An AIMMS model for Inventory Balancing and Vehicle Routing in Bike Sharing Systems

Course project for Artificial Intelligence, modullo 2

Chun Tian

Scuola di Scienze, Università di Bologna
chun.tian@studio.unibo.it

Abstract. The purpose of this course project is to learn constraint programming (CP) and then use the CP technology provided by AIMMS to resolve the problem of Inventory Balancing and Vehicle Routing in Bike Sharing Systems. The core CP model is based on previous work [1] with some minor modifications. Then we've done some experiments on the model, using either random data and real data from the city Bologna of Italy.

1 Introduction

Bike sharing systems have been installed in many major cities around the world. In these systems, users can pickup and return bikes at designated bike sharing stations with a finite number of docks. Unfortunately, user behavior results in spatial imbalance of the bike inventory over time. The system equilibrium is often characterized by unacceptably low availability of bikes or open docks, for pickups or returns respectively. Therefore, operators deploy a fleet of trucks to rebalance the bike inventory.

According to [2], this problem consists of two main components. First, determining the desired inventory level at each bike station, which is typically done by an analysis of historic user data. Second, designing truck routes that will perform the necessary pickups and deliveries in order to reach the target inventory levels. In this project, we will assume for simplicity that we are given minimum and maximum target inventory thresholds for each bike station, and further more, when generating random data, we have used the same target inventory thresholds for every station. And the problem is then to find truck routes that will pickup and deliver the bikes at the visited stations such that the inventory level for each station is between the minimum and maximum threshold, with minimum total distance.

2 Constraint Programming in AIMMS

AIMMS is an algebraic modeling language used to provide access to integer linear programming (ILP), quadratic programming (QP), and nonlinear programming

(NLP) technologies. Constraint Programming (CP) was a late-introduced new extension of AIMMS in 2013. To support CP modeling and solving, AIMMS added a group of new Global constraints (i.e. `cp::AllDifferent`), while the old non-CP language syntax can still be used. And an important benefit of AIMMS is that it can not only be used to quickly develop a prototype, but also subsequently easily enhance that prototype into a user friendly end-user application through graphical pages, or an application deployed as part of the software framework of a company.

3 Model parameters in AIMMS

In the model of bike sharing systems, there're concepts of stations and vehicles. They're modeled as two SETs with correspond parameters indicating the number of stations (`NrStations`) and vehicles (`NrVehicles`) in the problem.

```
SET:
   identifier   :  Stations
   indices      :  l, i, j
   definition   :  ElementRange(1, NrStations, 1, "s-") ;

SET:
   identifier   :  Vehicles
   index        :  v
   definition   :  ElementRange(1, NrVehicles, 1, "v-") ;
```

The properties of stations include their geography locations (`Xcoord` and `Ycoord`), `StationCapacity`, `StartInventory`(initial number of bikes) and service level agreements (`ServiceLevelMax` and `ServiceLevelMin`).

Once we have the locations of each station, it's then possible to know the distance between each of them and the traveling time from one station to another. Since all stations are in the same city, we assume all stations are fully connected (which means it's possible to travel between any of two stations in the city), and here we use the Manhattan distance instead of Euclid distance, to better fit the conditions in a city. However, to support real geolocation data in Bologna, which is in forms of latitude and longitude degrees, the result of `Distance` is in very small non-integer values. To support scheduling of resources in CP Solver, we have to make a scale to these small values into integer values. Since the absolute value and unit of `DistanceInTime` is not important for the CP solving process, we simply define it as an enlarged `Distance`:

```
PARAMETER:
   identifier   :  Distance
   index domain :  (i,j) | i <> j
   definition   :  ceil(abs(Xcoord(i) - Xcoord(j)) +
                        abs(Ycoord(i) - Ycoord(j))) ;
```

```
PARAMETER:
   identifier   :  DistanceInTime
   index domain :  (i,j)
   range        :  integer
   definition   :  1000*Distance;
```

4 The CP model

4.1 Activities, Variables and Resources

The CP model as a scheduling problem consists of two activities, *Pickup* and *Delivery*, together with two variables, PickupAmount and DeliveryAmount. They are all defined on the index domain of (Stations, Vehicles). The variable *PickupAmount* is the variable $y^{-}_{i,v}$ in [1] and the variable *DeliveryAmount* is the variable $y^{-}_{i,v}$. Activities are scheduled on timing set `ScheduleHorizon`, which is defined on integers between 1 and MaxTime (minutes in one day).

There's one sequential resource, *VehicleTime* and two parallel resources, *VehicleInventory* and *StationInventory*. Their definitions are exactly the same as in [1]. See Fig. 1 for the details of these definitions.

4.2 Constraints

There're five constraints in our CP model. The first two, SLAConstraintLow and SLAConstraintHigh correspond to the constraints (21) and (22) in original model [1], they represent the SLA constraints. The constraint PickupOrDelivery corresponds to the constraint (23) in original model, it's the so-called alternative resource (but here we didn't use cp::Alternative). For the rest four constraints (24-27) in original model, we've merged them into two constraints PickupPresent and DeliveryPresent, using AND and OR logical expressions. See Fig. 2 for the details.

4.3 Solver and Mathematical Program

The MP (Mathematical Program) is defined by minimizing the variable `TotalTime` (see Fig. 3), which has the following definition, which is the max end time of the last activity: And the Type of MP is COP (Constraint Optimization Problem) and we've applied a time limit on the Solver. The definition code is trivial.

5 GUI frontend

The GUI (Graphics User Interface) in AIMMS is very convenient to use. It provides an easy way to quickly generate the random data and visualization of problem solution. To be able to draw the actual routing information in the map, we need to define a binary parameter Arc(i,j), which is used in the network

```
ACTIVITY:
   identifier       :  Pickup
   index domain     :  (i,v)
   schedule domain  :  ScheduleHorizon
   property         :  Optional
   length           :  0 ;

ACTIVITY:
   identifier       :  Delivery
   index domain     :  (i,v)
   schedule domain  :  ScheduleHorizon
   property         :  Optional
   length           :  0 ;

VARIABLE:
   identifier       :  PickupAmount
   index domain     :  (i,v)
   text             :  "y(i,v)-"
   range            :  {0..VehicleCapacity(v)} ;

VARIABLE:
   identifier       :  DeliveryAmount
   index domain     :  (i,v)
   text             :  "y(i,v)+"
   range            :  {0..VehicleCapacity(v)} ;

RESOURCE:
   identifier       :  VehicleTime
   usage            :  sequential
   index domain     :  (v)
   schedule domain  :  ScheduleHorizon
   activities       :  Pickup(i,v), Delivery(i,v)
   group set        :  Stations
   group definition :  Pickup(i, v) : i,
                       Delivery(i, v) : i
   group transition :  (i, j) : DistanceInTime(i, j) ;

RESOURCE:
   identifier       :  VehicleInventory
   usage            :  parallel
   index domain     :  (v)
   schedule domain  :  ScheduleHorizon
   activities       :  Pickup(i,v), Delivery(i, v)
   level range      :  {0..VehicleCapacity(v)}
   initial level    :  StartVehicleCapacity(v)
   begin change     :  Delivery(i,v) : -DeliveryAmount(i,v)
   end change       :  Pickup(i,v) : PickupAmount(i,v) ;

RESOURCE:
   identifier       :  StationInventory
   usage            :  parallel
   index domain     :  (i)
   schedule domain  :  ScheduleHorizon
   activities       :  Pickup(i,v), Delivery(i, v)
   level range      :  {0..StationCapacity(i)}
   initial level    :  StartInventory(i)
   begin change     :  Delivery(i,v) : DeliveryAmount(i,v)
   end change       :  Pickup(i,v) : -PickupAmount(i,v) ;
```

Fig. 1. Activities and resources in the CP model

```
 CONSTRAINT:
   identifier       :  SLAConstraintLow
   index domain     :  (i)
   text             :  "C21"
   definition       :  StartInventory(i) +
   sum(v, DeliveryAmount(i,v)-PickupAmount(i,v)) >= ServiceLevelMin(i) ;

 CONSTRAINT:
   identifier       :  SLAConstraintHigh
   index domain     :  (i)
   text             :  "C22"
   definition       :  StartInventory(i) +
   sum(v, DeliveryAmount(i,v)-PickupAmount(i,v)) <= ServiceLevelMax(i) ;

 CONSTRAINT:
   identifier       :  PickupOrDelivery
   index domain     :  (i)
   text             :  "C23"
   definition       :  sum(v, Pickup(i,v).Present +
   Delivery(i,v).Present) <= 1 ;

 CONSTRAINT:
   identifier       :  PickupPresent
   index domain     :  (i,v)
   text             :  "C24"
   definition       :  (Pickup(i,v).Present = 1 AND PickupAmount(i,v) >= 1)
                       OR
                       (Pickup(i,v).Present = 0 AND PickupAmount(i,v) = 0) ;

 CONSTRAINT:
   identifier       :  DeliveryPresent
   index domain     :  (i,v)
   text             :  "C26"
   definition       :  (Delivery(i,v).Present = 1 AND DeliveryAmount(i,v) >= 1)
                       OR
                       (Delivery(i,v).Present = 0 AND DeliveryAmount(i,v) = 0) ;
```

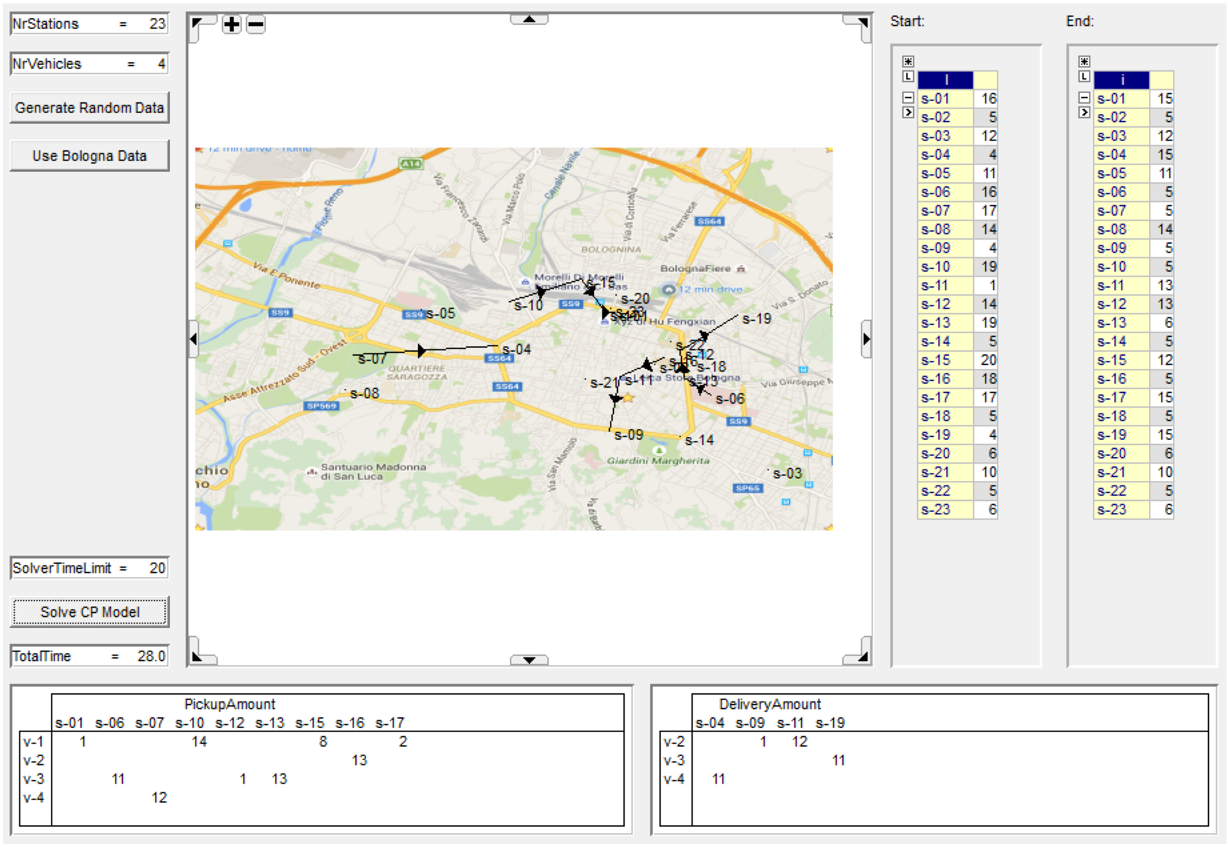
Fig. 2. Constraints in the CP model

```
VARIABLE:
   identifier       :  TotalTime
   range            :  free
   definition       :  max((i,v), max(Pickup(i,v).End,Delivery(i,v).End)) ;
```

Fig. 3. Constraints in the CP model

widget of AIMMS. The idea is to find all "direct" connections between stations in which there's no other stations in the middle for the same vehicle: (for rest of parameters used in this definition, please refer to the AIMMS project file)

For the entire GUI work we've built (with solutions displayed), please see Fig. 4.



NrStations = 23

NrVehicles = 4

SolverTimeLimit = 20

TotalTime = 28.0

Start:

i	
s-01	16
s-02	5
s-03	12
s-04	4
s-05	11
s-06	16
s-07	17
s-08	14
s-09	4
s-10	19
s-11	1
s-12	14
s-13	19
s-14	5
s-15	20
s-16	18
s-17	17
s-18	5
s-19	4
s-20	6
s-21	10
s-22	5
s-23	6

End:

i	
s-01	15
s-02	5
s-03	12
s-04	15
s-05	11
s-06	5
s-07	5
s-08	14
s-09	5
s-10	5
s-11	13
s-12	13
s-13	6
s-14	5
s-15	12
s-16	5
s-17	15
s-18	5
s-19	15
s-20	6
s-21	10
s-22	5
s-23	6

	PickupAmount								
	s-01	s-06	s-07	s-10	s-12	s-13	s-15	s-16	s-17
v-1	1			14			8		2
v-2								13	
v-3		11			1	13			
v-4			12						

	DeliveryAmount			
	s-04	s-09	s-11	s-19
v-2		1	12	
v-3				11
v-4	11			

Fig. 4. GUI frontend, it contains buttons for generating random data or Bologna data, edit boxes for setting some parameters, the number of bicycles in each generated stations, before and after the CP solving process. The solution is shown in two forms: 1) at the bottom, it shows how each vehicle should pickup and deliver the number of bicycles, and in the central map, the routing path of each vehicle is shown, in this way we could know the order of pickups/deliveries for each vehicle.

6 Experiments on the model

6.1 Bike stations in Bologna

The bike station data for Bologna is taken from a map on `http://www.bologna.bo/mappa-bike-sharing/`, see Fig. 5 and Table 1 for the location details of each station.

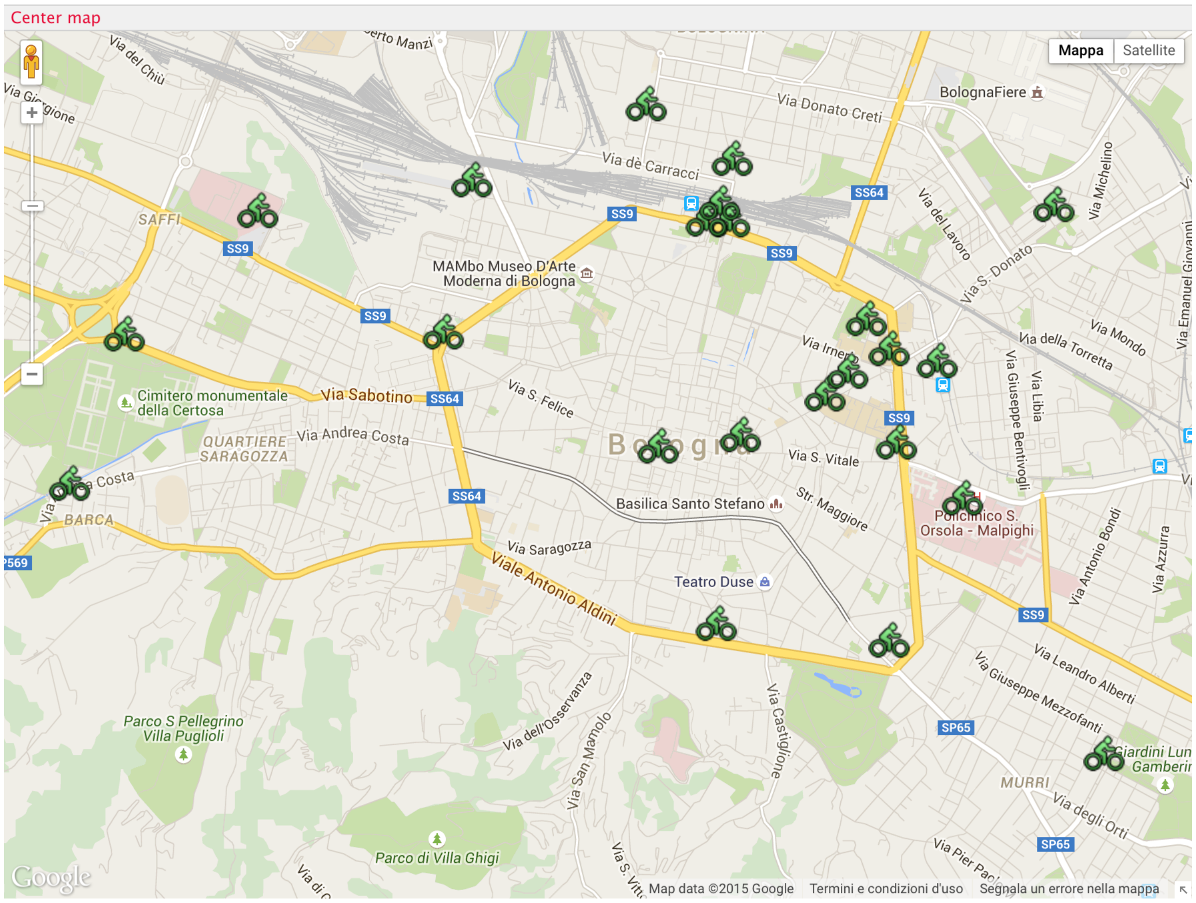


Fig. 5. Bike station map in Bologna

6.2 Other parameters used in the experiment

Our experiments are based on following typical parameters in real bike sharing systems:

Table 1. Bike stations in Bologna

ID	Location	Latitude	Longitude
1	Autostazione	44,504197 N	11,345882 E
2	Largo Alfredo Trombetti	44,4961895 N	11,3519957 E
3	Largo Lercaro	44,4797011 N	11,3699371 E
4	Mura di Porta San Felice	44,499049 N	11,327468 E
5	Ospedale Maggiore	44,504631 N	11,315512 E
6	Ospedale Sant'Orsola	44,49141 N	11,360846 E
7	Parcheggio Certosa Nord	44,497577569708 N	11,304716 E
8	Parcheggio Ghisello	44,492084 N	11,303447 E
9	Parcheggio Staveco	44,4856678 N	11,3450197 E
10	Parcheggio Tanari	44,506021 N	11,329324 E
11	Piazza di Porta Ravegnana	44,4943316 N	11,3465504 E
12	Piazza di Porta San Donato	44,4982667 N	11,3561362 E
13	Piazza di Porta San Vitale	44,494006 N	11,356606 E
14	Piazza di Porta Santo Stefano	44,484888 N	11,356124 E
15	Piazza Liber Paradisus	44,509521 N	11,340496 E
16	Piazza Vittorio Puntoni	44,4972324 N	11,3534553 E
17	Piazza XX Settembre	44,5042142 N	11,3443509 E
18	SFM S. Vitale	44,496388269708 N	11,357859269709 E
19	Via Garavaglia	44,5037962 N	11,3650535 E
20	Via Giacomo Matteotti	44,507015 N	11,346034 E
21	Via IV Novembre	44,493811 N	11,341295 E
22	Via Re Filippo	44,4996567 N	11,3546497 E
23	Viale Pietro Pietramellara	44,504958 N	11,345226 E

1. Each bike station has a capacity for holding 20 bikes.
2. Each vehicle has a capacity of 25 bikes.
3. The SLAs are targeted at 5 at minimal and 15 at maximum.
4. We assume at the beginning, all vehicles are empty (without taken any bike), and at the end of rebalancing they don't need to deliver all bikes that were picked up.
5. There's no initial costs when any vehicle reaches to the first station.

6.3 Experiments on the number of vehicles

In this experiment, we use Bologna station locations with random generated initial status of each station. We want to know, with each status, what's the best number of vehicles that we should deploy. Keeping in mind that, adding one more vehicle could cost more money and human resource, but two few of vehicles will not be able to match the service requirements, and it's quite possible that, before the one round of vehicle routing is finished, the status of each stations has greatly changed, which means after the whole process there're still lots of unbalanced stations left.

Table 2. Experiments on the number of vehicles

Number of vehicles	**Data 1**	**Data 2**	**Data 3**	**Data 4**
1	119	128	166	111
2	55	62	77	50
3	43	42	49	30
4	27	29	35	29
5	25	18	26	29
6	21	13	22	29
7	21	13	19	29

From the experiment results shown in Table 2, we can make following conclusions:

– More vehicles involves more time for the CP solver to find the optimized solution.
– There're easy cases (like **Data 4**) and hard cases (like **Data 3**). For easy cases, a small number of vehicles will reach the best solution, and further increasing the number of vehicles doesn't have any impact (just a waste of money and human resources). For hard cases, the solution could always better when increasing the number of vehicles from 1 up to 7, and for too many vehicles, the CP Solver gets terminated by the time limits, which means a even better solution may exist but the Solver failed to find it.
– Different random cases give similar results when the number of vehicle is also the same.

– Generally speaking, the ideal number of vehicles used in Bologna is four (4). Further increasing the number of vehicles doesn't make too much benefits. However this conclusion is also based on the SLAs and other parameters.

6.4 Experiments on the SLAs

In the last experiments, we assume SLAs are all targeted at 5 at minimal and 15 at maximum. Now we set the number of vehicles as fixed to 4, and try different SLAs: `(2 18), (3 17), (4 16), (5 15), (6 14), (7 13)`. It's imaginable that, if the SLA is too relax, like `(2 18)`, most of stations will be self-sufficient state that a vehicle doesn't need to visit (unless some bikes in it are needed for delivering to other stations). And when SLA is getting more and more strict, the expect routing time should also increase. For the initial state, we've used the hard case (data 3) in the last experiment. And indeed, we did get the expect results from the model, as shown in Table 3.

Table 3. Experiments on different SLAs

s_i^{min}	s_i^{max}	**total time**
2	18	1
3	17	16
4	16	30
5	15	35
6	14	37
7	13	37

Noticed that, in real cases, the s_i^{min} and s_i^{max} for each station may be different, and the actual value should be based on statistical data observed from historical data. And the value of s_i^{min} and s_i^{max} are not necessary symmetric: some stations may have more pickups than deliveries, other stations may have more deliveries than pickups. But generally speaking, a bigger SLA range $(s_i^{max} - s_i^{min})$ will result in less routing efforts.

6.5 Performance of the CP solver

Our experiment shows that, in all generated problems, the CP solver quickly conveys to the final time costs, but there're easy and hard problems needing quite different time to stop the Solver with optimal solution. In good cases we need only 5 seconds to get the optimal results and in bad cases the Solver needs over 30 seconds to get it, but in about 10 seconds it can always get a quite good result, just the searching process didn't terminate. General optimization technologies like variable priorities doesn't work in this model, because there're two few types of variables and activities, and they're tightly related with each other at the same priority. Generally speaking, in 20 seconds, the solver can always get a almost perfect solution for a 30-stations problem.[1].

7 Conclusions

In this project, we've explored the Inventory Balancing and Vehicle Routing problems in Bike Sharing Systems and have built a Constraint Programming model in AIMMS software, followed by the modeling idea proposed by previous researchers. We've learnt how to use AIMMS to express such ideas and build a simple GUI page for prototype solution software. Then we've done some experiments using the real station data from Bologna bike sharing system, and the experiments suggested an ideal number of vehicles at a balance of costs and performance.

References

1. Schuijbroek, J., Hampshire, R., van Hoeve, W.J.: Inventory rebalancing and vehicle routing in bike sharing systems. Technical report, Tepper School of Business Working Paper 2013-E1, Carnegie Mellon University (2013)
2. van Hoeve, W.J.: Developing constraint programming applications with aimms. In: CP2013, COSpel workshop, Citeseer (2013)

[1] The experiment environment is a Windows 8.1 virtual machine running on Apple Macbook Pro with 2.6GHz Intel Core i7 processor, AIMMS only uses one processor