that the variable, not yet assigned, with the smallest domain is chosen and the maximum value of its domain is selected.

The second heuristic is specific for this problem formulation. Variables to assign are selected in the same way as the first heuristic, but the value selection is different. If assigned values corresponding to different maintenance points lie



	A	B	C
1	Plan for vehicle with VIN:	2058182	Note that a plan can be s
2	Plan activation date:	2013-04-16	
3	Plan activation milage:	900065	
4	Plan based on Km/week:	7103	
5	Plan score:	54	
6			
7	Approximate Date	Milage	Estimated Time (Hours)
8	2013-05-14	928477	2,19
9	2013-06-11	956889	0,83
10	2013-07-09	985301	3,38
11	2013-08-06	1013713	1,9
12	2013-09-03	1042125	0,34
13	2013-10-01	1070537	0,25
14	2013-10-29	1098949	1,91
15	2013-11-26	1127361	1,39
16	2013-12-24	1155773	0,34
17	2014-01-21	1184185	1,91

Fig. 2. An example of the maintenance plan of a vehicle.

within the domain of the selected variable, we select the assignment with the highest value. If such a value does not exist, the variable is assigned to the highest value in its domain like the first heuristic. The motivation for this heuristic is that we want to co-locate maintenance points in time so that we get as few maintenance occasions as possible.

Pseudo-code for the value selection heuristics are shown in Algorithm 1. The search heuristic takes a constraint store as input and returns a modified constraint store. The function ValSel uses the constraint store and the variable with the smallest domain to assign a value to the variable using the function GetBestValue. The function GetBestValue uses the constraint store and the variable with the smallest domain and returns either the highest value of the intersection between the selected variable and any other assigned variable or, if no such intersection exists, returns the highest value in the domain of the selected variable. The function Max returns the highest value of a domain and the functions GetNextAssigned iterates over assigned variable in constraint store and returns false when all has been shown. The function Intersect returns the elements in the intersection and the function FirstBound returns true if it is the first time its argument variable is assigned and false otherwise.

A new propagator was implemented for the controlling the maximum time of each occasion. This propagator is executed whenever a variable is assigned a value. Each maintenance point has a standard time associated with it, i.e. the time for the mechanic to complete the maintenance point task. The propagator has a week-time-list where it keep track of current summarized work time for each week up to the horizon. For each new assignment this list is updated, and each variable that is not assigned, is checked one at a time, if the summarized standard time exceed the time limit or not. If the time limit is breached, the week is removed from the variable's domain. If the propagator cannot exclude a

Algorithm 1 Search heuristic for few maintenance occasions

Inputs: $constraintStore, selVar$
Outputs: $constraintStore$

function ValSel($constraintStore, selVar$)
  $tVal \leftarrow$ GetBestValue($constraintStore, selVar$)
  if FirstBound($tVal$) **then** $selVar \leftarrow tVal$
  else
    $selVar \neq tVal$
    $selVar \leftarrow$ GetBestValue($constraintStore, selVar$)
  end if
  return $constraintStore$
end function

function GetBestValue($constraintStore, selVar$)
  $tempVal \leftarrow 0$
  $maxVal \leftarrow 0$
  while $var \leftarrow$ GetNextAssigned($constraintStore$) **do**
    $intSect \leftarrow$ Intersect($selVar, var$)
    if $\emptyset \neq intSect$ **then**
      $tempVal \leftarrow$ Max($intSect$)
      if $tempVal > maxVal$ **then** $maxVal \leftarrow tempVal$
    end if
  end while
  if $maxVal > 0$ **then**
    return $maxVal$
  else
    return Max($selVar$)
  end if
end function

value from any variable's domain the propagator fails. The pseudo-code for the propagator are shown in Algorithm 2.

The input is the constraint store, maximum time, week-time-list and the assigned week, the output is a possibly modified constraint store. The function SumAssignedStandardTimes uses week-time-list and the assigned week to update the week-time-list with the standard time of the corresponding maintenance point. The function GetNextUnAssigned iterates over the constraint store and returns the next not assigned variable. The function Intersect returns the intersection between the variable and week. GetStTime returns the standard time for the variable, i.e. a maintenance point standard time. GetNext iterates over a week-time-list and if no value are left return false. ReVal removes the current week from the variable domain and, finally the function NoMoreUnAssigned returns true if no more unassigned values are left to iterate over in the constraint store.

Algorithm 2 Maximum time propagator

Inputs: $constraintStore, week, weekTimeL, maxTime$
Output: $constraintStore$

$weekTimeL' \leftarrow$ SumAssignedStandardTimes$(weekTimeL, week)$
repeat
  $var \leftarrow$ GetNextUnAssigned$(constraintStore)$
  $intSect \leftarrow$ Intersect$(week, var)$
  if $\emptyset \neq intSect$ **then**
    $varWeekTime \leftarrow$ GetStTime(var)
    if $maxTime < varWeekTime$ **then**
      $var' \leftarrow$ ReVal$(var, week)$
      if $var' = \emptyset$ **then return** $Fail$
    end if
  end if
until NoMoreUnAssigned$(constraintStore)$
return $constraintStore$

4.3 Implementation

The maintenance planner is implemented in SICStus Prolog [14] using the clp(FD) library for Constraint Logic Programming over Finite Domains [3]. Users interact with the planner through a simple command prompt interface providing functions for setting certain constraints, creating maintenance plans, and outputting maintenance protocols. A screenshot from the user interface is shown in Figure 3. The user has the ability to create new maintenance plans, update existing ones, view maintenance history, and set various settings.

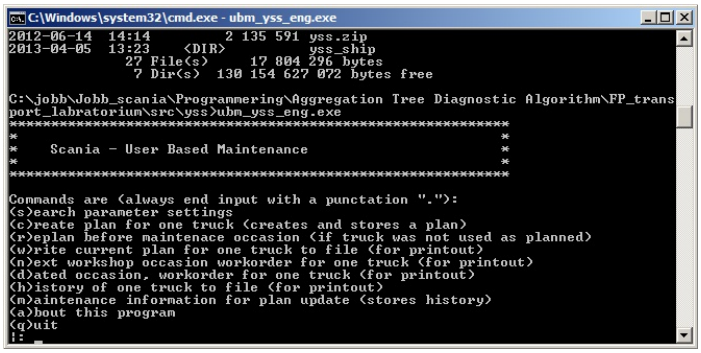


Fig. 3. Main menu of the user interface.

4.4 Typical Usage Pattern

During the PoC at the Scania Transport Laboratory, the typical usage of the maintenance planner was as following:

- One week before a scheduled maintenance occasion, the workshop planner checks using telemetry the actual mileage of the vehicle and regenerates the maintenance plan. If the mileage is lower or higher than expected, fewer or more maintenance points needs to be done at this occasion.
- With the content of the next maintenance occasion fixed, the workshop planner prints out the maintenance protocol and orders the parts needed for the occasion as specified by the protocol.
- When the vehicle is at the workshop, a mechanic performs maintenance according to the protocol.
- After the maintenance, the workshop planner reports back into the system which maintenance points that were addressed and creates a new updated plan for the vehicle. When the new maintenance plan is created the workshop planner may use the current mileage per week or manually set it to a new value if a different driving behavior is anticipated in the future.

5 Results

The vehicles that participated in the study were all of similar type and had similar driving patterns. Table 1 shows a comparison between S, M, and L plans and the automated planner for a representative vehicle from the Scania Transport Laboratory. This vehicle is a long haulage truck that has the expected usage of 6 760 km per week. The interval between maintenance occasions with the S, M and L program for such a vehicle is once every 90 000 km, or once every 13.3 weeks. For each maintenance occasion we have reported the standard time for completing all maintenance points scheduled for the occasion. Despite that the S, M, L program neither respects the periodicity or the time limit constraints

the sum of all standard times is higher. This is because the intervals for certain maintenance points could be stretched further when it no longer has to fit within the S, M, and L modules. The gain is potentially much larger, because for this study only a handful of the maintenance points had their intervals re-evaluated while most were the same as in the S, M, L program (90 000, 180 000, or 360 000 km).

Table 1. Comparing standard method and new automated planner.

	Standard	New Automated Planner
Visits	8	11
Total Time	33.4	28.9

The experiment with the haulage contractor was intended as a PoC of generating maintenance plans automatically at a much finer granularity than before. The haulage contractor requirements on the maintenance plan that could not be satisfied with the previous planning method. For example, the largest maintenance module, L, takes longer than the required maximal four hours of stand still.

6 Thoughts on the Next Generation Planner

The maintenance planner should, in the future, be extended to do optimization instead of only returning the first solution. Hence we have started doing some experiments using this set up. This section is dedicated to describing our current findings in this experimental work.

The maintenance planning problem, as formulated, has few constraints associated with it. Therefore, we must resort to using search while exploring possible solutions. Using multi-core parallel search will let us explore a larger part of the solutions space in a smaller amount of time than a single-core solution.

We choose to use the Gecode [13] toolkit for exploring a possible multi-core solution. The reasons for this choice is that Gecode has built-in support for parallel search, using a variant of work stealing [4] for handling and assigning computational tasks to idle threads without more involvement from the user than selecting the number of threads to be used. Gecode also has an excellent performance record, winning the MiniZinc Challenge [15, 9].

Our intention is to use branch-and-bound techniques for optimization. So far we have implemented an objective function to test the branch-and-bound approach. The objective function allows users to set their preferences for few maintenance occasions or minimizing waste using weights, where waste refers to not utilizing maintenance point intervals fully:

$$cost = noOccassions \times occasionsWeight + waste \times wasteWeight. \quad (7)$$

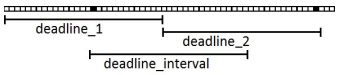


Fig. 4. Deadline interval constraint for the matrix problem formulation.

When using a multi-core solution, it is important how much memory a particular problem formulation needs, because it will need multiple copies of the constraint store. For our problem description we have identified two different problem formulations. We will describe them and run experiment focusing on their respective memory and CPU time needs.

Matrix Problem Formulation. One way to formulate the problem is to create a matrix with the same number rows as there are maintenance points and the same number of columns as there are days within the planning horizon.

- Variables. For each cell in the matrix a constraint variable is defined with domain between zero or one. The assigned value at a specific row and column denotes whether the maintenance point associated to the row should be performed during the day associated to the column. A one means that the maintenance point should be performed and a zero means that it should not.
- Constraints. Maintenance point intervals for a row are regulated by a constraint that all cells in the row up to the deadline interval should sum to 1. Hence only one maintenance occasion should occur within interval, as is shown in equation 8. This constraint is repeated for each interval up to the planning horizon.

$$SumToOne : \sum_{i=0}^{interval} cell_i = 1 \tag{8}$$

This is however not enough to ensure that the plans are correct because it can be the case that a solution ends up with a distance between two one's (1) that is greater than the interval. This is illustrated in Figure 4. Here two constraints are set, where each of their interval must sum to one but still we can end up with a plan that violates the deadlines.

One way of correcting this is to add a constraint which keep track of the number of zeros in a row and make sure that the length of the zeros do not exceed the deadline. This constraint is illustrated in equation 9, where zerosInRow takes the i:th row and returns the maximum consecutive number of zeros and, maxZeros is the limit for consecutive zeros on this row.

$$zerosInRow(row_i) < maxZeros_i. \tag{9}$$

Unable to express this constraint as an extensional constraint, we implemented a new propagator. The propagator uses the notion of 'blocks' of zeros,

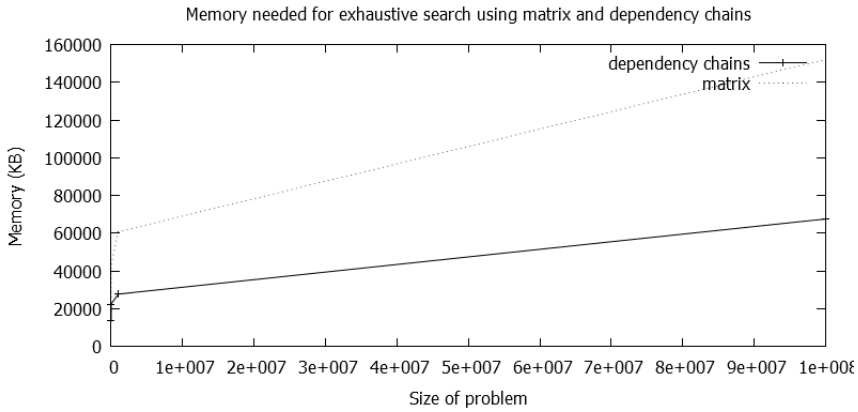


Fig. 5. Comparison of memory needs.

which can be merged if two blocks are adjacent to each other, creating a bigger block. New blocks are created when a new zero is assigned that has no adjacent blocks. The propagator first checks the constraint store so that no deadline interval is breached. Then it propagate a one if there exists a block with the same length as the interval minus one. With this propagator the planner behaves as desired.

As with the previous planner, the user can specify the minimal interval with a parameter.

Dependency Chains. The second problem formulation is the same as the one used for the first single-core maintenance planner. Each variable corresponds to an occasion of a maintenance point and the value corresponds to the time when this occasion should occur.

Experiment. We have implemented both formulations of the problem in Gecode 3.7.3 and examined their CPU usage and memory needs when conducting an exhaustive search for a planning problem with one maintenance point and a maintenance interval of 10 days. The planning horizon varied from 20 days to 80 days, which causes the total number of possible solutions to vary from 100 to 100 million.

They executed on a quad-core Intel i7-2760QM processor running at 2.4 GHz. The comparison of the memory need is shown in Figure 5 and the comparison of the CPU times is shown in Figure 6.

As expected, the matrix problem formulation consumes more memory and more CPU time to complete the exhaustive search problems. In this experiment we had only one maintenance point and the maximum planning horizon was 80 days. In a realistic setting we have almost a hundred maintenance points and we need a planning horizon of up to 365 days. Thus in our application using the

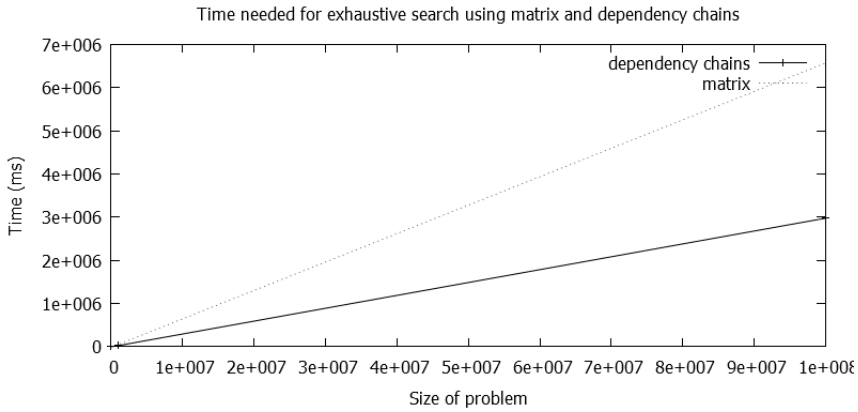


Fig. 6. Comparison of CPU time.

dependency chain problem formulation is preferable to using the matrix problem formulation.

7 Discussion

Scania gained much experience from developing a maintenance planning system for its internal haulage contractor, both in terms of what functionality a workshop planner needs and how constraint programming can be used to realize this functionality. Based on this new knowledge, work has begun with the next generation of the maintenance planner, as already mentioned. Apart from improved user interface we will implement more ways for the user of the planner to influence what constitutes good plans. Here we are considering more constraints and optimization parameters. Furthermore, the group of intended users of the planner is expanded to include fleet planners and sales personnel. We also want to do more exact planning and have decided to use a time scale of days instead of weeks for this next generation of the planner.

Development The development of the planning prototype was not more difficult than a single developer familiar with constraint programming techniques could design and implement the entire application in a few months. However, more effort was required from the experts that had to set maintenance intervals since this was made manually. The intention is that in the future these intervals will be set automatically based on data.

When the application was delivered to the workshops, a Scania engineer created the first plan for all the vehicles based on a default expected mileage. The users at the workshops were only supposed to use the application to read out maintenance plans and protocols and to input the actual times and mileages when each maintenance point is performed. Early in the experiment, it became

evident that a re-plan functionality was needed to make sure that the maintenance plan was correct at the designated date for maintenance. Usually the workshop planner re-planned the schedule for a vehicle scheduled a couple of days before the planned maintenance occasion. After this functionality had been added we saw that sometimes planned maintenance occasions could be avoided due to less vehicle usage than predicted. In some cases it was the opposite and more maintenance was needed than in the previous plan.

Another appreciated functionality that was added later was the possibility to output a list of consumable goods (e.g. oil quantities and part numbers of filters) together with the maintenance protocol. This made it possible for the workshop to make sure that all necessary products were in place in time for the maintenance occasion. This was not as important with the fixed maintenance protocols since the list of consumable goods are always the same.

Release to Customers Because of the rather primitive user interface, education of the managers responsible for planning at the workshop, was important for the users to understand how the system worked. As a part of this, users were encouraged to create simulated maintenance plans using the system.

One user at the workshops was assigned as superuser. The superuser and developer had regular meetings where they could discuss problems and questions regarding the application. The superuser could collect opinions about the system from the other users and also educate them. During the first two months many changes to the application were made because of feedback from the users.

Initially the program was unstable and would crash if fed with illegal input or if the constraints were set so that no valid plan existed. Also there were problems with the dynamic creation of maintenance point variables causing unnecessary maintenance points to pile up at the end of the plan. This had no real consequence for the performance of the plans because the remainder of the plan was correct and the unnecessary maintenance points would always be pushed ahead whenever the maintenance plan was re-planned. However it looked bad and confused the users. All data such as previous maintenance and mileages were saved in Prolog using its program state. This prevented users from correcting erroneous input. Instead users were instructed to store and keep copies of previous entire program states to do roll-back upon if the system contained information that did not align with reality.

Once these and other problems were corrected and the users had become acquainted with the program, we started getting positive feedback from the users. Some users even preferred the command prompt interface over a graphical user interface because interaction with the program was really fast.

Apart from the obvious fact that we could not satisfy the requirement of having fixed maintenance occasions every fourth week with a time limit of four hours, using the standard maintenance program, a preliminary comparison show that gains in time and money can be made with the new planning system.

8 Conclusion

The possibility to individually plan each maintenance point allows us to create more efficient maintenance plans that also consider the needs of the vehicle owner. Previously, many of these needs have been disregarded. A purpose of the PoC was to gain a better understanding of these needs and the potential gain of creating an automated maintenance planner based on constraint programming techniques.

The PoC was developed with limited resources, but the testers at the Scania Transport Laboratory were tolerant and put up with the initially buggy planner and gave back precious feedback on how to develop it further. This way of working is fine when, as in this case, the test group is a small group of users belonging to a subsidiary company. However, it would not be feasible for a test on a larger scale. We learned that, even with a primitive maintenance planner such as this PoC, the maintenance costs can be significantly reduced and user preferences that previously were ignored now can be regarded.

The PoC maintenance planner showed us that a planner based on constraint programming techniques is a good way to go. Constraint programming is a good framework for expressing and solving combinatorial problems for which human capabilities are not sufficient to cope with. The maintenance planning problem, as it has been formulated so far, has not been very constrained and had more characteristics in common with a search problem. However, if more user constraints are added, the planning problem may well prove to move from a problem with a dense distribution of solutions to one where they are sparse. Then the benefit of constraint programming may become even greater.

For example, we may want to consider when and where there is a workshop that can perform maintenance on a vehicle. This may make it necessary to extend the planning to multiple vehicles. Also the fleet planner may have its requirements on where the vehicles must be at given times and how far and fast they can travel.

Methods for estimating the remaining useful life of components to create predictive models for maintenance is currently under investigation at Scania. The maintenance interval of a component would then change dynamically, which could affect the stability of maintenance plans. This may not be a desired behavior of the planner and therefore the maintenance planner must support constraints regulating how an existing plan may change when new input arrives.

For all these possible extensions to the maintenance planning problem, a solution based on constraint programming appears the most promising.

9 Acknowledgments

This work has been funded by Scania CV AB and the Vinnova program for Strategic Vehicle Research and Innovation (FFI).

References

[1] Ian K Jennions et. al. *Integrated Vehicle Health Management: Perspectives on an Emerging Field*. Ed. by Ian K Jennions. SAE, 2011.

[2] J. Christopher Beck and Philippe Refalo. "A Hybrid Approach to Scheduling with Earliness and Tardiness Costs". In: *Annals OR* 118.1-4 (2003), pp. 49–71.

[3] Mats Carlsson, Greger Ottosson, and Björn Carlson. "An open-ended finite domain constraint solver". In: *Programming Languages: Implementations, Logics, and Programs*. Ed. by Hugh Glaser, Pieter Hartel, and Herbert Kuchen. Vol. 1292. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 1997, pp. 191–206.

[4] Geoffrey Chu, Christian Schulte, and Peter J. Stuckey. "Confidence-based Work Stealing in Parallel Constraint Programming". In: *Fifteenth International Conference on Principles and Practice of Constraint Programming*. Ed. by Ian Gent. Vol. 5732. Lecture Notes in Computer Science. Lisbon, Portugal: Springer-Verlag, Sept. 2009, pp. 226–241.

[5] Tom Creemers et al. "Constraint-Based Maintenance Scheduling on an Electric Power-Distribution Network". In: *Proc.of the Third International Conference on the Practical Application of Prolog*. Paris, 1995, pp. 135–144.

[6] Safaai Deris, Sigeru Omatu, and Hiroshi Ohta. "Timetable planning using the constraint-based reasoning". In: *Computers & Operations Research* 27.9 (2000), pp. 819 –840. ISSN: 0305-0548.

[7] Jon Dunsdon and Mark Harrington. "The Application of Open System Architecture for Condition Based Maintenance to Complete IVHM". In: GE Aviation.

[8] Sami Gabteni and Mattias Grönkvist. "Combining column generation and constraint programming to solve the tail assignment problem". In: *Annals OR* 171.1 (2009), pp. 61–76.

[9] MiniZinc organization. *MiniZinc Challenge*. Apr. 2013. URL: `http://www.minizinc.org/`.

[10] José Palma et al. "Scheduling of maintenance work: A constraint-based approach". In: *Expert Syst. Appl.* 37.4 (Apr. 2010), pp. 2963–2973. ISSN: 0957-4174.

[11] Scania CV. *Scania Fixed Price Repair programme extended*. Apr. 2013. URL: `http://www.scania.co.uk/about-scania/media/press-releases/2009/fixed-price-repair-programme-extended.aspx`.

[12] Scania CV. *Scania Repair & Maintenance contract*. Apr. 2013. URL: `http://www.scania.com/products-services/services/workshop-services/repair-maintenance-contract/`.

[13] Christian Schulte, Guido Tack, and Mikael Z. Lagerkvist. *Modeling and Programming with Gecode*. 2010. URL: `http://www.gecode.org/doc-latest/MPG.pdf`.

[14] *SICStus Prolog User's Manual*. Intelligent Systems Laboratory, Swedish Institute of Computer Science. 2013.

[15] Peter J. Stuckey, Ralph Becket, and Julien Fischer. “Philosophy of the MiniZinc challenge”. In: *Constraints* 15.3 (July 2010), pp. 307–316. ISSN: 1383-7133.