1-1-2016

Optimizing Vehicle Usage Using CSP, SAT and MAX-SAT

Raheem T. Al Rammahi
University of Denver

Optimizing Vehicle Usage Using CSP, SAT and MAX-SAT

Abstract

Most of the companies in Iraq spend significant amounts of time and money when transferring employees between home and work. In this thesis, we model the problem of the Dhi Qar Oil company (DQOC) transportations using three modeling languages from AI: Constraint Programing (CP), Boolean Satisfiability (SAT), and Maximum Satisfiability (MAX-SAT). We then use solvers to find optimal solutions to this problem.

We show which of these solvers is more efficient when finding optimal solutions. For this purpose, we create a test suite of 360 problems to test these solvers. All solvers are applied to these problems and the final efficiency is shown.

Document Type

Thesis

Degree Name

M.S.

Department

Computer Science

First Advisor

Nathan R. Sturtevant, Ph.D.

Second Advisor

Ronald DeLyser

Third Advisor

Anneliese Andrews

Keywords

CSP, Communicating sequential processes, MAX-SAT, Maximum satisfiability, Optimization, SAT, Boolean satisfiability

Subject Categories

Computer Sciences

Optimizing Vehicle Usage using CSP, SAT and MAX-SAT

A Thesis

Presented to

the Faculty of the Daniel Felix Ritchie School of Engineering and Computer Science

University of Denver

in Partial Fulfillment

of the Requirements for the Degree

Master of Science

by

Raheem T. Al Rammahi

June 2016

Advisor: Nathan R. Sturtevant

Author: Raheem T. Al Rammahi
Title: Optimizing Vehicle Usage using CSP, SAT and MAX-SAT
Advisor: Nathan R. Sturtevant
Degree Date: June 2016

Abstract

Most of the companies in Iraq spend significant amounts of time and money when transferring employees between home and work. In this thesis, we model the problem of the Dhi Qar Oil company (DQOC) transportations using three modeling languages from AI: Constraint Programing (CP), Boolean Satisfiability (SAT), and Maximum Satisfiability (MAX-SAT). We then use solvers to find optimal solutions to this problem.

We show which of these solvers is more efficient when finding optimal solutions. For this purpose, we create a test suite of 360 problems to test these solvers. All solvers are applied to these problems and the final efficiency is shown.

Acknowledgements

I would like to express my special appreciation and thanks to my advisor Dr. Nathan Sturtevant. I would like to thank you for instructing me how to write a thesis. I would also like to thank my committee members: Dr. Ronald DeLyser, Dr. Anneliese Andrews, and Dr. Nathan Sturtevant for serving as my committee members and providing useful suggestions about this thesis. Special thanks to my family: my grandmother, mother, father, and my wife Shaymaa for their sacrifices for me. I would also like to thank the Higher Committee for the Education Development in Iraq (HCED) that supported me financially to complete my master's study.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction:

1.1 The Problem

Many fields in computer science have been used directly for solving real-world problems. One such field is artificial intelligence (AI). The goal of this thesis is to use work from the field of AI to address the problem of hiring cars in the Dhi Qar Oil company (DQOC). DQOC has many employees and engineers living in the cities around the Dhi Qar province. On a daily basis, they must move from their homes to their workplaces and the oil fields, which are far from any city. The company is responsible for hiring a car for each employee to pick him or her up from home in time to arrive at work for his or her shift and return him or her home after work.

Right now, they hire cars as follows:

They hire one car for each group of fewer than four employees that they can move from the same city to the same workplace at the same time. They keep the car with them until the work day is finished. As a result, DQOC hires an enormous number of cars that are largely idle during the day. Cars are currently scheduled by hand at great cost. For example, DQOC sometimes hires a car for just one employee for the whole day. Therefore, the question of this thesis is how to optimize the use of each car, with the goal of decreasing the number of hired cars and the total cost of hiring these cars.

1.2 The Suggestion Solutions, The Optimizing Process

The central idea of how to optimize the use of each car and reduce the total cost to make each car serve more than one employee or more than one group of employees. The optimization is executed assuming that the times for starting work and for departure for all the staff are known, along with the roads between the cities and the workplace locations.

Additional planning can further optimize the usage of the hired cars. For example, if a human hires the cars arbitrarily, he or she may hire many cars in a city that contains few employees and hire few cars in a city that contains many employees. This leads to inefficient use of the cars, as they must travel before they can pick up passengers. Therefore, by making the system distribute the cars between the cities as needed, the total number of cars can be reduced.

In order to build a system that can optimize the use of the cars, reduce the total number vehicles required, and meet all the requirements, we model this problem with three different artificial intelligence modeling languages: Constraint Programming, Boolean Satisfiability (Boolean SAT), and Maximum Satisfiability (MAX-SAT). We use these solvers to find optimal solutions to the transportation problem and show that with our formulation we are able to find optimal solutions most efficiently with a CP solver.

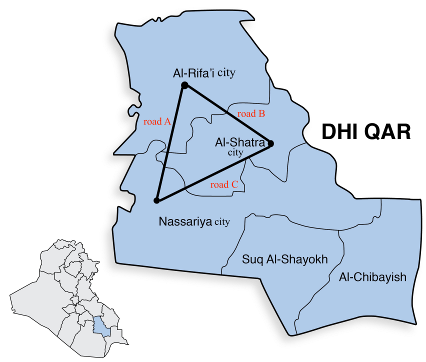


Figure 1.1: [16], Some Dhi Qar Roads Map:
This map is example to show that the short roads is not always useful for optimizing, it is not a real road map for Dhi Qar.

Chapter 2

Problem Definition

The Dhi Qar Oil Company has a number of oil fields scattered on the outskirts of the cities of the province of Dhi Qar; Figure 1.1 shows the map of Dhi Qar province.

All company staff travel between these fields, the company's administrative departments, and the cities in which they live. Many of the employees' traveling times are known, but some of them are not known for special reasons (e.g., emergency workers). The company is responsible for all the transportation of the employees, so they hire vehicles for this purpose. The scheduling and allocating vehicles for each employee are currently performed by hand. Therefore, DQOC hires many more cars than are needed. To reduce the number of hired cars and optimize the use of each hired car, we must consider the people, the environment, and the resources.

- The people: we have to consider a schedule just for the employees who have a known traveling time. If there is no regular work schedule, there is no way to make a travel schedule. We call the employees who have known traveling times *passengers*.
- The environment: environment is places and times people work.

- We call each of the cities in Figure 1.1, the administrative departments, and the oil fields, a *Car Station*. Each of these *car stations* is connected with at least one another *Car Station* by a road. The connected *car stations* are called *Adjacent Car Stations*.
- The company divides the work day into several time units; each time unit called $Time_i$

- The resources: the hired cars can travel between the *Adjacent Car Stations*. They start in different cities depending on where the passengers live. The hired cars always start work at 8 am; they are hired on a daily basis.

2.1 Constraints in the Problem

There are many constraints that should be considered to solve this problem:

- The passengers have a fixed starting and departure time. The passengers should arrive on time to begin their work and to allow the previous shift to depart in the shift system. Time arrival is the responsibility of the company. They must hire a car for each passenger, and the company must have cars ready to take the passengers home when they set off work. Each passenger has a home location and a work location. The passenger can switch between cars during the trip to work or home. Also, the passengers are free to take their own cars after work, but they must notify the company ahead of time. Any passenger can leave the hired car during his trip to home. If the employee requests permission to leave work early for special circumstances, the company is only responsible for the transfer of that employee in the case of sick leave. If any employee does not come to work for the whole day if she or he is sick for instance, she or he should notify the company in the previous work day. Otherwise, they will hire a car for that employee. Also, there are some special constraints like if one employee and his wife are both employees in the company, they could request to be transferred in the same car if they are in the same workplace, but we are not to consider these constraints because they occur very rarely. Other rare constraints, include if employees request personal protection during their travel to and from work (general manager of the company, and heads of departments, for example). The company also sometimes send an employee to another province (usually Basra or Baghdad), and they hire a car for him or her.

- The hired cars have a fixed starting time. Because the company hires cars on a daily basis, a car finishes after dropping off the last passenger. The start locations of all the cars are known in one of the Dhi Qar province cities or the oil fields; the company tells the hired cars where to start depending on passengers density in each city and each oil field, but who decides the starting location of the cars is not part of the problem. The capacity of each car is less than four passengers at any time. The cars must follow the roads between the cities and the oil fields. In addition to public roads, there is a network of private roads. In other words, the cars can travel just between the adjacent cities or oil fields. Each car can service multiple passengers at one time, even if they are from different cities, if they can be brought to their destinations in time.

- The cost is the number of the hired cars. The company hires cars on a daily basis and the cost on a daily basis is the same for all the cars.

- The primary goal is to find a feasible schedule that shows the times and the locations of all the passengers and all the cars during the day, as well as which car will service which passengers. A secondary goal is to minimize the total number of hired cars.

In the next three chapters, all of these constraints must be taken into consideration to find a solution for the problem using CSP, SAT, and MAX-SAT solvers.

Chapter 3

Formulate as Constraint Program

3.1 Define CP

Constraint Programming is a programming paradigm that focuses on the variables, values, and constraints of the problem. The goal is to find a feasible solution to a problem rather than finding the optimal solution. CP has been used for many areas of logic programming, optimization, and others [5].

The constraint program consists of a set of variables, domains which are the possible values that each variable can take, and a set of constraints C = $\{c_1, ..., c_m\}$ on the values that a variable can take. The constraints depend on the properties of a solution to be found.

A constraint consists of a finite number of variables c$(x_1, ..., x_n)$. Each variable x_i has a domain D_i which consists of a finite number of possible values $(v_1, ..., v_n)$. The constraint is a relation between the set of domains D. A constraint problem is satisfiable or solvable if each variable takes one of the values in its domain and satisfies all the constraints in the problem. If the problem is solvable, then we can apply a function F on all or some of the variables to maximize or minimize the solution. The constraints are classified into many types. One such type is a global constraint. Global constraints are the constraints that apply to all or some of the variables to specify a relation between them; for instance, the *alldifferent* constraint means that all variables under that constraint should take distinct values. One of the other types is an arithmetic constraint. Arithmetic constraints are the constraints that make arithmetic relations between two or more variables like linear or nonlinear equations and inequalities. For example, if there are three variables A, B, and C, one of the arithmetic constraints in these variables A= B + C, or C≠B. Also, arithmetic constraints can apply to one variable, for instance, C≠0.

3.2 Map Problem to CP

To map the problem to CP, we have to specify the variables, domains and the constraints in the problem.

3.2.1 The Problem Variables and the Domains

Suppose the company has n *passengers*; the company can hire up to m *cars* to transfer those n *passengers*; also it divides the day into t *times*, and there are c *car stations*.

Let us symbolized for *passengers* by P, *cars* by C, *times* by T, and *car stations* by S

So the list of variables as follows:

- **Passengers variables**: to build a reliable schedule, the company should keep track of where each passenger is at all times. Therefore, they should know where each P is at all times (t times). passengers variables are in the form P_{ij} where i is the passenger's id, and j is the time. For example, if $n = 2$ and $t = 3$, the passengers variables will be as follows:

  For P_1 there are three variables which are P_{11}, P_{12}, P_{13}. Where P_{11} means

$passenger_1$ at $time_1$,P_{12} means $passenger_1$ at $time_2$, and P_{13} means $passenger_1$ at $time_3$.

For P_2 there are three variables also P_{21}, P_{22}, P_{23}.

In general, for any n *passengers* and t *times*, the variables would be:

P_{11},P_{12},P_{13},...,P_{1t}

P_{21},P_{22},P_{23},...,P_{2t}

$\vdots$

P_{n1},P_{n2},P_{n3},...,P_{nt}.

So there are $n \times t$ variables for the *passengers*.

The **domain** of these $n \times i$ variables is the c *car stations*. This means that for each variable, there are c possible values.

- **Cars variables**: if the company wanted to keep track of where each car is at all times (t times), they should know where each C is at t times. cars variables are in the form C_{ij} where i is the car's number, and j is the time. For example, if $m = 2$ and $t = 3$, the cars variables will be as follows:

  For C_1 there are three variables which are C_{11},C_{12},C_{13}.Where C_{11} means car_1 at $time_1$,C_{12} means car_1 at $time_2$, and C_{13} means car_1 at $time_3$.

  For C_2 there are three variables also C_{21},C_{22},C_{23}.

  In general, for any m *cars* and t *times*, the cars variables would be:

  C_{11},C_{12},C_{13},...,C_{1t}

  C_{21},C_{22},C_{23},...,C_{2t}

  $\vdots$

  C_{m1},C_{m2},C_{m3},...,C_{mt}.

  So there are $m \times t$ variables for the *cars*.

  The **domain** of these $m \times t$ variables is the c *car stations*. This means that for each car variable, there are c possible values.

- **Flag variables**: there are two types of flag variables, one for *passengers*, and the other for *cars*.
    - **Passengers flag variables (PFV)**: each *passenger* should be either in a *car station* or in a *car*. If that *passenger* is in one of the *cars* at a particular time, then there is a boolean variable to indicate that. PFVs are in the form PFV_{ijk} where i is the passenger's id, j is the car's number, and k is the time. For each of the $n \times t$ passengers variables, there are m variables, such that PFV_{111} means P_1 in C_1 at T_1, PFV_{121} means P_1 in C_2 at T_1,PFV_{nmt} means P_n in C_m at T_t.
    There are $n \times m \times t$ PFV variables. The **domain** of these $n \times m \times t$ variables is (0,1).
    - **Cars flag variables (CFV)**: if at least one of the n *passengers* is in one of the m *cars* at any *time*, that means that *car* is used for some time. Therefore, there is one boolean variable per car to show if it is used or not.

As a summary, the total number of variables is

$$\begin{aligned} &= P \quad + \quad C + PFV + CFV \\ &= (n \times t) \quad + \quad (m \times t) + (n \times m \times t) + (m) \end{aligned}$$

3.2.2 The Problem Constraints

Since the computer does not understand how to make each car service a particular number of passengers, nor how the cities are connected, the system should be built with some constraints. There are six main types of constraints in the problem:

- **Initial constraints:** for the system to allocate a car for a particular passenger, it should know at least where that passenger is at the beginning time and

the departure time. So P_{p1},P_{pt} where p=1 to n, which means $\forall$ P, T_1 and T_t should be already known. In addition, if other *times* are known, they should be set; for example, if the P_1 will stay in $carstation_8$ for the first five *times* then P_{11}=8, P_{12}=8, P_{13}=8, P_{14}=8, P_{15}=8.
So there are at least $2 \times n$ initial constraints for the *passengers*.

For the *cars*, just the beginning time should be set, and then the system lets the *cars* go anywhere, so there are m initial constraints for the *cars*.

- **Adjacent constraints:** since the *cars* can only move from a *car station* to its adjacent *car stations*, we have to Figure out the adjacency constraints. Suppose d is the number of the adjacent *car stations* for a *car station* s, and the adjacent *car stations* $= \{s_1, s_2, s_3, ..., s_d\}$, then if the car i at the *car station* s at $time_k$, it must be at *car station* s or at one of its d adjacent *car stations* at $time_{k+1}$.

$$(C_{11} = s) \implies (C_{12} = s) \lor (C_{12} = s_1) \lor (C_{12} = s_2) \lor (C_{12} = s_3) \lor \ldots \lor (C_{12} = s_d)$$

$$(C_{12} = s) \implies (C_{13} = s) \lor (C_{13} = s_1) \lor (C_{13} = s_2) \lor (C_{13} = s_3) \lor \ldots \lor (C_{13} = s_d)$$

$$(C_{13} = s) \implies (C_{14} = s) \lor (C_{14} = s_1) \lor (C_{14} = s_2) \lor (C_{14} = s_3) \lor \ldots \lor (C_{14} = s_d)$$

$$\vdots$$

$$(C_{1(t-1)} = s) \implies (C_{1t} = s) \lor (C_{1t} = s_1) \lor (C_{1t} = s_2) \lor (C_{1t} = s_3) \lor \ldots \lor (C_{1t} = s_d)$$

So there are (t-1) adjacent constraints for only $car1$ and the same number of adjacent constraints for each car, therefore, the total adjacent constraints is $(t-1) \times m$.

- **Capacity constraints:** each *car j* has its capacity, if the $capacity = r$, and because the PFV are boolean we have to be sure that the summation of the PFV at time k for all the *passenger* is less than or equal r

$$PFV_{1jk} + PFV_{2jk} + PFV_{3jk} + ... + PFV_{njk} = \sum_{x=1}^{n} PFV_{xjk} \leqslant r \quad (3.2.1)$$

So there are t capacity constrains for each car. The total number of the capacity constrains are $m \times t$.

- **One car at a time constraints**: this type of constraint is to prevent the *passenger* from being in two or more cars at the same time. If there are m *cars*, the form of this constraint for *passenger p* at time k is:

$$PFV_{p1k} + PFV_{p2k} + PFV_{p3k} + ... + PFV_{pmk} = \sum_{j=1}^{m} PFV_{pjk} = 1 \quad (3.2.2)$$

So there are t one car at a time constraints for each passenger. The total number of the one car at a time constraints are $n \times t$.

- **Traveling constraints:** for any *car i* and *passenger p* at *time k*, the *flag variables* would be $True$ if $C_{ik} = P_{pk}$ and $C_{i(k+1)} = P_{p(k+1)}$

$$PFV_{pik} \implies (C_{ik} = P_{pk}) \wedge (C_{i(k+1)} = P_{p(k+1)}) \quad (3.2.3)$$

Since there are $n \times m \times t$ PFV (see PFV on page 11), the total number of **traveling constraints** is $n \times m \times t$.

- **Location constraints:** we must ensure that the *passengers* ride one of the *cars* and also move to the next *car station*. In other words, the case of the PFV gets $True$ depending on traveling constraints but the *passenger* stays in the same *car station*. Because the *passenger* either rides a *car* and transfers to another *car station* or does not ride a *car* and does not transfer, we need to consider the location Constraints. Suppose the *passenger* p in *car station* s at $time$ k, and j is the *car number* from 1 to m, then either all PFV_{pjk} are $false$ and $P_{pk} = P_{p(k+1)}$ or one of the PFV_{pjk} is $true$ and $P_{pk} \neq P_{p(k+1)}$, so the following constraints for only $passenger_p$ are:

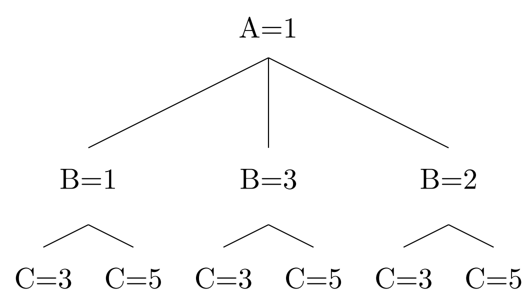
$$\begin{aligned}
(P_{p1} \neq P_{p2}) &\implies PFV_{p11} \vee PFV_{p21} \vee PFV_{p31} \vee ... \vee PFV_{pm1} \\
(P_{p2} \neq P_{p3}) &\implies PFV_{p12} \vee PFV_{p22} \vee PFV_{p32} \vee ... \vee PFV_{pm2} \\
(P_{p3} \neq P_{p4}) &\implies PFV_{p13} \vee PFV_{p23} \vee PFV_{p33} \vee ... \vee PFV_{pm3} \\
&\vdots \\
(P_{p(t-1)} \neq P_{pt}) &\implies PFV_{p1(t-1)} \vee PFV_{p2(t-1)} \vee PFV_{p3(t-1)} \vee ... \vee PFV_{pm(t-1)}
\end{aligned}$$

  So there are (t-1) location constraints for only one *passenger*, and the total location constraints $= n \times (t-1)$.

3.3 Solving Sample Problem by CP

The primary algorithm to solve the CPs is the backtracking which is simply assigned variables with values depending on the constraints in the problem, and then if all the variables have values the problem is solved. If one of the variables does not get assigned with a value that satisfied all the constraint then backtrack and try other values for the variables. For example, if there are three variables A=(1), B=(1,3,2), C=(3,5) and there is one constraint which is $B \geq C$, then to solve the problem assign A=1, B=1, C=3 but the value assigned for B does not satisfy the constraint $B \geq C$. So backtrack to assign A=1, B=3, C=3 and the problem is solved in that way. CP strategies could be applied to check if the problem of transportation in DQOC is solvable with a particular number of cars, then if it is solvable, apply an objective function to minimize the number of cars.

The constraint programming solver used to solve DQOC transportation problem is called Minion [7]. Minion can find the solution of the problem, and it can find all the solution if there are some solutions. If the problem is not solvable, it will report that. The technique used by Minion is called branching and propagation. The branching is to assign value to each variable with consideration to the values that other variables had taken. For the example above, the branching assignments will be in the following tree.



Propagation simplifies the problem by ignoring branches. For example, during the search for a solution in the example above, if there is a constraint (A$\neq$ B), the propagation here means ignore all the first branch of the tree above whatever its size.

3.3.1 Map Sample Problem to CP

Since the problem of transportation in DQOC is complex, the best way to describe it in CP is to take a small sample problem and build it in CP, then extend the solution of that sample to the general problem.

To formulate a sample problem, consider a company that has two *passengers* $\{P_1, P2\}$, the company can hire up to two *cars* to take these two *passengers*, $\{C_1,$

C_2}, and it divides the day into three *times*. To make them readable let us suppose the *times* are $\{9am, 10am, 11am\}$, and there are three *car stations* {S_1, S_2, S_3}. We have to establish the *variables* as in section 3.2.1.

There are four types of variables in Minion: DISCRETE, BOOL, BOUND, and SPARSEBOUND [4], where DISCRETE is for variables with integers domain, BOOL is for the variables with domain {0, 1}, BOUND is also for variables with integers domain, and SPARSEBOUND is for variables with a domain of an arbitrary range of integers. So the variables for this sample will be as follows:

- **Passengers**:

  p19 {1..3}, p110 {1..3}, p111 {1..3}, p29 {1..3}, p210 {1..3}, p211 {1..3} where, for example, p19 is the name of the variable for P_1 at $Time$ 9, and {1..3} is the *domain* of the variable p19 which means the *car stations*. So the type of that variable is DISCRETE.

- **Cars:**

  the same number of variables, the same domains, and the same type for the cars as for the passenger because n equals m and the possible values for this type of variable are the *car stations*. So the cars variables in this sample problem are c19 {1..3}, c110 {1..3}, c111 {1..3}, c29 {1..3}, c210 {1..3}, and c211 {1..3}.

- **Flag variables:**

  - Passengers Flag Variables (PFV): since there are two *passengers*, two *cars*, and three *times* and there is no need for the *passenger* in the last *time* in this case, so there are $2 \times 2 \times 2$ =8 PFV variables, which are p1c19, p2c19, p1c110, p2c110, p1c29, p2c29, p1c210, and p2c210, where, for example, p1c19 is the name of PFV of *passenger_1* in *car_1* at *time_1*, and there is no need to declare the domain if the type is **boolean**.

- Cars Flag Variables (CFV): in CP, the goal is just to check if the problem is solvable with this number of *cars* rather than to optimize the number of the *cars*, so there is no need to have (CFV).

Next we have to establish the **constraints** as in section 3.2.2. Constraint in Minion is the same as a function in programming languages and there are no nested constraints except in reify and reifyimply constraints [4]. So the constraints in this sample are as follows:

- **Initial constraints:** the initial constraints for *passengers* and *cars* in this sample are mentioned for every *passenger* when the location is known and for the *cars* at $time_1$ only. Minion has the constraint eq, in the form eq(x0,x1), which is equivalent to x0=x1. [4]. The initial values for this sample are as in Table 3.1 where "_" means unknown location.In this table, both *cars* start work at $carstation_3$ and both *passengers* live in $carstation_3$ but $passenger_1$ gets to work at $carstation_1$ by *time 11* and $passenger_2$ gets to work at $carstation_2$ by *time 10.* See the initial constraint part from the solution on page 20.

carNumber	at 9	at 10	at 11
1	3	-	-
2	3	-	-
passengers			
1	3	-	1
2	3	2	2

Table 3.1: The initial values for *passengers* and *cars*

- **Adjacent constraints:** this type of constraint in this sample depends on the map in Figure 3.1, so the *passengers* in **Al-Rifa'i** can transfer directly just to **Al-Shatra** and the same for the *passengers* in **Nassariya** but for *passengers* in

Al-Shatra they can transfer to **Al-Rifa'i** or **Nassariya** directly. Therefore, the adjacent constraint for C_1 at $time_1$ (for example) will be in the form:

$(C_{19} = 1) \implies (C_{110} = 1) \vee (C_{110} = 2)$

In logic, A $\implies$ B is the same as $\neg$A $\vee$ B, where $\neg$A denotes negation A, so the form of adjacent constraint above would be:

$(C_{19} \neq 1) \vee (C_{110} = 1) \vee (C_{110} = 2)$

in Minion there is no $\vee$ and no $\neq$ but there is a constraint called **watched-or** in the form watched-or(C1,...,Cn), which is equivalent to C1$\vee$C2$\vee$,...,$\vee$Cn, and a constraint called **diseq** in the form diseq(v0,v1) ensures two variables take different values [4]. Therefore, the adjacent constraint for C_1 at $time_1$ would be watched-or(diseq(c19,1),eq(c110,1),eq(c110,2)). See the adjacent constraints part from the solution on page 20.

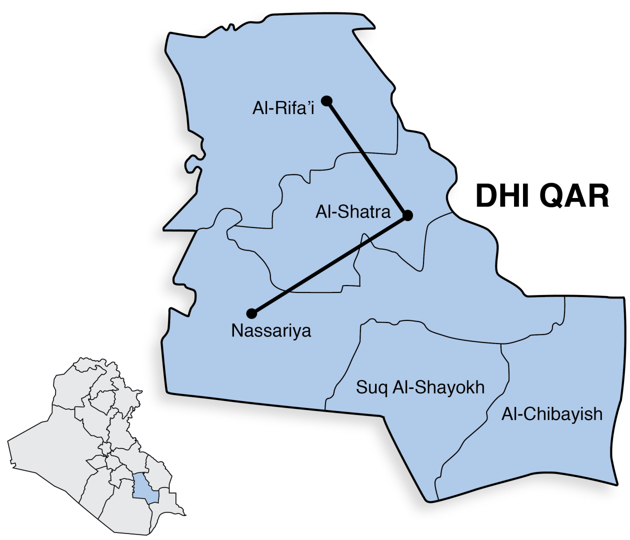


Figure 3.1: CarStations Sample Map

- **Capacity constraints:** this constraint is in the form $\sum_{x=1}^{n} PFV_{xct} \leqslant r$ so for car_1 for example, the capacity=r=1,and $time_1 = 9$ then

$\sum_{x=1}^{2} PFV_{x11} \leqslant 1 \Rightarrow (PFV_{111} + PFV_{211}) \leqslant 1 \Rightarrow (p1c19 + p2c19) \leqslant 1$
Since there is no + constraint in Minion, we can avoid that in this sample by making one of (p1c19, p2c19)=0, and by that we can ensure that one of the *passengers* could take car_1 at $time_1$. So the Capacity Constraint for car_1 at $time_1$ is watched-or(eq(p1c19,0),eq(p2c19,0)). See the Capacity Constraints part from the solution on page 21.

- **One car at a time constraints**: this constraint is in the form
  $\sum_{j=1}^{m} PFV_{pjk} = 1$ so,for example, for $passenger_1$ and $time_1 = 9$ then
  $\sum_{j=1}^{2} PFV_{1j1} = 1 \Rightarrow (PFV_{111} + PFV_{121}) = 1 \Rightarrow (p1c19 + p1c29) = 1$
  Since there is no + constraint in Minion, we can avoid that in this sample by making one of (p1c19, p1c29)=0, and by that we can ensure that $passenger_1$ could take only one car at $time_1$. So this Constraint for $passenger_1$ at $time_1$ is watched-or(eq(p1c19,0),eq(p1c29,0)). See the one car at a time constraints part from the solution on page 22.

- **Traveling constraints:** the form of this type of constraints is in the Equation 3.2.3. Instead of $\Longrightarrow$ there is a constraint in Minion called **reifyimply** in the form reifyimply(constraint, r) where r is a boolean variable [4], and instead of $\wedge$ there is a constraint called **watched-and** in the form watched-and(C1,...,Cn), which is equivalent to C1∧C2∧,...,∧Cn. By this constraint, the Equation 3.2.3 would be
  reifyimply(watched-and($\{eq(P_{pk}, C_{jk}), eq(P_{p(k+1)}, C_{j(k+1)})\}$), PFV_{pjk}).
  See the traveling constraints part from the solution on page 22.

- **Locations constraints:** for this type of constraints, we can modify the form
  $(P_{p1} \neq P_{p2}) \implies PFV_{p11} \vee PFV_{p21} \vee PFV_{p31} \vee ... \vee PFV_{pm1}$.
  As addressed in the Adjacent Constraints, A $\Longrightarrow$ B is an abbreviation for

$\neg A \vee B$, so the form above would be:

$(P_{p1} = P_{p2}) \vee PFV_{p11} \vee PFV_{p21} \vee PFV_{p31} \vee ... \vee PFV_{pm1}$

See the locations constraints part from the solution on page 22 which contain watched-or and eq constraints in Minion.

The Solution[1]:

MINION 3

VARIABLES

#passenger1

DISCRETE p19 {1..3}

DISCRETE p110 {1..3}

DISCRETE p111 {1..3}

#passenger2

DISCRETE p29 {1..3}

DISCRETE p210 {1..3}

DISCRETE p211 {1..3}

#car_1

DISCRETE c19 {1..3}

DISCRETE c110 {1..3}

DISCRETE c111 {1..3}

#car2

DISCRETE c29 {1..3}

DISCRETE c210 {1..3}

DISCRETE c211 {1..3}

#Flag variables

[1]MINION 3 to start of the minion file, # means comments in Minion,**VARIABLES** to start the Variables part in Minion, **CONSTRAINTS** to start Constraints part in Minion, and **EOF** is the end of minion file.

```
BOOL p1c19
BOOL p2c19
BOOL p1c110
BOOL p2c110
BOOL p1c29
BOOL p2c29
BOOL p1c210
BOOL p2c210
**CONSTRAINTS**
#initial constraints
eq(c19,3)
eq(c29,3)
eq(p19,3)
eq(p111,1)
eq(p29,3)
eq(p210,2)
eq(p211,2)
# adjacent constraints
watched-or(diseq(c19,1),eq(c110,1),eq(c110,2))
watched-or(diseq(c110,1),eq(c111,1),eq(c111,2))
watched-or(diseq(c19,3),eq(c110,3),eq(c110,2))
watched-or(diseq(c110,3),eq(c111,3),eq(c111,2))
watched-or(diseq(c29,1),eq(c210,1),eq(c210,2))
watched-or(diseq(c210,1),eq(c211,1),eq(c211,2))
watched-or(diseq(c29,3),eq(c210,3),eq(c210,2))
watched-or(diseq(c210,3),eq(c211,3),eq(c211,2))
#Capacity Constraints
```

```
watched-or(eq(p1c19,0),eq(p2c19,0))
watched-or(eq(p1c110,0),eq(p2c110,0))
watched-or(eq(p1c29,0),eq(p2c29,0))
watched-or(eq(p1c210,0),eq(p2c210,0))
#One car at a time constraints
watched-or(eq(p1c19,0),eq(p1c29,0))
watched-or(eq(p1c110,0),eq(p1c210,0))
watched-or(eq(p2c19,0),eq(p2c29,0))
watched-or(eq(p2c110,0),eq(p2c210,0))
# traveling constraints
reifyimply(watched-and({eq(p19,c19),eq(p110,c110)}),p1c19)
reifyimply(watched-and({eq(p110,c110),eq(p111,c111)}),p1c110)
reifyimply(watched-and({eq(p19,c29),eq(p110,c210)}),p1c29)
reifyimply(watched-and({eq(p110,c210),eq(p111,c211)}),p1c210)
reifyimply(watched-and({eq(p29,c19),eq(p210,c110)}),p2c19)
reifyimply(watched-and({eq(p210,c110),eq(p211,c111)}),p2c110)
reifyimply(watched-and({eq(p29,c29),eq(p210,c210)}),p2c29)
reifyimply(watched-and({eq(p210,c210),eq(p211,c211)}),p2c210)
#Locations Constraints
watched-or({eq(p19,p110),eq(p1c19,1),eq(p1c29,1)})
watched-or({eq(p110,p111),eq(p1c110,1),eq(p1c210,1)})
watched-or({eq(p29,p210),eq(p2c19,1),eq(p2c29,1)})
watched-or({eq(p210,p211),eq(p2c110,1),eq(p2c210,1)})
**EOF**
```

3.4 Solving the Problem with lots of Passenger, Cars, Times, and CarStations by CP

To formulate the DQOC transportation problem with n *Passanger*, m *cars*, t *times*, and c *car stations*, we can just modify the sample in section 3.3 by using a matrix for the variables as follows:

- **Passengers**:

  DISCRETE p[n,t] {1..c}

  where DISCRETE means the type is integer, p is the name of the array of variables of n rows and t columns, and {1..c} is the *domain* of the variables where c is the total number of *car stations*.

- **Cars:**

  DISCRETE cs[m,t] {1..c}

  where cs is the name of the array of variables of m rows and t columns.

- **Flag variables:**

  - $Passengers\ FlagVariables(PFV):$

    BOOL PFV[n,m,t-1]

    where **BOOL** means the type is Boolean, PFV the name of the Three-dimensional array for PFV variables. Since there are n *passengers*, m *cars*, and t *times* and there is no need to the PFV in the last $Time$, so there are $n \times m \times (t-1)$ PFV variables.

  - $Cars'\ Flagvariables(CFV)$: there is no need to $Cars'\ Flagvariables(CFV)$ as addressed in section 3.3.

The constraints are the same as in the sample in Section 3.3 with some changes to work with arrays as follows:

- **Initial constraints:** for every *cars* j, $time_1$ should be known. For every *passengers* p $time_1$ and $time_t$ should be know. So the initial constraints

  eq(cs[j,0],s) for j={0... m-1}, and s = one possible value of the range {1...c}
  eq(p[p,0],s) for p={0... n-1}, and s = one possible value of the range {1...c}
  eq(p[p,i-1],s) for p={0... n-1}, and s = one possible value of the range {1...c}
  and any other known locations for any passenger at the other *times*

- **Adjacent constraints:** suppose d= the number of the adjacent *car stations* for *car station* s, and the adjacent *car stations* $= \{s_1, s_2, s_3, ..., s_d\}$, then the adjacent constraints for *car* j=0...m-1

  watched-or({diseq(cs[j,0],s),eq(cs[j,1],s),eq(cs[j,1],s_1),eq(cs[j,1],s_2),eq(cs[j,1],s_3) ,...,eq(cs[j,1],s_d)})
  watched-or({diseq(cs[j,1],s),eq(cs[j,2],s),eq(cs[j,2],s_1),eq(cs[j,2],s_2),eq(cs[j,2],s_3) ,...,eq(cs[j,2],s_d)})
  watched-or({diseq(cs[j,2],s),eq(cs[j,3],s),eq(cs[j,3],s_1),eq(cs[j,3],s_2),eq(cs[j,3],s_3) ,...,eq(cs[j,3],s_d)})
  ⋮
  watched-or({diseq(cs[j,t-2],s),eq(cs[j,t-1],s),eq(cs[j,t-1],s_1),eq(cs[j,t-1],s_2), eq(cs[j,t-1],s_3),...,eq(cs[j,t-1],s_d)})

- **Capacity constraints:** by modifying the form 3.2.1 to work with arrays, it will be $\sum_{x=1}^{n} PFV[x, j, i] \leqslant r$ for j={0...m-1},i={0...t-1}, so for car_1, at $time_1$, and if the capacity=r=3, then $\sum_{x=1}^{n} PFV[x, 0, 0] \leqslant 3$.

In Minion, there is a constraint called "occurrenceleq(vec, elem, count) ensures that there are *at most* count occurrences of the value elem in the vector vec (elem and count must be constants)" [4]. We use _ instead of a row or a column, if we want to get that row or column. So by **occurrenceleq(bool[_,j,i], 1, r)**,we ensure that the capacity for *car j* at $time_i$=r. So the capacity for car_1 will be:

occurrenceleq(bool[_,0,0], 1, 3)

occurrenceleq(bool[_,0,1], 1, 3)

occurrenceleq(bool[_,0,2], 1, 3)

⋮

occurrenceleq(bool[_,0,t-1], 1, 3)

Moreover, the same for all the cars.

- **One car at a time constraints:** by modifying the form 3.2.2 to work with arrays, it will be $\sum_{j=1}^{m} PFV[i,j,k] = 1$ for i={0...n-1},k={0...t-1}, so for $passenger_1$, this constraint will be $\sum_{j=1}^{m} PFV[0,j,0] = 1$.
  By **occurrenceleq(bool[i,_,k], 1, 1)**,we ensure that *passenger i* will ride only one car at $time_k$. So the one car at a time constraints for $passenger_1$ will be:

  occurrenceleq(bool[0,_,0], 1, 1)

  occurrenceleq(bool[0,_,1], 1, 1)

  occurrenceleq(bool[0,_,2], 1, 1)

  ⋮

  occurrenceleq(bool[0,_,t-1], 1, 1)

  Moreover, the same for all the passengers.

- **Traveling constraints:** by modifying the Equation 3.2.3 and its version in the sample in section 3.3, the form will be:

  reifyimply(watched-and({eq(P[p,k],C[i,k]),eq(P[p,k+1],C[i,k+1])}), PFV[p,i,k]) therefore the traveling constraints for $passenger_1$ at $time_1$ are:

reifyimply(watched-and({eq(p[0,0],c[0,0]),eq(p[0,1],c[0,1])}),PFV[0,0,0])

reifyimply(watched-and({eq(p[0,0],c[1,0]),eq(p[0,1],c[1,1])}),PFV[0,1,0])

$\vdots$

reifyimply(watched-and({eq(p[0,0],c[m-1,0]),eq(p[0,1],c[m-1,1])}),PFV[0,m-1,0])

And the same at all other *times*, and the same for all other *passengers* at all *times*.

- **Locations constraints:** for this type of constraints, by modifying the form sample in section 3.3

  $(P_{p1} = P_{p2}) \vee PFV_{p11} \vee PFV_{p21} \vee PFV_{p31} \vee ... \vee PFV_{pm1}$

  will get:

  watched-or({eq(p[p,0],p[p,1]),eq(bool[p,0,0],1),eq(bool[p,1,0],1),eq(bool[p,2,0],1)
  ,...,,eq(bool[p,m-1,0],1)})

  watched-or({eq(p[p,1],p[p,2]),eq(bool[p,0,1],1),eq(bool[p,1,1],1),eq(bool[p,2,1],1)
  ,...,,eq(bool[p,m-1,1],1)})

  watched-or({eq(p[p,2],p[p,3]),eq(bool[p,0,2],1),eq(bool[p,1,2],1),eq(bool[p,2,2],1)
  ,...,,eq(bool[p,m-1,2],1)})

  $\vdots$

  watched-or({eq(p[p,t-2],p[p,t-1]),eq(bool[p,0,t-2],1),eq(bool[p,1,t-2],1),eq(bool[
  p,2,t-2],1),...,eq(bool[p,m-1,t-2],1)})

  For p={0,1,2,...,n-1}

Chapter 4

Boolean Satisfiability (SAT) Formulation

The Boolean Satisfiability Problem (SAT) is a decision problem where a solver must determine if a boolean expression (also called a formula) is satisfiable. That is, there is an assignment of values that make the expression true. Boolean expressions are expressions with boolean variables (with domain {true, false}), logic operations: AND (conjunction), OR (disjunction), and NOT (negation) and parenthesis. A boolean expression is satisfiable if all its boolean variables are assigned with values (true or false) in a way that yields the boolean expression to be true. In SAT, the variables are with Boolean domains, and the constraints are translated to clauses to represent the CP problem in a very simple language instead of high-level representation by CSP. The simple representation of SAT leads to an efficient implementation for the problem, but it increases the effort to express the problem as SAT instance [3, 12, 17, 2, 1]. Moreover, because constraint programming solvers have a library of constraints that helps the user to address the problems in a very direct way as we will see how to map the CSP constraints to SAT. Therefore Using CP (high-level paradigm) allows a more natural expression of the problem, and then the CP instance will be translated into SAT. In the following sections, the logic operations AND, OR, and NOT are symbolized by $\wedge$, $\vee$, and $\neg$ respectively.

There are three standardizations for the boolean expression: disjunctive normal form (DNF), conjunctive normal form (CNF) and negation normal form (NNF. The SAT solvers use the CNF.

4.1 Conjunctive Normal Form (CNF)

CNF is a boolean formula which consists of a group of boolean expressions separated by a conjunction ($\wedge$) while each boolean expression consists of one or more variables or their negation separated by disjunction ($\vee$). The boolean expression is called **clause** that consists of some literals (the variable or its negation) and disjunction. For example, $(A \vee B) \wedge (\neg B \vee C)$ is a CNF with two clauses each of them with two literals. In other words, boolean expression is in CNF if it is a conjunction of clauses and each clause is a disjunction of literals or their negation.

4.2 Map Problem to CNF

To map a problem to CNF, we have to specify the variables and the constraints in the problem in a CNF format.

4.2.1 The Problem Variables

Suppose the company has n *passengers*, the company can hire up to m *cars* to transfer those n *passengers* between c *car stations*, the company divides the day into t *times*, and there are c *car stations*.

Let us symbolized for *passengers* by P, *cars* by C, *times* by T, and *car stations* by

S

So the list of variables as follows:

- **Passengers variables**: Because the domain of the variable in CNF is {0, 1}, so we have to modify the passengers' variables with a domain {1,...,c} to be with a domain {0, 1}. In order to set that, we make each passengers' variables with domain {1,...,c} of c boolean variable. The new passengers' variables are in the form P_{ijk} where i is the passenger's id, j is the time, and k is the car station number. For example, if n=2, t=3 and c=3, the passengers' variables will be:

  For P_1 there are nine boolean variables which are P_{111}, P_{112}, P_{113}, P_{121}, P_{122}, P_{123}, P_{131}, P_{132}, P_{133}. For example, P_{111} means $passenger_1$ at $time_1$ in $carstation_1$, and P_{112} means $passenger_1$ at $time_1$ in $carstation_2$.

  For P_2 also there are nine boolean variables P_{211}, P_{212}, P_{213}, P_{221}, P_{222}, P_{223}, P_{231}, P_{232}, P_{233}.

  In general, for any n *passengers*, t *times*, and c *car stations*, the variables would be

  P_{111}, P_{112}, P_{113}, ..., P_{1tc}

  P_{211}, P_{212}, P_{213}, ..., P_{2tc}

  $\vdots$

  P_{n11}, P_{n12}, P_{n13}, ..., P_{ntc}.

  So there are $n \times t \times c$ boolean variables for the *passengers.*

- **Cars variables**: To modify the cars' variables from being with a domain {1,...,c} to be with a domain {0, 1}, we do the same as for passengers' variables. The new cars' variables are in the form C_{ijk} where i is the car number, j is the time, and k is the car station number. There are $m \times t \times c$ boolean variables for the *cars.*

- **Flag variables**: The flag variables in the problem as SAT are the same as in CP. There are two types of Flag variables, one for *passengers*, and the other for *cars*.

  - **Passengers flag variables (PFV)**: PFVs are in the form PFV_{ijk} where i is the passenger's id, and j is the car's number, and k is the time. For each of the $n \times t$ passengers variables, there are m variables, such that PFV_{111} means P_1 in C_1 at T_1, PFV_{121} means P_1 in C_2 at T_1,PFV_{nmt} means P_n in C_m at T_t.
    So there are $n \times m \times t$ PFV variables. The **domain** of these $n \times m \times t$ variables is (0,1).
  - **Cars flag variables (CFV)**: if at lease one of the n *passengers* in one of the m *cars* at any *time*, that means that *car* is used for sometime. Therefore, there is one boolean variable per car to show if it used or not.

4.2.2 The Problem Constraints

The problem constraints in SAT are as follows:

- **One location constraints:** In addition to the five types of constraints in section 3.2.2, there is a need to a new constraint to keep each passenger and each car at any time in only one of the car stations. In other words, this constraint is to keep only one of the c boolean variables for passenger i at time j equals true. For example, if there are three car stations, only one of the $passenger_1$ variables at $time_1$ equals true. So the one location constraint for $passenger_1$ at $time_1$ if there are three car stations would be:
  $(\neg P_{111} \vee \neg P_{112}) \wedge (\neg P_{111} \vee \neg P_{113}) \wedge (\neg P_{112} \vee \neg P_{113}) \wedge (P_{111} \vee P_{112} \vee P_{113})$.
  By this constraint, we ensure that $passenger_1$ at $time_1$ will be in only one car station, and we have to repeat this constraint for $passenger_1$ for all the

times and all the passengers at all the times. In general, for c car stations, the one location constraint consists of the disjunction between every possible pair of the negation of the passengers variables at a particular time and one more clause of the disjunction of all the c variable at that time. The same applies for the cars. So there are $n \times t \times (\binom{c}{2} + 1)$ clauses for the passengers and $m \times t \times (\binom{c}{2} + 1)$ clauses for the cars.

- **Initial constraints:** In this type of constraint, the clauses will be only of one literal. The beginning time and the departure time for the passengers must be known. So the initial constraint will be $P_{p1k} \wedge P_{ptk}$ where p=1 to n and k=1 to c. If other *times* are known, they should be set, for example, if the P_1 will stay in $carstation_8$ for the first five *times*, then $P_{118} \wedge P_{128} \wedge P_{138} \wedge P_{148} \wedge P_{158}$. So there are at least $2 \times n$ clauses for initial constraints for the *passengers*. For the cars, there are m clauses because only the beginning time should be set.

- **Adjacent constraints:** As addressed in section 3.2.2 in the adjacent constraints on page 12, since there is no $\implies$ in the CNF, and in logic, A $\implies$ B is the same as ¬A ∨ B. So the adjacent constraint for car_1 if d is the number of the adjacent *car stations* for a *car station* s, and the adjacent *car stations* = $\{s_1, s_2, s_3, ..., s_d\}$ will be:
$(\neg C_{11s} \vee C_{12s} \vee C_{12s_1} \vee C_{12s_2} \vee C_{12s_3} \vee \ldots \vee C_{12s_d}) \wedge (\neg C_{12s} \vee C_{13s} \vee C_{13s_1} \vee C_{13s_2} \vee C_{13s_3} \vee \ldots \vee C_{13s_d}) \wedge (\neg C_{13s} \vee C_{14s} \vee C_{14s_1} \vee C_{14s_2} \vee C_{14s_3} \vee \ldots \vee C_{14s_d}) \wedge \ldots \wedge (\neg C_{1(t-1)s} \vee C_{1ts} \vee C_{1ts_1} \vee C_{1ts_2} \vee C_{1ts_3} \vee \ldots \vee C_{1ts_d})$
And the same for all other cars.

- **Capacity constraints:** for each car j at time k, the capacity constraints consist of the disjunctions between any possible subset of size = capacity +1 of the negation of the PFV_{ijk} for i= 1 to n. For example, if the car capacity is 1 and there are only three passengers, then the capacity constraint for car_1 at time 1 will be:

$(\neg PFV_{111} \vee \neg PFV_{211}) \wedge (\neg PFV_{111} \vee \neg PFV_{311}) \wedge (\neg PFV_{211} \vee \neg PFV_{311})$

But if the capacity is 2, then the capacity constraint will be:

$(\neg PFV_{111} \vee \neg PFV_{211} \vee \neg PFV_{311})$

And the same for all other times and for all other cars at all times. So there are $m \times t \times \binom{n}{capacity+1}$ clauses for the capacity constraints.

- **One car at a time constraints:** for each passenger i at time k, the one car at a time constraints consist of the disjunctions between any possible subset of size = 2 of the negation of the PFV_{ijk} for j= 1 to m. For example, if there are only three cars, then the One Car at a Time constraint for $passenger_1$ at time 1 will be:

  $(\neg PFV_{111} \vee \neg PFV_{121}) \wedge (\neg PFV_{111} \vee \neg PFV_{131}) \wedge (\neg PFV_{121} \vee \neg PFV_{131})$

  And the same for all other times and for all other passengers at all times. So there are $n \times t \times \binom{m}{2}$ clauses for the One car at a time constraints.

- **Traveling constraints:** The form of these constraints is:

  $PFV_{pik} \implies (C_{ik} = P_{pk}) \wedge (C_{i(k+1)} = P_{p(k+1)})$

  Since there is no $\implies$ in the CNF, and in logic, A $\implies$ B is the same as $\neg$A $\vee$ B. So, the form above will be:

  $\neg PFV_{pik} \vee (((P_{pk1} \wedge C_{ik1}) \vee (P_{pk2} \wedge C_{ik2}) \vee (P_{pk3} \wedge C_{ik3}) \vee \ldots \vee (P_{pkc} \wedge C_{ikc})) \wedge ((P_{p(k+1)1} \wedge C_{i(k+1)1}) \vee (P_{p(k+1)2} \wedge C_{i(k+1)2}) \vee (P_{p(k+1)3} \wedge C_{i(k+1)3}) \vee \ldots \vee (P_{p(k+1)c} \wedge C_{i(k+1)c})))$.

  Then we have to modify this form to be in the CNF, and there is a tool to convert any boolean expression to CNF [15].For example, if there are 3 car stations, the form above will be:

  $(\neg PFV_{pik} \vee P_{pk1} \vee P_{pk2} \vee P_{pk3}) \wedge (\neg PFV_{pik} \vee P_{pk1} \vee P_{pk2} \vee C_{ik3}) \wedge (\neg PFV_{pik} \vee$

$P_{pk1} \vee C_{ik2} \vee P_{pk3}) \wedge (\neg PFV_{pik} \vee P_{pk1} \vee C_{ik2} \vee C_{ik3}) \wedge (\neg PFV_{pik} \vee C_{ik1} \vee P_{pk2} \vee P_{pk3}) \wedge (\neg PFV_{pik} \vee C_{ik1} \vee P_{pk2} \vee C_{ik3}) \wedge (\neg PFV_{pik} \vee C_{ik1} \vee C_{ik2} \vee P_{pk3}) \wedge (\neg PFV_{pik} \vee C_{ik1} \vee C_{ik2} \vee C_{ik3}) \wedge (\neg PFV_{pik} \vee P_{p(k+1)1} \vee P_{p(k+1)2} \vee P_{p(k+1)3}) \wedge (\neg PFV_{pik} \vee P_{p(k+1)1} \vee P_{p(k+1)2} \vee C_{i(k+1)3}) \wedge (\neg PFV_{pik} \vee P_{p(k+1)1} \vee C_{i(k+1)2} \vee P_{p(k+1)3}) \wedge (\neg PFV_{pik} \vee P_{p(k+1)1} \vee C_{i(k+1)2} \vee C_{i(k+1)3}) \wedge (\neg PFV_{pik} \vee C_{i(k+1)1} \vee P_{p(k+1)2} \vee P_{p(k+1)3}) \wedge (\neg PFV_{pik} \vee C_{i(k+1)1} \vee P_{p(k+1)2} \vee C_{i(k+1)3}) \wedge (\neg PFV_{pik} \vee C_{i(k+1)1} \vee C_{i(k+1)2} \vee P_{p(k+1)3}) \wedge (\neg PFV_{pik} \vee C_{i(k+1)1} \vee C_{i(k+1)2} \vee C_{i(k+1)3})$.
The same for each PFV.

- **Locations constraints:** By modifying the form of locations constraints in section 3.2.2 on page 14, which is:

$$
\begin{aligned}
(P_{p1} \neq P_{p2}) &\implies PFV_{p11} \vee PFV_{p21} \vee PFV_{p31} \vee ... \vee PFV_{pm1} \\
(P_{p2} \neq P_{p3}) &\implies PFV_{p12} \vee PFV_{p22} \vee PFV_{p32} \vee ... \vee PFV_{pm2} \\
(P_{p3} \neq P_{p4}) &\implies PFV_{p13} \vee PFV_{p23} \vee PFV_{p33} \vee ... \vee PFV_{pm3} \\
&\vdots \\
(P_{p(t-1)} \neq P_{pt}) &\implies PFV_{p1(t-1)} \vee PFV_{p2(t-1)} \vee PFV_{p3(t-1)} \vee ... \vee PFV_{pm(t-1)}
\end{aligned}
$$

and its modification on page 19, which is (for $passenger_1$ at $time_1$):
$(P_{p1} = P_{p2}) \vee PFV_{p11} \vee PFV_{p21} \vee PFV_{p31} \vee ... \vee PFV_{pm1}$
the location constraint form in SAT at time k will be:
$((P_{pk1} \wedge P_{p(k+1)1}) \vee (P_{pk2} \wedge P_{p(k+1)2}) \vee (P_{pk3} \wedge P_{p(k+1)3}) \vee ... \vee (P_{pkc} \wedge P_{p(k+1)c})) \vee PFV_{p1k} \vee PFV_{p2k} \vee PFV_{p3k} \vee ... \vee PFV_{pmk}$.
And this is the same for all other times for this passenger and the same for all other passengers at all the times. Then we have to modify this form to be in the CNF by using the same tool [15]. For example, if there are 3 car stations and 2 cars, the form above at time 1 will be:
$(P_{p11} \vee P_{p12} \vee P_{p13} \vee PFV_{p11} \vee PFV_{p21}) \wedge (P_{p11} \vee P_{p12} \vee P_{p23} \vee PFV_{p11} \vee PFV_{p21}) \wedge$

$(P_{p11} \vee P_{p22} \vee P_{p13} \vee PFV_{p11} \vee PFV_{p21}) \wedge (P_{p11} \vee P_{p22} \vee P_{p23} \vee PFV_{p11} \vee PFV_{p21}) \wedge$
$(P_{p21} \vee P_{p12} \vee P_{p13} \vee PFV_{p11} \vee PFV_{p21}) \wedge (P_{p21} \vee P_{p12} \vee P_{p23} \vee PFV_{p11} \vee PFV_{p21}) \wedge$
$(P_{p21} \vee P_{p22} \vee P_{p13} \vee PFV_{p11} \vee PFV_{p21}) \wedge (P_{p21} \vee P_{p22} \vee P_{p23} \vee PFV_{p11} \vee PFV_{p21})$.

4.3 Solving Sample Problem by CNF

The SAT solver used to solve the DQOC transportation problem is called UBCSAT [14]. The solver can find one solution, many solutions, or prove that there are no solution. The input file format for UBCSAT solver is called DIMACS format. For example, if there are five variables and three clauses as follows:

Variables: v_1, v_2, v_3, v_4, v_5.

Clauses: $(v_1 \vee \neg v_5 \vee v_4) \wedge (\neg v_1 \vee v_5 \vee v_3 \vee v_4) \wedge (\neg v_3 \vee \neg v_4)$.

The DIMACS format for this example will be:

`c comment`

`p cnf 5 3` `c 5 variables 3 clauses`

`1 -5 4 0` `c` $v_1 \vee \neg v_5 \vee v_4$

`-1 5 3 4 0` `c` $\neg v_1 \vee v_5 \vee v_3 \vee v_4$

`-3 -4 0` `c` $\neg v_3 \vee v_4$

`c the comments can be anywhere in the file`

The line "p cnf *numbervar numberclauses*" means the format is CNF; numbervar is "the number of variables in the file; numberclauses is the number of clauses in the file. The clause is a sequence of distinct non-null numbers between -numbervar and numbervar ending with 0 on the same line" [8]. The clause will ignored if it contains the "opposite literals i and -i simultaneously. Positive numbers denote the corresponding variables. Negative numbers denote the negations of the corresponding variables" [8]. The lines that begin with the character c indicating comments.

4.3.1 Map Sample Problem to CNF

To formulate a sample problem, let us use the same sample in section 3.3, which suppose a company that has two *passengers* $\{P_1, P2\}$, the company can hire up to two *cars* to take these two *passengers*, $\{C_1, C_2\}$, and it divides the day into three *times*. Because the variables are only numbers, and because we should make it easy to track those variables and the expected solution for this sample, we need to set up the variables in a way that can be easy to remember:

- **Passengers**:

  Since the day is divided into three times and there are three *car stations*, there are nine variables for each passenger. Passenger1's variables are P_{111}, P_{112}, P_{113}, P_{121}, P_{122}, P_{123}, P_{131}, P_{132}, P_{133}, which in DIMACS format are variables 1, 2, 3, ..., 9, respectively. Passenger2's variables are P_{211}, P_{212}, P_{213}, P_{221}, P_{222}, P_{223}, P_{231}, P_{232}, P_{233}, which in DIMACS format are variables 10, 11, 12, ..., 18, respectively.

- **Cars:**

  As for the passengers, there are nine variables for each car. Car1's variables are C_{111}, C_{112}, C_{113}, C_{121}, C_{122}, C_{123}, C_{131}, C_{132}, C_{133}, which in DIMACS format are variables 19, 20, 21, ..., 27, respectively. Car2's variables are C_{211}, C_{212}, C_{213}, C_{221}, C_{222}, C_{223}, C_{231}, C_{232}, C_{233}, which in DIMACS format are variables 32, 33, 34, ..., 40, respectively.

- **Flag variables:** The PFVs are the same as in the sample in section 3.3, which are PFV_{111}, PFV_{112}, PFV_{211}, PFV_{212} for car_1, and PFV_{121}, PFV_{122}, PFV_{221}, PFV_{222} for car_2. And, in DIMACS format these are variables 28, 29, 30,31 for car_1 and 41, 42, 43, 44 for car_2.

Next, we have to establish the **constraints** as in section 4.2.2. The constraints in this sample are as follows:

- **One location constraints:** As addressed in section 4.2.2, the form of this constraint for $passenger_1$ (for example) in this sample is:

  $(\neg P_{111} \vee \neg P_{112}) \wedge (\neg P_{111} \vee \neg P_{113}) \wedge (\neg P_{112} \vee \neg P_{113}) \wedge (P_{111} \vee P_{112} \vee P_{113})$

  Where the corresponding numbers in DIMACS format for $P_{111}, P_{112}, P_{113}$ are 1, 2, and 3 respectively. So the **One location constraints** for $passenger_1$ at $time_1$ will be:

  -1 -2 0

  -1 -3 0

  -2 -3 0

  1 2 3 0

  and the same for $passenger_1$ for the other times and for all other passengers at all the times. The same applies to the cars as for the passengers. See the **One location constraints** part in the solution for this sample on page 40. The number of the clauses in this constraint as addressed in section 4.2.2, and it is $n \times t \times (\binom{c}{2} + 1)$+m$\times t \times (\binom{c}{2} + 1)$= $2 \times 3 \times (\binom{3}{2} + 1) + 2 \times 3 \times (\binom{3}{2} + 1)$=48.

- **Initial constraints:** The initial values for this sample are the same for the sample in section 3.3, which are illustrated in Table 4.1.

carNumber	at 9	at 10	at 11
1	3	-	-
2	3	-	-
passengers			
1	3	-	1
2	3	2	2

Table 4.1: The initial values for *passengers* and *cars*

See the initial constraint part from the solution on page 40.

- **Adjacent constraints:** As in the adjacent constraints in section 3.3, this type of constraint in this sample also depends in how the map works in Figure 3.1. So, the adjacent constraint for C_1 at $time_1$, if it is in *car station 1* (for example) will be in the form:
  $(\neg C_{111} \vee C_{121} \vee C_{122}) \wedge (\neg C_{121} \vee C_{131} \vee C_{132})$, which is in DIMACS format
  -19 22 23 0
  -22 25 26 0
  And, the same for car_1 if it is in *car station 2* and the same for car_2. See the adjacent constraints part from the solution on page 40.

- **Capacity constraints:** Since the car capacity in the sample is 1, then the capacity constraint for car_1 will be: $(\neg PFV_{111} \vee \neg PFV_{211}) \wedge (\neg PFV_{112} \vee \neg PFV_{212})$, which is in DIMACS format:
  -28 -30 0
  -29 -31 0
  See the Capacity Constraints part from the solution on page 43.

- **One car at a time constraints:** Since there are only two cars in this sample, then the One Car at a Time constraint for $passenger_1$ will be: $(\neg PFV_{111} \vee \neg PFV_{121}) \wedge (\neg PFV_{112} \vee \neg PFV_{122})$, which is in DIMACS format:
  -28 -41 0
  -29 -42 0
  See the one car at a time constraints part from the solution on page 43.

- **Traveling constraints:** The form of these constraints is:
  $PFV_{pik} \implies (C_{ik} = P_{pk}) \wedge (C_{i(k+1)} = P_{p(k+1)})$
  Since there is no $\implies$ in the CNF, and in logic, A $\implies$ B is the same as $\neg$A $\vee$ B. So, the form above will be:
  $\neg PFV_{pik} \vee (((P_{pk1} \wedge C_{ik1}) \vee (P_{pk2} \wedge C_{ik2}) \vee (P_{pk3} \wedge C_{ik3})) \wedge ((P_{p(k+1)1} \wedge C_{i(k+1)1}) \vee$

$(P_{p(k+1)2} \wedge C_{i(k+1)2}) \vee (P_{p(k+1)3} \wedge C_{i(k+1)3})))$. For example, PFV_{111}, which means P_1 rides C_1 at $time_1$ will be true in this form $\neg PFV_{111} \vee (((P_{111} \wedge C_{111}) \vee (P_{112} \wedge C_{112}) \vee (P_{113} \wedge C_{113})) \wedge ((P_{121} \wedge C_{121}) \vee (P_{122} \wedge C_{122}) \vee (P_{123} \wedge C_{123})))$. Then, after converting this form to CNF by using the tool in [15], it will be: $(\neg PFV_{111} \vee P_{111} \vee P_{112} \vee P_{113}) \wedge (\neg PFV_{111} \vee P_{111} \vee P_{112} \vee C_{113}) \wedge (\neg PFV_{111} \vee P_{111} \vee C_{112} \vee P_{113}) \wedge (\neg PFV_{111} \vee P_{111} \vee C_{112} \vee C_{113}) \wedge (\neg PFV_{111} \vee C_{111} \vee P_{112} \vee P_{113}) \wedge (\neg PFV_{111} \vee C_{111} \vee P_{112} \vee C_{113}) \wedge (\neg PFV_{111} \vee C_{111} \vee C_{112} \vee P_{113}) \wedge (\neg PFV_{111} \vee C_{111} \vee C_{112} \vee C_{113}) \wedge (\neg PFV_{111} \vee P_{121} \vee P_{122} \vee P_{123}) \wedge (\neg PFV_{111} \vee P_{121} \vee P_{122} \vee C_{123}) \wedge (\neg PFV_{111} \vee P_{121} \vee C_{122} \vee P_{123}) \wedge (\neg PFV_{111} \vee P_{121} \vee C_{122} \vee C_{123}) \wedge (\neg PFV_{111} \vee C_{121} \vee P_{122} \vee P_{123}) \wedge (\neg PFV_{111} \vee C_{121} \vee P_{122} \vee C_{123}) \wedge (\neg PFV_{111} \vee C_{121} \vee C_{122} \vee P_{123}) \wedge (\neg PFV_{111} \vee C_{121} \vee C_{122} \vee C_{123})$, which in DIMACS format is:

-28 1 2 3 0

-28 1 2 21 0

-28 1 20 3 0

-28 1 20 21 0

-28 19 2 3 0

-28 19 2 21 0

-28 19 20 3 0

-28 19 20 21 0

-28 4 5 6 0

-28 4 5 24 0

-28 4 23 6 0

-28 4 23 24 0

-28 22 5 6 0

-28 22 5 24 0

-28 22 23 6 0

-28 22 23 24 0

See the traveling constraints part from the solution on page 43.

- **Locations constraints:** By modification the form of locations constraints on page 19, which is (for $passenger_1$ at $time_1$):

  $(P_{p1} = P_{p2}) \vee PFV_{p11} \vee PFV_{p21} \vee PFV_{p31} \vee ... \vee PFV_{pm1}$

  the location constraint form for this sample in SAT at time k will be:

  $((P_{pk1} \wedge P_{p(k+1)1}) \vee (P_{pk2} \wedge P_{p(k+1)2}) \vee (P_{pk3} \wedge P_{p(k+1)3})) \vee PFV_{p1k} \vee PFV_{p2k}$.

  For example, this form for P_1 at $time_1$ will be:

  $((P_{111} \wedge P_{121}) \vee (P_{112} \wedge P_{122}) \vee (P_{113} \wedge P_{123})) \vee PFV_{111} \vee PFV_{121}$. Then, after converting this form to CNF by the tool in [15], it will be:

  $(P_{111} \vee P_{112} \vee P_{113} \vee PFV_{111} \vee PFV_{121}) \wedge$

  $(P_{111} \vee P_{112} \vee P_{123} \vee PFV_{111} \vee PFV_{121}) \wedge$

  $(P_{111} \vee P_{122} \vee P_{113} \vee PFV_{111} \vee PFV_{121}) \wedge$

  $(P_{111} \vee P_{122} \vee P_{123} \vee PFV_{111} \vee PFV_{121}) \wedge$

  $(P_{121} \vee P_{112} \vee P_{113} \vee PFV_{111} \vee PFV_{121}) \wedge$

  $(P_{121} \vee P_{112} \vee P_{123} \vee PFV_{111} \vee PFV_{121}) \wedge$

  $(P_{121} \vee P_{122} \vee P_{113} \vee PFV_{111} \vee PFV_{121}) \wedge$

  $(P_{121} \vee P_{122} \vee P_{123} \vee PFV_{111} \vee PFV_{121})$,

  which in DIMACS format is:

  1 2 3 28 41 0

  1 2 6 28 41 0

  1 5 3 28 41 0

  1 5 6 28 41 0

  4 2 3 28 41 0

  4 2 6 28 41 0

  4 5 3 28 41 0

4 5 6 28 41 0

See the locations constraints part from the solution on page 48.

c The CNF Solution:

p cnf 44 227

c **One location constraints**

c passenger1 time 1

-1 -2 0

-1 -3 0

-2 -3 0

1 2 3 0

c passenger1 at time 2

-4 -5 0

-4 -6 0

-5 -6 0

4 5 6 0

c passenger1 at time 3

-7 -8 0

-7 -9 0

-8 -9 0

7 8 9 0

c passenger2 at time 1

-10 -11 0

-10 -12 0

-11 -12 0

10 11 12 0

c passenger2 at time 2

-13 -14 0

-13 -15 0

-14 -15 0

13 14 15 0

c passenger2 at time 3

-16 -17 0

-16 -18 0

-17 -18 0

16 17 18 0

c car1 at time 1

-19 -20 0

-19 -21 0

-20 -21 0

19 20 21 0

c car1 at time 2

-22 -23 0

-22 -24 0

-23 -24 0

22 23 24 0

c car1 at time 3

-25 -26 0

-25 -27 0

-26 -27 0

25 26 27 0

c car2 at time 1

-32 -33 0

-32 -34 0

-33 -34 0

32 33 34 0

c car2 at time 2

-35 -36 0

-35 -37 0

-36 -37 0

35 36 37 0

c car2 at time 3

-38 -39 0

-38 -40 0

-39 -40 0

38 39 40 0

c **Initial Conditions**

c eq(c19,3)

21 0

c eq(c29,3)

34 0

c eq(p19,3)

3 0

c eq(p111,1)

7 0

c eq(p29,3)

12 0

c eq(p210,2)

14 0

c eq(p211,2)

17 0

c **Adjacent constraints**

c car_1

-19 22 23 0

-22 25 26 0

-21 24 23 0

-24 27 26 0

c car 2

-32 35 36 0

-35 38 39 0

-34 37 36 0

-37 40 39 0

c **Capacity constraints**

-28 -30 0

-29 -31 0

-41 -43 0

-42 -44 0

c **One car at a time constraints**

-28 -41 0

-29 -42 0

-30 -43 0

-31 -44 0

c **Traveling constraints:**

c reifyimply(watched-and(eq(p19,c19),eq(p110,c110)),p1c19) in CP

c $\neg p1c19 \vee (((p_{191} \wedge c_{191}) \vee (p_{192} \wedge c_{192}) \vee (p_{193} \wedge c_{193})) \wedge ((p_{1101} \wedge c_{1101}) \vee (p_{1102} \wedge c_{1102}) || (p_{1103} \wedge c_{1103})))$

-28 1 2 3 0

-28 1 2 21 0

-28 1 20 3 0

-28 1 20 21 0

-28 19 2 3 0

-28 19 2 21 0

-28 19 20 3 0

-28 19 20 21 0

-28 4 5 6 0

-28 4 5 24 0

-28 4 23 6 0

-28 4 23 24 0

-28 22 5 6 0

-28 22 5 24 0

-28 22 23 6 0

-28 22 23 24 0

c reifyimply(watched-and(eq(p110,c110),eq(p111,c111)),p1c110)

-29 4 5 6 0

-29 4 5 24 0

-29 4 23 6 0

-29 4 23 24 0

-29 22 5 6 0

-29 22 5 24 0

-29 22 23 6 0

-29 22 23 24 0

-29 7 8 9 0

-29 7 8 27 0

-29 7 26 9 0

-29 7 26 27 0

-29 25 8 9 0

-29 25 8 27 0

-29 25 26 9 0

-29 25 26 27 0

c reifyimply(watched-and(eq(p29,c19),eq(p210,c110)),p2c19)

-30 10 11 12 0

-30 10 11 21 0

-30 10 20 12 0

-30 10 20 21 0

-30 19 11 12 0

-30 19 11 21 0

-30 19 20 12 0

-30 19 20 21 0

-30 13 14 15 0

-30 13 14 24 0

-30 13 23 15 0

-30 13 23 24 0

-30 22 14 15 0

-30 22 14 24 0

-30 22 23 15 0

-30 22 23 24 0

c reifyimply(watched-and(eq(p210,c110),eq(p211,c111)),p2c110)

-31 13 14 15 0

-31 13 14 24 0

-31 13 23 15 0

-31 13 23 24 0

-31 22 14 15 0

-31 22 14 24 0

-31 22 23 15 0

-31 22 23 24 0

-31 16 17 18 0

-31 16 17 27 0

-31 16 26 18 0

-31 16 26 27 0

-31 25 17 18 0

-31 25 17 27 0

-31 25 26 18 0

-31 25 26 27 0

c reifyimply(watched-and(eq(p19,c29),eq(p110,c210)),p1c29)

-41 1 2 3 0

-41 1 2 34 0

-41 1 33 3 0

-41 1 33 34 0

-41 32 2 3 0

-41 32 2 34 0

-41 32 33 3 0

-41 32 33 34 0

-41 4 5 6 0

-41 4 5 37 0

-41 4 36 6 0

-41 4 36 37 0

-41 35 5 6 0

-41 35 5 37 0

-41 35 36 6 0

-41 35 36 37 0

c reifyimply(watched-and(eq(p110,c210),eq(p111,c211)),p1c210)

-42 4 5 6 0

-42 4 5 37 0

-42 4 36 6 0

-42 4 36 37 0

-42 35 5 6 0

-42 35 5 37 0

-42 35 36 6 0

-42 35 36 37 0

-42 7 8 9 0

-42 7 8 40 0

-42 7 39 9 0

-42 7 39 40 0

-42 38 8 9 0

-42 38 8 40 0

-42 38 39 9 0

-42 38 39 40 0

c reifyimply(watched-and(eq(p29,c29),eq(p210,c210)),p2c29)

-43 10 11 12 0

-43 10 11 34 0

-43 10 33 12 0

-43 10 33 34 0

-43 32 11 12 0

-43 32 11 34 0

-43 32 33 12 0

-43 32 33 34 0

-43 13 14 15 0

-43 13 14 37 0

-43 13 36 15 0

-43 13 36 37 0

-43 35 14 15 0

-43 35 14 37 0

-43 35 36 15 0

-43 35 36 37 0

c reifyimply(watched-and(eq(p210,c210),eq(p211,c211)),p2c210)

-44 13 14 15 0

-44 13 14 37 0

-44 13 36 15 0

-44 13 36 37 0

-44 35 14 15 0

-44 35 14 37 0

-44 35 36 15 0

-44 35 36 37 0

-44 16 17 18 0

-44 16 17 40 0

-44 16 39 18 0

-44 16 39 40 0

-44 38 17 18 0

-44 38 17 40 0

-44 38 39 18 0

-44 38 39 40 0

c **Locations constraints**

c watched-or(eq(p19,p110),eq(p1c19,1),eq(p1c29,1)) c $(p_{191} \wedge p_{1101}) \vee (p_{192} \wedge p_{1102}) \vee (p_{193} \wedge p_{1103}) \vee (p_{1c19}) \vee (p_{1c29})$

1 2 3 28 41 0

1 2 6 28 41 0

1 5 3 28 41 0

1 5 6 28 41 0

4 2 3 28 41 0

4 2 6 28 41 0

4 5 3 28 41 0

4 5 6 28 41 0

c watched-or(eq(p110,p111),eq(p1c110,1),eq(p1c210,1))

4 5 6 29 42 0

4 5 9 29 42 0

4 8 6 29 42 0

4 8 9 29 42 0

7 5 6 29 42 0

7 5 9 29 42 0

7 8 6 29 42 0

7 8 9 29 42 0

c watched-or(eq(p29,p210),eq(p2c19,1),eq(p2c29,1))

10 11 12 30 43 0

10 11 15 30 43 0

10 14 12 30 43 0

10 14 15 30 43 0

13 11 12 30 43 0

13 11 15 30 43 0

13 14 12 30 43 0

13 14 15 30 43 0

c watched-or(eq(p210,p211),eq(p2c110,1),eq(p2c210,1))

13 14 15 31 44 0

13 14 18 31 44 0

13 17 15 31 44 0

13 17 18 31 44 0

16 14 15 31 44 0

16 14 18 31 44 0

16 17 15 31 44 0

16 17 18 31 44 0

c the end of CNF solution for this sample.

The result for this solution by using UBCSAT solver is:

-1 -2 3 -4 5 -6 7 -8 -9 -10

-11 12 -13 14 -15 -16 17 -18 -19 -20

21 -22 23 -24 25 -26 -27 28 29 -30

-31 -32 -33 34 -35 36 -37 -38 39 -40

-41 -42 43 -44

The variables that are true are 3 5 7 12 14 17 21 23 25 28 29 34 36 39 43, which are P_{113}, P_{122} , P_{131}, P_{213}, P_{222}, P_{232}, C_{113}, C_{122}, C_{131}, PFV_{111}, PFV_{112}, C_{213}, C_{222}, C_{232}, PFV_{221}, respectively.

This variables in this result mean:

- P_{113}: $passenger_1$ at $time_1$ in $carstation_3$
- P_{122}: $passenger_1$ at $time_2$ in $carstation_2$
- P_{131}: $passenger_1$ at $time_3$ in $carstation_1$
- P_{213}: $passenger_2$ at $time_1$ in $carstation_3$
- P_{222}: $passenger_2$ at $time_2$ in $carstation_2$

- P_{232}: $passenger_2$ at $time_3$ in $carstation_2$
- C_{113}: car_1 at $time_1$ in $carstation_3$
- C_{122}: car_1 at $time_2$ in $carstation_2$
- C_{131}: car_1 at $time_3$ in $carstation_1$
- PFV_{111}: $passenger_1$ in car_1 at $time_1$
- PFV_{112}: $passenger_1$ in car_1 at $time_2$
- C_{213}: car_2 at $time_1$ in $carstation_3$
- C_{222}: car_2 at $time_2$ in $carstation_2$
- C_{232}: car_2 at $time_3$ in $carstation_2$
- PFV_{221}: $passenger_2$ in car_2 at $time_1$

And that shows the problem is solvable, $passenger_1$ transfer by car_1 from $carstation_3$ to $carstation_1$ through $carstation_2$, and $passenger_2$ transfer by car_2 from $carstation_3$ to $carstation_2$ and stay in $carstation_2$.

Chapter 5

Maximum Satisfiability (MAX-SAT) Formulation

The goal of formulating the problem as Maximum Satisfiability Problem (MAX-SAT) is to minimize the number and the cost of hired cars not just to determine if the problem is solvable with a specific number of cars. So, we have to assign a cost for each hired car. Maximum Satisfiability Problem (MAX-SAT) is an optimization version of the Boolean Satisfiability Problem (SAT). SAT is to determine if a boolean expression is satisfiable, while MAX-SAT is to determine the maximum number of satisfiable clauses in CNF formula [9, 13, 10]. For example, in SAT, the boolean expression $(A \lor B) \land (\neg A \lor \neg B) \land (\neg A \lor B) \land (A \lor \neg B)$ is not satisfiable, but in MAX-SAT, the result will be three, which is the maximum number of satisfiable clauses. Additionally, in the DQOC transportation problem, if the problem is not solvable with a particular number of cars, we can check how many passengers can transfer by that number of cars by applying MAX-SAT. However, the goal of formulating the problem as MAX-SAT is to minimize the number of used cars in the company. There are three versions of the MAX-SAT problem the weighted MAX-SAT, the partial

MAX-SAT problems, and the weighted partial MAX-SAT problem. In the weighted MAX-SAT problem, each clause is assigned a positive weight, and the goal is to maximize the sum of weights of satisfied clauses. While in the partial MAX-SAT problem, some clauses must be satisfied by any solution. The mandatory clauses are always represented by a clause with a large weight. In the weighted partial MAX-SAT problem, there are also some mandatory clauses, and the goal is to maximize the sum of weights of satisfied non-mandatory clauses.

5.1 Define WCNF

WCNF is a boolean expression in the CNF formula with weight assigned to each clause. For example, $(A \vee B) \wedge (\neg B \vee C)$ is a WCNF with two clauses, each of them with two literals.

5.2 Map Problem to WCNF

We map the problem of the DQOC transportation to the weighted partial MAX-SAT because there are many mandatory clauses, and the goal is to minimize the number of the hired cars. To map a problem to WCNF, we have to specify the variables and the constraints in the problem:

- **The Problem Variables:** The passengers variables, the cars variables, and Passengers flag variables (PFV) are the same as in SAT. We only have to Figure out the Cars flag variables (CFV), which are if at lease one of the n *passengers* in one of the m *cars* at any *time*, that means that *car* is used for sometime. Therefore, there is one boolean variable per car to show if it used or not. There are m CFVs in the form CFV_i, where i is the car number. For example, $\neg CFV_5$ shows car_5 is not used in the problem while CFV_1 shows car_1 is used in the problem.

- **The Problem constraints:** In addition to the six types of constraints in SAT in section 4.2.2, there is a need for two new constraints:
  - **Used cars constraints**: This type of constraint is to label the car as used if at least one passenger used that car at any time; and if the car is used that means one of the passengers must ride that car. The used car constraint is in the form:
    $CFV_i \equiv PFV_{1i1} \vee PFV_{2i1} \vee PFV_{3i1} \dots PFV_{ni1} \vee PFV_{1i2} \vee PFV_{2i2} \vee PFV_{3i2} \dots PFV_{ni2} \vee \dots \vee PFV_{nit}$.
    In CNF, the used cars constraint will be:
    $(\neg CFV_i \vee PFV_{1i1} \vee PFV_{2i1} \vee PFV_{3i1} \dots \vee PFV_{ni1} \vee PFV_{1i2} \vee PFV_{2i2} \vee PFV_{3i2} \dots PFV_{ni2} \vee \dots \vee PFV_{nit}) \wedge (\neg PFV_{1i1} \vee CFV_i) \wedge (\neg PFV_{2i1} \vee CFV_i) \wedge (\neg PFV_{3i1} \vee CFV_i) \wedge \dots \wedge (\neg PFV_{ni1} \vee CFV_i) \wedge (\neg PFV_{1i2} \vee CFV_i) \wedge (\neg PFV_{2i2} \vee CFV_i) \wedge (\neg PFV_{3i2} \vee CFV_i) \wedge \dots \wedge (\neg PFV_{ni2} \vee CFV_i) \wedge \dots \wedge (\neg PFV_{nit} \vee CFV_i)$.
  - **Optimization constraints**: To minimize the number of the hired cars, we add non mandatory clauses in the form (cost $\neg CFV_i$), which means if car_i is unused, then the company will get that cost. The weighted partial MAX-SAT tries to maximize the sum of weights of satisfied clauses, so the system will maximize the number of the unused cars.

5.3 Solving Sample Problem by WCNF

The MAX-SAT solver used to solve the DQOC transportation problem is called toysat [11]. The input file format for the toysat solver is in DIMACS format. For example, if there are five variables and five clauses as follows,

Variables: v_1, v_2, v_3, v_4, v_5.

Clauses: $(v_1 \vee \neg v_5 \vee v_4) \wedge (\neg v_1 \vee v_5 \vee v_3 \vee v_4) \wedge (\neg v_3 \vee \neg v_4) \wedge v_3 \wedge v_4$.

then the DIMACS format for this example will be,

```
c comment
p wcnf 5 5 100      c 5 variables 5 clauses and top=100
100 1 -5 4 0      c v1 ∨ ¬v5 ∨ v4 (hard clause)
100 -1 5 3 4 0      c ¬v1 ∨ v5 ∨ v3 ∨ v4(hard clause)
100 -3 -4 0         c ¬v3 ∨ ¬v4(hard clause)
10 4 0              c v3 (soft clause)
5 3 0               c v4 (soft clause)
c the comments can be anywhere in the file
```

The line p wcnf numbervar numberclauses top means the format is WCNF; the first integer in the clause is the weight, which must be greater than or equal to 1. The weight of the mandatory clause (also called hard clause) is "top", while the weight of the non-mandatory clause (also called soft clause) is less than "top".

5.3.1 Map Sample Problem to WCNF

To formulate a sample problem, let us use the same sample in section 4.3 with some modification by adding one car to show how the number of the hired cars is minimized. The problem's variables are as following:

- **Passengers**:

  The passengers' variables are the same as in SAT sample in section 4.3, which are: Passenger1's variables are P_{111}, P_{112}, P_{113}, P_{121}, P_{122}, P_{123}, P_{131}, P_{132}, P_{133}, which in DIMACS format are variables 1, 2, 3, ..., 9, respectively. Passenger2's variables are P_{211}, P_{212}, P_{213}, P_{221}, P_{222}, P_{223}, P_{231}, P_{232}, P_{233}, which in DIMACS format are variables 10, 11, 12, ..., 18, respectively..

- **Cars:**

  The cars' variables are the same as in SAT sample in section 4.3,which are: Car1's variables are C_{111}, C_{112}, C_{113}, C_{121}, C_{122}, C_{123}, C_{131}, C_{132}, C_{133}, which in DIMACS format are variables 19, 20, 21, ..., 27, respectively. Car2's variables are C_{211}, C_{212}, C_{213}, C_{221}, C_{222}, C_{223}, C_{231}, C_{232}, C_{233}, which in DIMACS format are variables 32, 33, 34, ..., 40, respectively. In addition, car_3's variables, which are C_{311}, C_{312}, C_{313}, C_{321}, C_{322}, C_{323}, C_{331}, C_{332}, C_{333}, which in DIMACS format are 45, 46, 47, ..., 53, respectively.

- **Flag variables:**
  - **Passengers flag variables (PFV)**: The PFVs are also the same as in in section 4.3, which are $PFV_{111}, PFV_{112}, PFV_{211}, PFV_{212}$ for car_1, and $PFV_{121}, PFV_{122}, PFV_{221}, PFV_{222}$ for car 2. And, in DIMACS format these are variables 28, 29, 30,31 for car_1 and 41, 42, 43, 44 for car 2. In addition, $PFV_{131}, PFV_{132}, PFV_{231}, PFV_{232}$ for car 3, which in DIMACS format are 54, 55, 56, 57, respectively.
  - **Cars flag variables (CFV)**: The CFVs are the new variables, which are CFV_1, CFV_2, CFV_3. And, in DIMACS format are 58, 59, 60, respectively.

The problem's constraints are the same as in section 4.3, and adding the **Used cars constraint** and the **optimization constraints**:

- **Used cars constraints**: Since there are four PFVs associated with each car, this constraint will be in the form: $(\neg CFV_i \vee PFV_{1i1} \vee PFV_{2i1} \vee PFV_{1i2} \vee PFV_{2i2}) \wedge (\neg PFV_{1i1} \vee CFV_i) \wedge (\neg PFV_{2i1} \vee CFV_i) \wedge (\neg PFV_{1i2} \vee CFV_i) \wedge (\neg PFV_{2i2} \vee CFV_i)$. For example, used cars constraint for car_1 in DIMACS format is:

  100 -58 28 29 30 31 0

  100 -28 58 0

  100 -29 58 0

  100 -30 58 0

100 -31 58 0

See the **Used cars constraints** in the MAX-SAT solution on page 57.

- **Optimization constraints** To minimize the number of the hired cars, we add non-mandatory clauses in the form ($cost \vee \neg CFV_i$), which means if car_i is not used, then the company will get that cost. The weighted partial MAX-SAT tries to maximize the sum of weights of satisfied clauses, so the system will maximize the number of the unused cars.

  In this type of constraint, which is the only one that consists of non-mandatory clauses, the weight is the cost of the car. For example, the cost in this sample is 10. See the **Optimization constraints** in the MAX-SAT solution on page 58.

c The WCNF Solution:

p wcnf 60 328 100

c passenger1 time 1

100 -1 -2 0

100 -1 -3 0

100 -2 -3 0

100 1 2 3 0

⋮

the same as in SAT Sample, with adding 100 to the front of each clause.

⋮

c **Used cars constraints**

c car_1

100 -58 28 29 30 31 0

100 -28 58 0

100 -29 58 0

100 -30 58 0

100 -31 58 0

c car 2

100 -59 41 42 43 44 0

100 -41 59 0

100 -42 59 0

100 -43 59 0

100 -44 59 0

c car 3

100 -60 54 55 56 57 0

100 -54 60 0

100 -55 60 0

100 -56 60 0

100 -57 60 0

c **Optimization constraints**

10 -58 0

10 -59 0

10 -60 0

c the end of WCNF solution for this sample.

The result for this solution by using the maxhs solver is:

-1 -2 3 -4 5 -6 7 -8 -9 -10

-11 12 -13 14 -15 -16 17 -18 -19 -20

21 -22 23 -24 -25 26 -27 -28 -29 30

-31 -32 -33 34 -35 36 -37 38 -39 -40

41 42 -43 -44 -45 -46 47 -48 -49 50

-51 -52 53 -54 -55 -56 -57 58 59 -60

The variables that are true are 3 5 7 12 14 17 21 23 26 30 34 36 38 41 42 47 50 53 58 59, which are:

$P_{113}, P_{122}, P_{131}, P_{213}, P_{222}, P_{232}, C_{113}, C_{122}, C_{132}, PFV_{211}, C_{213}, C_{222}, C_{231}, PFV_{121}, PFV_{122}, C_{313}, C_{323}, C_{333}, CFV_1, CFV_2$, respectively.

The literals 58, 59, and -60 are showing that car_1 and car_2 are used, but car_3 is not used at all. This variables in this result mean:

- P_{113}: $passenger_1$ at $time_1$ in $carstation_3$
- P_{122}: $passenger_1$ at $time_2$ in $carstation_2$
- P_{131}: $passenger_1$ at $time_3$ in $carstation_1$
- P_{213}: $passenger_2$ at $time_1$ in $carstation_3$
- P_{222}: $passenger_2$ at $time_2$ in $carstation_2$
- P_{232}: $passenger_2$ at $time_3$ in $carstation_2$
- C_{113}: car_1 at $time_1$ in $carstation_3$
- C_{122}: car_1 at $time_2$ in $carstation_2$
- C_{132}: car_1 at $time_3$ in $carstation_2$
- PFV_{211}: $passenger_2$ in car_1 at $time_1$
- C_{213}: car_2 at $time_1$ in $carstation_3$
- C_{222}: car_2 at $time_2$ in $carstation_2$
- C_{231}: car_2 at $time_3$ in $carstation_1$
- PFV_{121}: $passenger_1$ in car_2 at $time_1$
- PFV_{122}: $passenger_1$ in car_2 at $time_2$

- C_{313}: car_3 at $time_1$ in $carstation_3$

- C_{323}: car_3 at $time_2$ in $carstation_3$

- C_{333}: car_3 at $time_3$ in $carstation_3$

- CFV_1: car_1 is used

- CFV_2: car_2 is used

And that shows the problem is solvable by only two cars, $passenger_1$ transfer by car_2 from $carstation_3$ to $carstation_1$ through $carstation_2$, and $passenger_2$ transfer by car_1 from $carstation_3$ to $carstation_2$ and stay in $carstation_2$. In addition, this results shows that only car_1 and car_2 are used, and car_3 is not used at all the times and it stays in $carstation_3$ for all the times.

Chapter 6

Experiments

After we model the DQOC transportation problem using CP, SAT, and MAX-SAT, we have use CP, SAT, and MAX-SAT solvers to produce solutions. If the solvers can find solutions, then we have to check to see which one is the most efficient. In particular, we are interested in knowing what is more efficient: multiple satisfiability queries by CP and SAT or a single optimal query by MAX-SAT.

6.1 Problem setup

In order to check the possibility of applying these solvers, we have to set up suitable problems with different sizes and with all types of solutions. Therefore; we run problems that differ in the number of car stations, differ in the number of passengers, and differ in the number of cars as follows:

- The number of passengers ranges from 10 to 100.
- The number of cities ranges from 3 to 14. For each number of cities, we run problems with 10 to 100 passengers by increments of 10.

- CP and SAT solvers perform decidable queries, that is they answer the question of whether we can solve an instance with X cars, so we need multiple queries with different values of X to find an optimal solution. MAX-SAT finds optimal solution directly. Therefore; the number of cars depends on the type of solutions:

  - In CP and SAT, the number of cars ranges from the ceiling of the number of passengers divided by the car capacity to the number of cars that makes the problem solvable. The ceiling of the number of passengers divided by the car capacity is the smallest number of cars that may make the problem solvable. We iteratively increase from min to max until we find a solvable instance. Min numbers of cars=$\lceil NumberOfPassengers \div CarCapacity \rceil$ to max number of cars= $NumberOfPassengers$.

  - In MAX-SAT, we initially used the number of cars as the number of passengers. But the solver cannot solve instances with such a large number of cars. We will see in the Equation 6.1.1 how the number of cars effects in the number of clauses. Instead, we set the number of cars to be the ceiling of the number of passengers divided by the car capacity and then add five, which is always enough to solve the problem. Numbers of cars=$\lceil NumberOfPassengers \div CarCapacity \rceil + 5$.

- The initial constraints in the experiments are as follows:

  - The initial constraints for the passenger was just to indicate where each passenger is living and where is his or her destination. Each passenger starts in a car station and goes to other car station, in other words, there is no passenger going to the same car station where they live. These constraints are chosen at random. Tabel 6.1 and 6.2 shows these constraints for some cities while for the other cities is in the same randomness.

  - In the initial constraints for the cars, we change the distribution of the cars between the car stations. Instead of distributing the cars between the car stations by the user, we make the system distributes cars between the car stations by the time of need. By adding an extra car station to each problem and putting all the cars in that car station in a time before the beginning time. For example, if the beginning time of the travel is 8:30am, all the cars should be at the extra car station at 8 am, they travel to any car stations by 8:30am. In order to make these cars can move to all the car stations and do not come back to the extra car stations to save the time and cars, we make this extra car station adjacent to all the car stations in only one way. Figure 6.1 shows how the extra car station is connected to all car stations.

- We run all the experiments in two topology for the car stations as follows:

  - The first topology was in a star shape, where one car station in the center and all other car stations around it and connected to it. The extra city - the city that contains the cars before the beginning time- is located out side of the star. Figure 6.1 is example of the first topology. Table 6.1 shows the distribution of the passengers in this topology.

No of Cities	The no of passengers									
3 car stations	case 10	20	30	c 40	50	60	c 70	c 80	c 90	c 100
1	5	10	15	20	25	30	35	40	45	50
2	0	0	0	0	0	0	0	0	0	0
3	5	10	15	20	25	30	35	40	45	50
4 car stations										
1	5	10	10	15	25	30	35	40	45	50
2	0	0	0	0	0	0	0	0	0	0
3	5	0	10	15	20	30	30	40	40	50
4	0	10	10	10	5	0	5	0	5	0
5 car stations										
1	4	9	10	15	25	26	35	36	45	45
2	0	0	0	0	0	0	0	0	0	0
3	5	10	10	15	19	30	30	36	40	46
4	0	0	10	10	5	0	5	0	5	
5	1	1	0		1	4	0	8	0	9
6 car stations										
1	3	6	8	13	23	23	31	32	41	40
2	0	0	0	0	0	0	0	0	0	0
3	5	7	9	13	17	27	27	32	36	42
4	0		10	10	5	0	5	0	5	0
5	1	1	0	0	1	4	0	8	0	9
6	1	6	3	4	4	6	7	8	8	9
7 car stations										
1	3	5	8	12	20	23	31	32	37	40
2	0	0	0	0	0	0	0	0	0	0
3	4	7	9	12	16	27	27	32	32	42
4	0	0	10	10	5	0	5	0	5	0
5	1	1	0	0	1	4	0	8	0	9
6	1	6	3	4	4	6	7	8	8	9
7	1	1	0	2	4	0	0	0	8	0

Table 6.1: The initial constraints for the passengers for some car stations in the first topology

No of Cities	The no of passengers									
3 car stations	case 10	c 20	c 30	c 40	c 50	c 60	c 70	c 80	c 90	c 100
1	4	7	10	14	17	20	24	27	30	34
2	3	6	10	13	16	20	23	26	30	33
3	3	7	10	13	17	20	23	27	30	33
4 car stations										
1	3	5	8	15	10	15	18	20	23	25
2	2	5	7	0	10	15	17	20	22	25
3	3	5	8	15	10	15	18	20	23	25
4	2	5	7	10	10	15	17	20	22	25
5 car stations										
1	2	4	6	8	10	12	14	16	18	20
2	2	4	6	8	10	12	14	16	18	20
3	2	4	6	8	10	12	14	16	18	20
4	2	4	6	8	10	12	14	16	18	20
5	2	4	6	8	10	12	14	16	18	20
6 car stations										
1	2	4	5	7	9	10	12	14	15	17
2	1	3	5	6	8	10	11	13	15	16
3	2	4	5	7	9	10	12	14	15	17
4	2	3	5	7	8	10	12	13	15	17
5	2	3	5	7	8	10	12	13	15	17
6	1	3	5	6	8	10	11	13	15	16
7 car stations										
1	2	3	5	6	8	9	11	12	14	15
2	1	3	5	6	8	9	11	12	14	15
3	2	3	5	6	8	9	11	12	14	15
4	2	3	4	6	7	9	10	12	13	15
5	1	3	4	6	7	9	10	12	13	15
6	1	3	4	6	7	9	10	12	13	15
7	1	2	3	4	5	6	7	8	9	10

Table 6.2: The initial constraints for the passengers for some car stations in the second topology

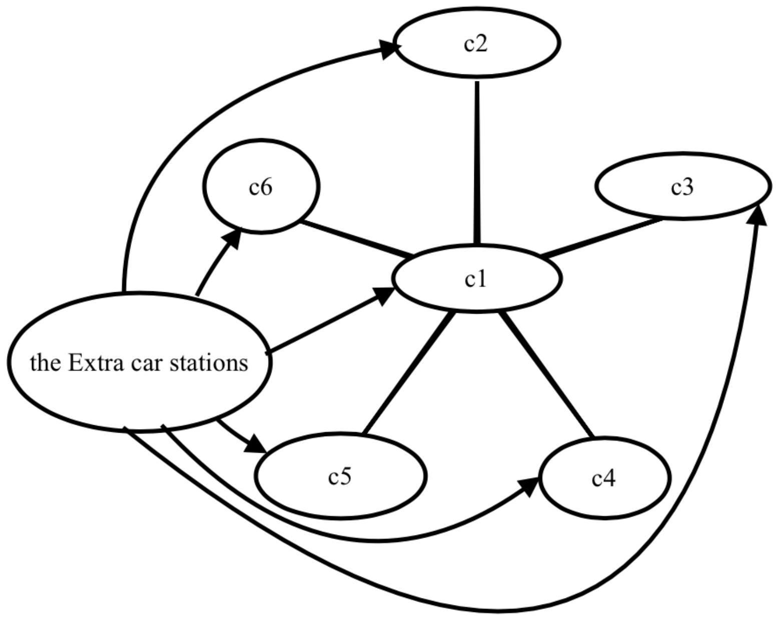


Figure 6.1: Example for the first topology of five car stations

The Figure 6.1 shows the map of the car stations and their destributions in case of six car stations, and how the extra car station connected to all other car stations.

- The second topology was, in addition to first topology, we add a road between each two neighbor car stations. Figure 6.2 is example of the second topology. Table 6.2 shows the distribution of the passengers in this topology.

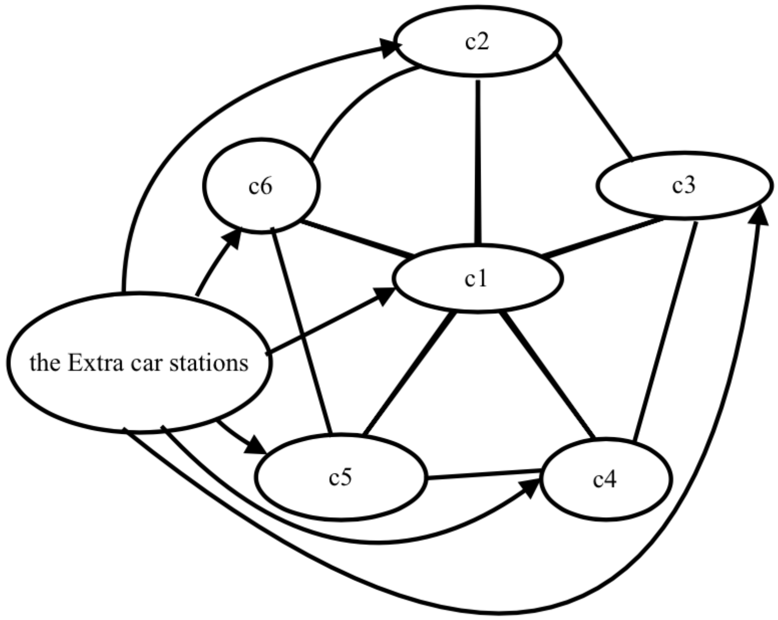


Figure 6.2: Example of the second topology of five car stations

The Figure 6.2 shows the map of the car stations and their destributions in case of six car stations, and how the extra car station connected to all other car stations.

- All the experiments run in one way and in three time steps, in other words, we run the problem in only when the passengers are going to work because it the same when we want to run it in the other way just by change the initial constraints for the passengers. So, we solve instances for times 8:00am-9:30am. All the workers are at home at 8:30am, and all of them are at work at 9:30am.

- The car capacity in all the problem that we run was three passengers. This increases the total number of the clauses in the problems in SAT and MAX-SAT over a two passengers limit because the number of the clauses in the capacity constraints depends on the car capacity. See the Equation 6.1.1, which shows the number of clauses in SAT.

- The number of problems was one hundred and twenty in each type, so the total number of problems was three hundred and sixty. As Table 6.3 shows, the size of the problems in CP ranges from 10,498 bytes (12 KB on disk, different than on RAM which would be KiB) to 785,310 bytes (791 KB on disk).The number of constraints in CP ranges from 177 constraints to 8,885 constraints. The size of the problem in SAT ranges from 199,968 bytes (201 KB on disk) to 27,176,741,630 bytes (27.18 GB on disk, different than on RAM which would be GiB). The number of variable and clauses ranges from 210 variables and 3,452 clauses to12,948 variables and 521,699,344 clauses. Equation 6.1.1 gives the number of clauses for SAT and Equation 6.1.2 gives the number of variables for SAT. The size of the problems in MAX-SAT ranges from 223,552 bytes (225 kB on disk) to 34,375,119,725 bytes (34.38 GB on disk). The number of variable and clauses ranges from 369 variables and 8,040 clauses to13,716 variables and 564,918,484 clauses. The number of clauses for MAX-SAT is calculated by adding $2 \times cars \times (1 + passengers)$ (used cars constraints plus optimization constraints) to the Equation 6.1.1 and taking into consideration the number of cars here is different. The size of the files in MAX-SAT is larger than the size of files in SAT because the increasing in the number of the clauses and the weight of each clause.The number of variables is calculated by adding the number of cars to the equations 6.1.2.

		size	constraints/ clauses	passengers	cars	car stations
CP	small	12 KB	177	10	4	3
	large	791 KB	8,885	100	36	14
SAT	small	201 KB	3,452	10	4	3
	large	27 GB	521,699,344	100	36	14
MAXSAT	small	225 KB	8,040	10	9	3
	large	34 GB	564,918,484	100	39	14

Table 6.3: Size of Experiments

$$\begin{aligned}
\text{No Of Clauses} = {} & \text{one locations for constraints for Passengers} \\
& + \text{one locations for constraints for cars} \\
& + \text{initial constraints+capacity constraints} \\
& + \text{adjacent constraints} \\
& + \text{one car constraints} + \text{traveling constrains}
\end{aligned} \tag{6.1.1}$$

one locations for constraints for Passengers $= (times \times \binom{carStations}{2} + times) \times passengers$

one locations for constraints for cars $= (times \times \binom{carStations}{2} + times) \times cars$

$initialconstraints = 2 \times passengers + cars$

$LocationsConstraints = 2^{carStations} \times 2 \times passengers$

$OneCarConstraints = \binom{cars}{2} \times (times - 1) \times passengers$

$adjacentConstraints = (carStations - 1) \times 2 \times cars + cars$

$CapacityConstraints = \binom{passengers}{capacity+1} \times 2 \times cars$

$$TravelingConstrains = 2^{carStations} \times 2 \times 2 \times passengers \times cars$$

$$\begin{aligned} NoOfVariables = & \; passengers \times carStations \times times \\ & + cars \times carStations \times times \\ & + passengers \times cars \times (times - 1) \\ & + cars + cars) \end{aligned} \tag{6.1.2}$$

6.2 Results

After running the problems in the three different solvers CP, SAT, and MAX-SAT solvers, the results were as following:

- In CP problems, one hundred and four problems are solved in only one second, then some problem are solved in less than eleven seconds and thirteen problems are not solved during the three hundred seconds in the first topology. Figure 6.3 shows this. In the second topology, all the instance are solved during the three hundred seconds. In CP, the increase in the number of car stations does not affect the number of solved instances as shown in Table 6.4 and Table 6.5.
- In SAT, seventy-one problems are solved and the time to solve these problems ranges from one to eighty-eight seconds in the first topology. In the second topology, the number of the solved problem is the same as in the first topology, but the the time required to solve the instance is less that that in the first topology. The other fifty problems are not solved in three hundred seconds. From the experiments, the SAT problem with eighty passengers are not solvable with a number of car stations ranging from three to nine; in ten car stations, the number of passengers starts decreasing. In fourteen car stations,

	Car Stations												Total solved	Total time
Solver	3	4	5	6	7	8	9	10	11	12	13	14		
CP	7	9	6	8	7	10	10	10	10	10	10	10	107	29.36
SAT	7	7	7	7	7	7	7	6	6	5	3	2	71	1208.04
MAX-SAT	1	1	1	1	1	1	1	1	1	1	0	0	10	268.75

Table 6.4: The number of solved problems in five minutes in the first topology

only the problems with twenty passengers or less are solvable. In SAT, the increase in the number of car stations effects in the number of solved instances as shown in Table 6.4 and Table 6.5 because it increases the number of the clauses in exponentially functions as Equation 6.1.1 shows.

- In MAX-SAT in both topology, only ten problems are solved, which are consists of ten passengers and less than thirteen car stations. As Figure 6.3 shows, the solvable problems are solved in a time ranging from one to one hundred and sixteen seconds. In MAX-SAT, the increase in the number of car stations effects in the number of solved instances as shown in Table 6.4 and Table 6.5 for the same reason in SAT.

The main points of the results above is that multiple satisfiability queries by CP and SAT is more efficient than a single optimal query by MAX-SAT because most of the problems in MAX-SAT are not solvable, and the solve time for the problems by CP and SAT is smaller than the solve time for the same problems by MAX-SAT.

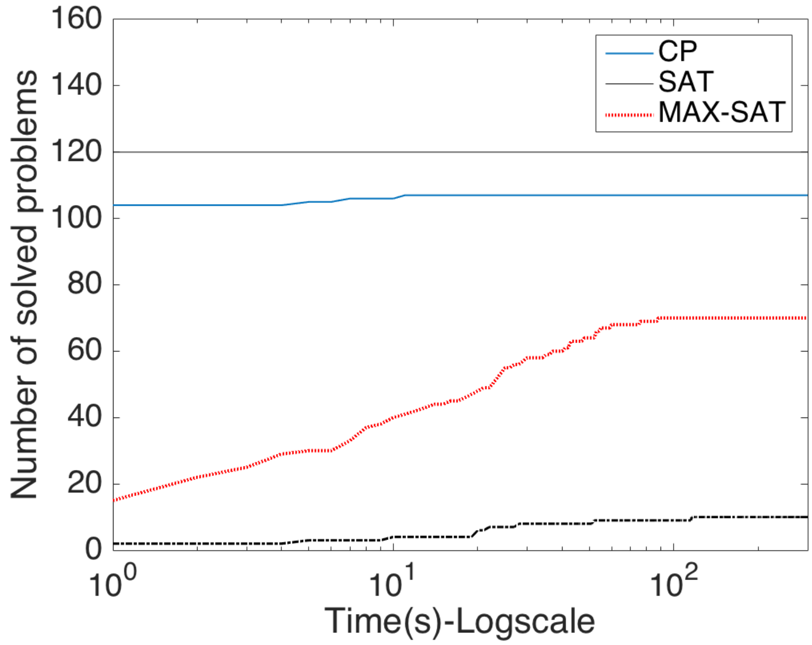


Figure 6.3: The number of solved problems in five minutes

This Figure is to show how many CP, SAT, MAX-SAT problems solved during five minutes.

	Car Stations												Total solved	Total time
Solver	3	4	5	6	7	8	9	10	11	12	13	14		
CP	10	10	10	10	10	10	10	10	10	10	10	10	120	454
SAT	7	7	7	7	7	7	7	6	6	5	3	2	71	182
MAX-SAT	1	1	1	1	1	1	1	1	1	1	0	0	10	296

Table 6.5: The number of solved problems in five minutes in the second topology

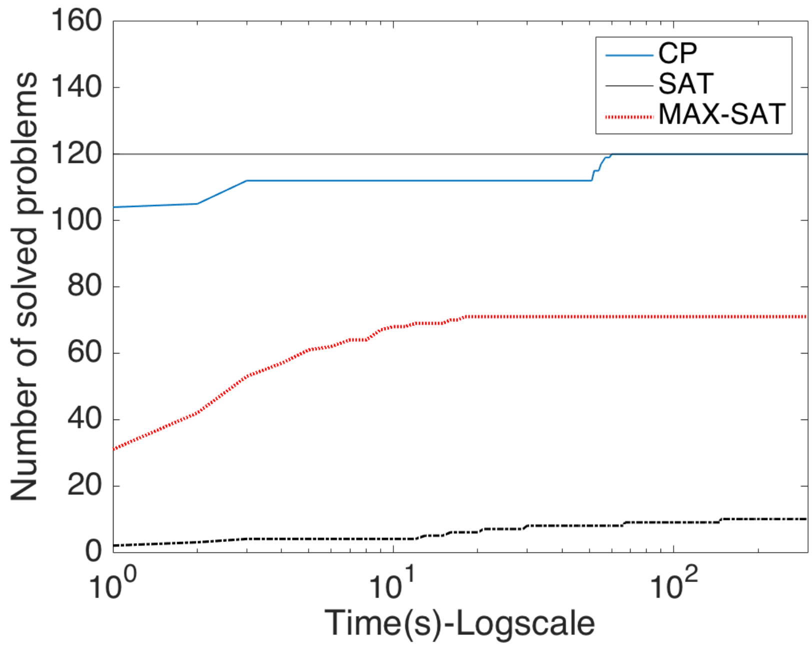


Figure 6.4: The number of solved problems in five minutes

This Figure is to show how many CP, SAT, MAX-SAT problems solved during five minutes.

Chapter 7

Conclusion

In this thesis, we have modeled the DQOC transportation problem using CP, SAT, and MAX-SAT. The main goal of modeling the problem is to find optimal solutions. The secondary goal is to check to see which solver and modeling language is the most efficient.

We showed that finding optimal solutions for the DQOC transportation problem using multiple queries with CP and SAT solvers is more efficient than finding it by a single optimal query using aMAX-SAT solver.

7.1 Future directions

There are still many other approaches that we could use to find an optimal solution for the DQOC transportation problem, such quantified Boolean formulas (QBF) [6]. Additionally, focusing on the second goal (finding solutions efficiently), we could perform further tests using different solvers built by other researchers. Finally, we could investigate alternate formulations of the problem that may represent the problem more efficiently or be easier for solvers to solve.

Bibliography

[1] Fahiem Bacchus. Csc2512 advanced propositional reasoning. Class Notes, winter 2015.

[2] Fahiem Bacchus. Csc2512 algorithms for solving propositional theories. Class note, Winter 2015.

[3] Lucas Bordeaux, Youssef Hamadi, and Lintao Zhang. Propositional satisfiability and constraint programming: A comparative survey. *ACM Computing Surveys (CSUR)*, 38(4):12, 2006.

[4] N. Moore P. Nightingale K. Petrie A. Rendl C. Jefferson, L. Kotthoff. *The Minion Manual*, minion version 1.7 edition, July 1 2014.

[5] Giuseppe F. Italiano Fabrizio Grandoni. Algorithms and constraint programming. In *Principles and Practice of Constraint Programming-CP 2006*, pages 2–14. Springer Berlin Heidelberg, 2006.

[6] Enrico Giunchiglia, Massimo Narizzano, and Armando Tacchella. Qube: A system for deciding quantified boolean formulas satisfiability. In *Automated Reasoning*, pages 364–369. Springer, 2001.

[7] The St Andrews Constraints Group. Minion, http://constraintmodelling.org/minion/.

[8] SAT Competition 2009: Benchmark Submission Guidelines. http://www.satcompetition.org/2009/format-benchmarks2009.html, 2009.

[9] M. Chavira A. Choi K. Pipatsrisawat, A. Palyan and Adnan Darwiche. Solving weighted max-sat problems in a reduced search space: A performance analysis. *Journal on Satisfiability, Boolean Modeling and Computation*, 4:191–217, 2008.

[10] Felip Manyà María Luisa Bonet, Jordi Levy. Resolution for max-sat. *Artificial Intelligence*, 171:606–618, 2007/6/30.

[11] Masahiro Sakai. toysat, https://github.com/msakai/toysolver, 2014.

[12] Peter J Stuckey. Lazy clause generation: Combining the power of sat and cp (and mip?) solving. *Lecture Notes in Computer Science*, 6140:5, 2010.

[13] Phil Sung. Maximum satisfiability. March 17 2006.

[14] Dave A. D. Tompkins and Holger H. Hoos. UBCSAT: An implementation and experimentation environment for SLS algorithms for SAT and MAX-SAT. In Holger Hoos and David Mitchell, editors, *Revised Selected Papers from the Seventh International Conference on Theory and Applications of Satisfiability Testing (SAT 2004)*, volume 3542 of *Lecture Notes in Computer Science*, pages 306–320. Springer Berlin / Heidelberg, 2005.

[15] Zvonimir Rakamaric Tyler Sorensen, Ganesh Gopalakrishnan. a python boolean algebra library.

[16] USAID/PHCPI. Qhi qar, http://phciraq.org/content/dhi-qar-province-primary-health-care-phc-centers, undated. [Online; accessed 16 JAN, 2016; by permission].

[17] Andra s Z. Salamon. *Transformations of representation in constraint satisfaction, http://ora.ox.ac.uk/objects/uuid:5d641fff-4d95-43b2-9ff8-73395d782ad8*. PhD thesis, University of Oxford, 2013.