Università di Pisa

Scuola di Dottorato in Ingegneria "Leonardo da Vinci"

Corso di Dottorato di Ricerca in Veicoli Terrestri e Sistemi di Trasporto, XXV Ciclo

Settore Scientifico Disciplinare ICAR/05

SIMULATION OF A CAR-SHARING TRANSPORT SYSTEM FOR URBAN MOBILITY

Dottorando:
Alessandro Farina

Relatore:
Dott. Ing. Elvezia Maria Cepolina

Coordinatore del Dottorato:
Chiar.mo Prof. Ing. Roberto Roncella

Pisa, Febbraio 2013

LIST OF SYMBOLS

I	Instance of a minimization problem
$z^A(I)$	Value of the solution provided by the heuristic algorithm A.
$z^*(I)$	Optimum value of the instance
S	Solution of an optimization algorithm
P	Optimization problem
f	Generic objective function
F	Feasible region in an optimization algorithm
$N(s)$	Neighbourhood of a solution s
p_{ij}	Probability of moving from a solution i to a solution j in a Simulated Annealing algorithm
δE	Increase in energy in Simulated Annealing algorithm
k	Boltzmann constant in Simulated Annealing algorithm
T	Temperature in Simulated Annealing algorithm
T	Tabu List in Tabu Search algorithms
t	Tabu tenure in Tabu Search algorithms
z	Cost function, which has been minimized in the thesis, and which takes into account system's cost and users' cost
$\mathbf{g}(\mathbf{s})$	Constraint of the optimization problem of the thesis
$C_d^{\,s}$	Transport system's cost in the objective function z
$C_d^{\,u}$	Users' cost in the objective function z
t_w^{50}	50th percentile of users' waiting times, in minutes

t_w^{90}	90th percentile of users’ waiting times, in minutes
t_w^{95}	95th percentile of users’ waiting times, in minutes
C_d^f	Cost of the fleet in the objective function z
C_d^{run}	Cost of system’s running in the objective function z
C_d^{su}	Cost of system’s setup in the objective function z
C_d^{man}	Cost of system’s management in the objective function z
C_d^{ts}	Cost of the tickets in the objective function z
n_v	PICAV fleet dimension
c_v	PICAV fleet purchase cost
r	Discount rate in the objective function
lt	PICAV vehicle lifetime in years in the objective function
C_d^r	Daily cost of relocation in the objective function z
c_r	Cost of each minute of relocation
t_{rj}	Time spent in relocation by the jth vehicle
c_w	Cost of a unit of waiting time = 0.10 €/min in the objective function
t_{wi}	Waiting time of the ith user in minutes in the objective function
$\mathbf{s}$	Vector solution of the optimization algorithm. Its components are: the number of vehicles at each station at the beginning of the simulation for the first management strategy; fleet dimension, low critical threshold and low buffer threshold for the other two strategies
$f(\mathbf{s})$	Cost function, equal to z but without the flat terms
N^a	Research space; for the first management strategy a = number of stations

$h(\mathbf{s})$	Cost function value, with the penalty term
$g_i(\mathbf{s})$	i-th constraint ($i = 1, 2$ or 3) of the optimization problem
$\hat{\mu}$	Weight of the penalty function (penalty factor)
β	Multiplicative coefficient of the penalty coefficient: namely $\mu_{k+1} = \beta\mu_k$
α	Coefficient of temperature reduction: usually $0.7 \leq \alpha \leq 0.95$
p_0	Probability of accepting the worse solution at the initialization of the Simulated Annealing algorithm.
T_0	Starting temperature of the Simulated Annealing algorithm
T_k	Temperature at the generic step k of the Simulated Annealing algorithm
p	Probability of choosing a worse solution in the Metropolis algorithm
$\mathbf{s_n}$	The new vector solution in the Simulated Annealing algorithm
s*	Current vector solution and departure point for the successive iteration of the parallel optimization. Or, vector solution which minimizes the penalty function h_k in a given iteration. See section 5.3.1.1. for further details.
f_{ob}	Cost function in an optimization problem
c_v	Daily amortization cost of each vehicle
μ	Penalty coefficient
k	Pedestrian density (pedestrian/m^2)
f_{ob}^{new}	Objective function calculated at the current iteration of the Simulated Annealing algorithm
f_{ob}^{old}	Objective function calculated at the previous iteration of the Simulated Annealing algorithm
a_r	Initial acceptance ratio of the Simulated Annealing algorithm

v	PICAV vehicle's speed
a	Acceleration of the vehicle in the vehicle – pedestrian model and in the formula for calculating resistances to motion
g	Variable which can assume only two values, either 0 or 1, in the vehicle – pedestrian model
a, b, c, d, e	Coefficients of the polynomial vehicle – pedestrian model
a, b	Coefficients of the exponential or logarithmic vehicle – pedestrian model
k_o	Threshold in the density for what regards the discontinuous linear model vehicle – pedestrian
P	Weight of the vehicle and of the occupant in kg in the formula for calculating resistances to motion
g	Constant of universal gravitation
c_r	Rolling coefficient: i.e. specific resistance to rolling, in the formula for calculating resistances to motion
i	Average slope in ‰, in the formula of the resistances to motion
ρ_a	Air density
C_x	Aerodynamic coefficient
S	PICAV vehicle section, in the formula of the resistances to motion
R	Total resistance to motion
SOC	State of charge of the battery
Q_e	Quantity of electricity supplied
C	Battery capacity
U_m	Battery voltage
E_e	Energy supplied by the battery

P_{DC}	Power in entrance to the inverter
F_r	Force at the wheels
η_c	Efficiency of the inverter
η_m	Efficiency of the engine
P_r	Power at the wheels

LIST OF FIGURES

LIST OF TABLES

SOMMARIO:

La presente tesi consiste nella progettazione e simulazione di un sistema di trasporto innovativo di tipo car sharing per la mobilità urbana. Il sistema oggetto dello studio è basato su di una flotta di veicoli intelligenti, monopersona, accessibili a tutte le categorie di utenti e progettati per operare nelle aree pedonali dei centri storici urbani; questi veicoli sono chiamati PICAV (Personal Intelligent City Accessible Vehicle). Il sistema di trasporto proposto non richiede prenotazione, non richiede che sia specificato l'orario in cui il veicolo verrà restituito, e non richiede che il veicolo venga restituito nella medesima stazione da cui è stato preso. Queste caratteristiche garantiscono agli utenti un elevato grado di flessibilità; d'altra parte, però, le stazioni possono essere sbilanciate in diversi periodi della giornata: alcune stazioni hanno un elevato numero di veicoli disponibili, metre altre hanno un elevato numero di utenti in coda. Perciò, è necessaria la riallocazione. Tre diverse strategie di riallocazione vengono proposte: nella prima strategia le riallocazioni vengono compiute direttamente dagli utenti del sistema, mentre nelle altre due i veicoli riallocano automaticamente senza necessità di essere guidati. Nelle prime due strategie di gestione, i veicoli sono accessibili esclusivamente in corrispondenza delle stazioni. Nella terza strategia, i veicoli sono accessibili da qualunque punto dell'area di studio.

Allo scopo di fornire ai decisori politici uno strumento utile per testare il sistema di trasporto in diverse realtà, è stato sviluppato un micro simulatore, basato sulla logica ad oggetti. Il micro simulatore fornisce in output le performance del sistema di trasporto, in termini di distribuzione dei tempi di attesa degli utenti e di efficienza del sistema, che è inversamente proporzionale alla dimensione della flotta e al numero di riallocazioni. E' stato poi sviluppato un algoritmo meta euristico di ottimizzazione, finalizzato ad ottimizzare le caratteristiche del sistema di trasporto. L'algoritmo di ottimizzazione richiama il micro simulatore per calcolare i suoi dati di input.

Il micro simulatore è stato calibrato e validato, e successivamente è stato applicato a studiare due scenari: ossia il centro storico di Genova, e la parte vecchia di Barreiro, in Portogallo. Infine, è stata realizzata un'analisi di sensibilità al fine di studiare le performance del sistema al variare della domanda, della rete, della dimensione della flotta e delle caratteristiche del sistema di trasporto

ABSTRACT

An innovative car-sharing systems for urban areas is proposed. The proposed system is based on a fleet of Personal Intelligent City Accessible Vehicles (PICAVs). The following specific services are provided: instant access, open ended reservation and one way trips. All these features provide users with high flexibility, but create a problem of uneven distribution of vehicles among stations. Therefore, relocations must be performed. Different relocation procedures are proposed: in the first relocation scheme relocations are performed by users while in the other two vehicles relocate automatically thanks to their automation. In the first two management strategies vehicles can be accessed and returned only at stations while in the last one they can be accessed also along the roads. In order to provide transport managers with a useful tool to test the proposed systems in different realities, an object-oriented micro simulator has been developed. The simulation gives in output the transport system performance, in terms of distribution of user waiting times, and the transport system efficiency, which is inversely proportional to the fleet dimension and the number of relocation trips. A meta heuristic optimization algorithm has been developed to optimize the transport system's characteristics. The optimization algorithm recalls the micro simulator to calculate the optimization's input data.

The micro simulator has been calibrated and validated, and afterwards applied to study two scenarios: Genoa historical city centre, Italy, and Barreiro old town, Portugal. Finally, a sensitivity analysis has been performed in order to study the performances of the system according to modifications of the demand, or of the fleet dimension or of the transport system characteristics.

Introduction

The concept of vehicle sharing has begun to emerge as a novel and popular mobility concept.

In car sharing schema, individuals gain access to vehicles by joining an organization that maintains a fleet of vehicles that are parked in designated, leased spaces in a network of locations. Vehicles are accessed on an as-needed basis, and members are typically charged each time they use a vehicle (Shaheen and Cohen 2007). Vehicles can be reserved minutes or months in advance for specific blocks of time, online or by phone. To use a car sharing vehicle, members simply walk to the car at the reserved time, use a wireless security keycard to unlock the door, and drive as usual. As the reservation ends, they return the car to its exclusive-use parking space, lock it with their keycard, and walk away. An onboard computer collects and wirelessly transmits trip data. Charges are either automatically billed to the member's credit card or deducted from their bank account.

From the point of view of non users, in cities with high population densities, car sharing exhibits great promise in decreasing parking demand, lowering emissions and guarantee mobility for all (Britton et al., 1999).

According to recent North American studies (Millard-Ball et al., 2005), each car sharing vehicle removes an average of 15 privately owned cars from the community, as participants sell a vehicle or forgo a planned purchase. The resulting decrease in local parking demand creates opportunities to permanently reallocate the land for additional green space, new mixed-use development, or other community needs. Furthermore, the vehicles these members sell or avoid purchasing tend to be the oldest, most polluting, and least reliable on the road. They are replaced by a relatively small number of high-efficiency, low-emission vehicles, including gasoline-electric hybrid cars, creating even greater improvements in local air quality, noise, and emissions.

Shared cars also generate social benefits, creating an affordable alternative to ownership for lower-income workers, students, and seniors. Those otherwise at risk of being marginalized can affordably maintain their mobility and participate fully in society.

From the point of view of users, they gain some of the benefits of private vehicle use without the costs and responsibilities of ownership. These benefits include comfort and the possibility to travel without unknown people. Privacy in fact is an important factor in the current society, and it will be even more in the future because of aging society.

However, from the point of view of users, car sharing is yet less flexible than private cars. In fact traditional car sharing schemas require: booking in advance and at the booking time, the returning time has to be specified. Trips by shared cars have to start and end in stations which are in limited number in the territory; users have to return the vehicle at the pick-up station.

One of the scopes of this research is to develop a car sharing system which provides users high flexibility: booking is not required and users can access the vehicles at the spur of the moment, users can keep the shared cars as long as they want and one way trips are possible. This flexibility puts severe problems in the transport system management since the distribution of vehicles in the territory can become unbalanced. This can lead to long queues of users waiting for a vehicle in some stations and long queues of unused vehicles in other stations. To avoid these situations, huge fleets can be necessary or a consistent number of operators is required to move vehicles between stations with a surplus of vehicles and stations with a lack of vehicles. Both these solutions have a high cost that often has determined the failure of car sharing systems. To limit this cost, three fleet management strategies are proposed in this thesis: they are alternative strategies that can solve the unbalancement problem in flexible car sharing systems in different realities.

In the first one, vehicles are always driven by users, and the uneven distribution of vehicles at the stations is overcome through users' redirection. Vehicles are available only at stations and also must be returned at stations.

The other two management strategies require instead vehicles to move in a fully automatic mode during relocation trips. In the second car sharing system, vehicles again can be accessed only at stations and returned only at stations.

In the third management strategy, instead, vehicles can be accessed both at stations and along the roads, and can also be returned at both. The user gives his position at the system manager and he delivers a vehicle to the user.

To check the performance of the proposed management strategies, a microscopic simulator has been developed following an object – oriented logic and the code has been written in Python language. The simulator receives in input the transport demand, the characteristics of the network, the relocation strategy and the PICAV fleet. It allows to track the second-by-second activity of each PICAV user, as well as the second-by-second activity of each vehicle. The simulator gives in output the transport system performances, in terms of Level Of Service provided to users and in terms of efficiency from the management point of view.

The microscopic simulator is a decision support tool that will help car sharing operators to plan and manage flexible car sharing schemas in different realities.

Using this tool, planners can explore the consequences of proposed planning decisions through the creation of alternative scenarios. The process of using the tool to explore the full consequences of potential decisions and assumptions enhances understanding of the system and of the real trade-offs among different alternatives. The microscopic simulator does not prescribe "best" solutions.

Since the input data are several and some of them cannot be directly quantifiable by observations on the field, an optimization procedures has been developed. It aims to find “good” values for critical management parameters.

The transport system this thesis deals with is a flexible car sharing system for historical city centers and pedestrian areas. The fleet is composed of PICAV vehicles. A PICAV vehicle is a one person electrically powered vehicle with very small dimensions. It has been designed to be fully accessible from people with mobility impairments. It provides driving assistance as well as automatic driving. It is able to move on uneven pavements and to move with agility in pedestrian flows.

Such a transport system aims to improve accessibility to all and release part of the repressed pedestrian mobility: people that have difficulties in covering long distances by foot, not only because of disabilities but also because, for example, of heavy items to carry, are excluded by performing activities in such areas which are often very attractive but where very often alternative transport modes cannot operate.

The proposed transport system has been planned and designed for two areas: Genoa historical city centre, Italy, and Barreiro old town, Portugal. The two areas are very different, either from the morphological point of view and the transport system organization. The first area is characterized by narrow and steep roads with steps, and high pedestrian flows; the second scenario instead is characterized of wider and flat roads. The proposed transport system integrates with different modes of transport. In particular, in the first scenario it integrates with public transport, such as bus and underground. In the second scenario, it integrates with both public and private transport.

Some of the transport system parameters have been optimized for the two areas under study.

The thesis is organized as follows.

Chapter 1 focuses on the context of the research: an overview on the main characteristics, mobility problems and adopted solutions for urban areas and, specially historical city centers, is provided. The PICAV vehicle is then introduced with its technical characteristics.

In chapter 2, a state of the art on vehicle sharing systems is given, with particular focus on the relocation procedures.

In chapter 3, the proposed transport system is described in detail. Particular focus is given to the three proposed management strategies.

In chapter 4, the modeling and the micro simulation is described. Firstly, an overview about simulation and object-oriented logic is provided. After, the modelization of the system and the micro simulator code are described in detail.

In chapter 5, the optimization procedure is described in detail. Because of the problem characteristics, a meta heuristic algorithm has been chosen.

In chapter 6, the calibration of the simulator's parameters and its validation are reported.

In chapter 7 and 8, the simulator has been applied to study two scenarios: the historical city centre of Genoa and the old town of Barreiro.

In chapter 9, a sensitivity analysis is described.

The research has been funded by the European Commission, specifically by the European project PICAV (Personal Intelligent City Accessible Vehicles) (FP7-SST-2007-RTD-1).

Chapter 1. The context

Introduction

Urban areas are highly congested, in particular historical city centres. The main reasons of congestion regard: the high level of urbanization (about 75% of the European population lives in cities); and the conformation of historical city centres, characterized of narrow roads, in some cities also steep and with steps, where also conventional public transport operates with difficulty.

Congestion causes delays in trips: it is estimated that about 1% of the European GDP is lost by considering the cost associated to urban congestion (Lindholm and Behrends, 2010). Congestion, as side effects, also increases the number of road accidents, and causes pollution, whose long-term effects are dangerous for people's health.

Therefore, a shift from private to public transport is desirable. This can be achieved through limitations on private transport, and through improvements of public transport. In order to avoid the decrease of the transport demand, public transport must be attractive and provide a high quality service.

Moreover, the population is ageing. Transport systems must therefore take into account the mobility needs of all categories of users.

It is in this context that the proposed transport system has been developed. The system proposed is a car sharing one, and it is aimed to provide users high comfort and flexibility, as similarly as possible to private car. It makes use of small electric vehicles, called PICAV, which are: non pollutant, thanks to their electric propulsion; ergonomic, designed for being accessible also by people with some impairments; and capable of travelling in the majority of urban environments, thanks to their capability to climb steps and overcome high slopes. PICAV vehicles are personal and therefore satisfy also the need for privacy of the users; this aspect is particularly relevant for aged people.

The chapter is organized as follows. Firstly, an overview on the main problems related to urban mobility is performed. Focus has been given to the main problems related to congestion, i.e. pollution and accidents, and to the main characteristics of historical city centres. Afterwards, the current trends, in terms of high growth of urbanization and ageing of society, are presented. Moreover, the policies aimed to perform a shift from private to public transport, and therefore reduce the number of circulating vehicles, are presented. Furthermore, the main advantages of car sharing and bike sharing systems are presented. Finally, the main characteristics of the proposed transport system and of the vehicle are presented.

1.1. Urban mobility.

1.1.1. Main problems related to road congestion in urban areas

1.1.1.1. Air pollution.

Results of a research performed by OMS-ARPAT (2006) show that more than 8,000 deaths per year occur in Italy because of long-term effects of pollution. The European directive 99/20/EC has set a limit for the year 2010 of 20 μg/m^3 for the concentration of PM10. The chronicle effects of PM10 pollution are co responsible of 742 deaths/year for lung cancer, 2562 for heart attack and 329 for stroke. In year 2005 several northern Italian cities have reached 35 days of excess of 50 μg/m^3 only in the months January – March (ARPAT news n° 107-2006).

The fine particulate, i.e. PM10, is a pollutant composed by particles which may be both solid and liquid and are in suspension in the air. These particles are varied for dimension, composition and origin, and their properties are summarised in their aerodynamic diameter. The smallest particles are called PM 2.5, whose diameter is smaller than 2.5 μm, they are not removed by precipitations and therefore can remain in the atmosphere also for several days. The most dangerous particulate is the PM 2.5, as they penetrate in the pulmonary alveoli. The main components of the particulate are sulphate, nitrate, ammonia, sodium chloride, carbon, mineral powders and water.

1.1.1.2. Road accidents.

In year 2010, 211,404 accidents have been registered in Italy, with 4,090 deaths and 302,735 injuries. Compared to 2009, a slight decrease is registered in the number of accidents (-1.9%) and injuries (-1.5%) and a more consistent decrease in the number of deaths (-3.5%). Compared with the target set by the European Union in the White Paper of 2001, which provided the reduction of deaths by 50% until 2010, Italy has reached a 42.4% of decrease in the number of deaths, which is conformed with the EU27 average, equal to -42.8% (ISTAT, 2012).

In urban roads, in Italy, 160,049 accidents have been registered in year 2010, with 218,383 injuries and 1,759 deaths. In motorways accidents have been 12,079 and 20,667 injuries and 376 deaths. In extra urban roads, accidents have been 39,276 with 63,685 injuries and 1,955 deaths. The most serious accidents happen in extra urban roads, with 5 deaths in 100 accidents; in urban roads this index is much less, i.e. 1.1 deaths by 100 accidents. The category of vehicles most involved in road accidents are: cars (67.8%), motorbikes (18.8%) and bicycles (3.9%). Statistical data about road accidents are provided in the tables 1.1 – 1.3 and in figure 1.1.

About 60% of accidents in Italy are concentrated in the biggest cities, such as: Torino, Milano, Verona, Venezia, Trieste, Genoa, Bologna, Firenze, Roma, Napoli, Bari, Palermo, Messina, Catania. Accidents in large metropolitan areas constitute a high quota of the overall incidentality in Italy: Roma 8.7%, Milano 5.7%, Genoa 2.3% and Torino 1.8%. The highest mortality rate is registered in Verona and

Palermo, with about 1.6 – 1.7 deaths per 100 accidents, while the lowest is registered in Milano and Bari (0.5).

Table 1.1. Road accidents in years 2001-2010. (Source: ISTAT, 2012). Years 2001 – 2010, absolute values, mortality index and percentual variations.

YEARS	Road accidents	Killed	Injured	Mortality Index (a)	Variations in change of the number of deaths in respect to the previous year (b)	Variations in change of the number of deaths in respect to 2001 (c)
2001	263.100	7.096	373.286	2,70	-	-
2002	265.402	6.980	378.492	2,63	-1,6	-1,6
2003	252.271	6.563	356.475	2,60	-6,0	-7,5
2004	243.490	6.122	343.179	2,51	-6,7	-13,7
2005	240.011	5.818	334.858	2,42	-5,0	-18,0
2006	238.124	5.669	332.955	2,38	-2,6	-20,1
2007	230.871	5.131	325.850	2,22	-9,5	-27,7
2008	218.963	4.725	310.745	2,16	-7,9	-33,4
2009	215.405	4.237	307.258	1,97	-10,3	-40,3
2010	211.404	4.090	302.735	1,93	-3,5	-42,4
2011	**205.638**	**3.860**	**292.019**	**1,88**	**-5,6**	**-45,6**

(a) The mortality index is the ratio between the number of killed persons in road accidents out of total road accidents resulting in death or injury (per 100)

(b) The variations in change of the number of deaths in respect to the previous year is: $(\frac{M^t}{M^{t-1}}-1)*100$

(c) The variations in change of the number of deaths in respect to 2001 (2001=100) is: $(\frac{M^t}{M^{2001}}-1)*100$

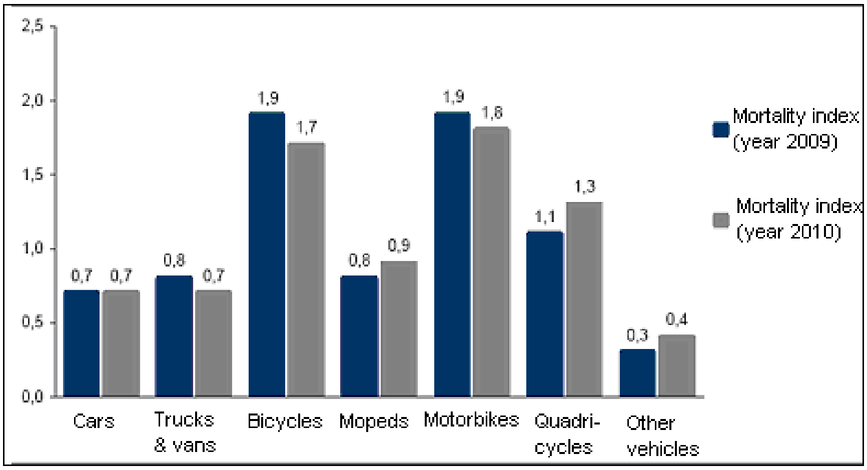


Fig. 1.1. death index according to the various typology of vehicles. As shown in the figure, the “weak” users, such as bicycles and motorbikes, register the higher percentage of deaths. (Source: ISTAT, 2012).

(a) The mortality index is calculated from the ratio between number of deaths and number of vehicles, by vehicle category, by 100
(b) From calculation of deaths and injuries pedestrians are excluded.

Table 1.2. Accidents in Italy according to the typology of roads. (Source: ISTAT, 2012). Year 2011 (absolute values, mortality index and variations in change 2011/2010).

ROAD TYPE	Road accidents	Killed	Injured	Mortality Index (a)	Variation in change Accidents 2011/2010	Variation in change Killed 2011/2010	Variation in change Injured 2011/2010
Inside urban area roads	157,023	1,744	213,001	1.1	-1.9	-0.9	-2.5
Motorways	11,007	338	18,515	3.1	-8.9	-10.1	-10.4
Rural roads	37,608	1,778	60,503	4.7	-4.2	-9.1	-5.0
Total	**205,638**	**3,860**	**292,019**	**1.9**	**-2.7**	**-5.6**	**-3.5**

(a) The mortality index is the ratio between the number of killed persons in road accidents out of total road accidents resulting in death or injury (per 100).

Table 1.3. Killed persons in road accidents within countries of the European Union (EU27). Years 2010 and 2011 (absolute numbers and variations in change) (a) (Source: ISTAT, 2012).

COUNTRIES EU27	Absolute numbers			Variations in change *(b)*		
	2001	2010	2011	2010/2001	2011/2001	2011/2010
Austria	958	552	523	-42.4	-45.4	-5.3
Belgium	1,486	840	875	-43.5	-41.1	4.2
Bulgaria	1,011	776	658	-23.3	-34.9	-15.2
Cyprus	98	60	71	-38.8	-27.6	18.3
Denmark	431	255	221	-38.5	-48.7	-13.3
Estonia	199	79	101	-60.8	-49.2	27.8
Finland	433	272	292	-37.6	-32.6	7.4
France	8,162	3,992	3,970	-51.1	-51.4	-0.6
Germany	6,977	3,651	4,002	-47.7	-42.6	9.6
Greece	1,880	1,258	1,087	-31.9	-42.2	-13.6
Ireland	411	212	186	-48.4	-54.7	-12.3
Italy	**7,096**	**4,090**	**3,860**	**-42.4**	**-45.6**	**-5.6**
Latvia	558	218	179	-60.9	-67.9	-17.9
Lithuania	706	299	297	-57.5	-57.9	-0.7
Luxembourg	70	32	33	-54.3	-52.9	3.1
Malta	16	15	17	-6.3	6.3	13.3
The Netherlands	1,083	640	661	-40.9	-39.0	3.3
Poland	5,534	3,907	4,189	-29.4	-24.3	7.2
Portugal	1,670	845	785	-49.4	-53.0	-7.1
United Kingdom	3,598	1,905	1,958	-46.0	-45.6	2.8
Czech Republic	1,334	802	707	-39.9	-47.0	-11.8
Romania	2,454	2,377	2,018	-3.1	-17.7	-15.1
Slovakia	625	353	324	-43.5	-48.2	-8.2
Slovenia	278	138	141	-50.4	-49.3	2.2
Spain	5,517	2,478	2,056	-55.2	-62.7	-17.0
Sweden	531	266	319	-49.9	-39.9	19.9
Hungary	1,239	740	638	-40.4	-48.5	-13.8
EU27	**54,355**	**31,052**	**30,168**	**-42.8**	**-44.5**	**-2.8**

(a) Source: European Transport Safety Council, Annual PIN report. Year 2012

(b) The variations in change 2010/2001 e 2010/2009 are calculated by means the formulas: $(\frac{M^{2010\ e\ 2011}}{M^{2001}} - 1) * 100$; e $(\frac{M^{2010}}{M^{2009}} - 1) * 100$

1.1.2. The development and the main problems of historical city centres

European cities have a specific conformation given by historical reasons: the inner zone is the oldest part, around which the new city has been built in a concentric manner in successive phases. The old town is characterized of narrow and congested roads, with reduced dimension, uneven pavements and high pedestrian flows.

The historical city centre has developed in the centuries, from the Roman age to the beginning of the Industrial Revolution, when the new quarters were built. A full description of the development of cities can be found in Astengo (1966). The following parts can be observed in historical city centres:

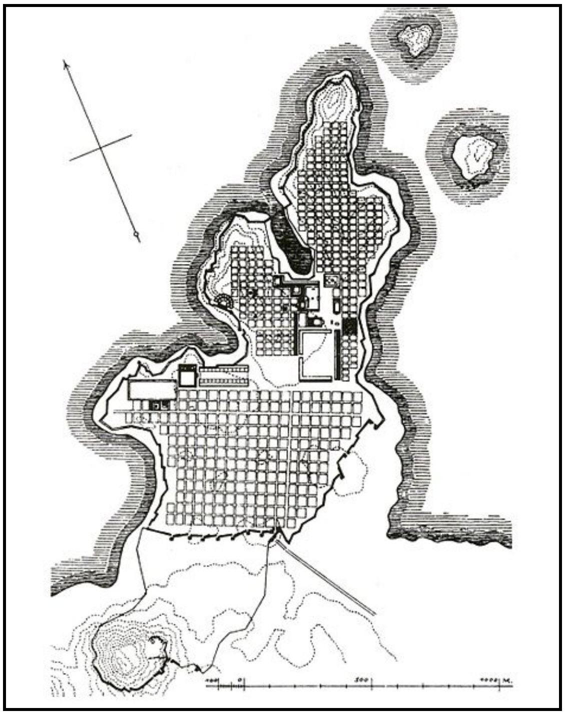


Fig. 1.2. The project for the urbanization of Miletus, traditionally attributed to Ippodamus (Miletos Stadt Plan).

- The Roman city. It is the most inner part. Roman cities are of squared shape, with perpendicular roads. This shape recalls the disposition of military camps – several Roman cities had been Roman military camps before – and it has the merit of allowing quick trips inside the urban areas. There were two main roads, called *cardo* and *decumano*, which were perpendicular with each other and crossed in correspondence of the main square. All the other roads were parallel to these and also crossed in perpendicular. This specific shape of the cities has been introduced by Ippodamo from Mileto, a Greek architect, who made some urbanization plans for Selinunte, Priene and Mileto (see fig. 1.2). However, outside the city walls all the roads, which connected the city to the other cities and villages, were radial, in order to reduce the travel time. Obviously when natural constraints existed this did not happen, but in all the Roman cities of the Po valley the roads outside the city were radial. This aspect is very important for the following.

- The medieval city. It grew around the Roman city, and all the new quarters were organized with radial roads. These new quarters indeed grew around the old extra urban Roman roads, which in fact had this conformation. These improvements of the city dimension however occurred as all the space inside the walls was consumed.
- Further improvements in the Renaissance and Modern Age. After all the space inside the city walls was consumed, other new quarters were built around the city walls, and one or two newer city walls were built, to protect the new quarters. The old walls were usually conglobated in the new houses. Therefore it is not unusual to see some parts of the city walls which have become walls of a house, and also it sometimes happens to see along the roads of the city centre some "isolated gates" which were the former access gates of the cities (see fig. 1.5). The newly built roads continued the older ones, and the new quarters maintained the shape of the old ones. The roads in the oldest quarters were usually much narrow because all the available space has been exploited for construction before increasing the dimension of the city – and the needs of mobility in those periods were much different from nowadays. A screenshot of the development of Bologna in the thirteenth century is provided in figure 1.6.

The development the historical city centre of Bologna in the centuries is shown in figures 1.3 and 1.4.

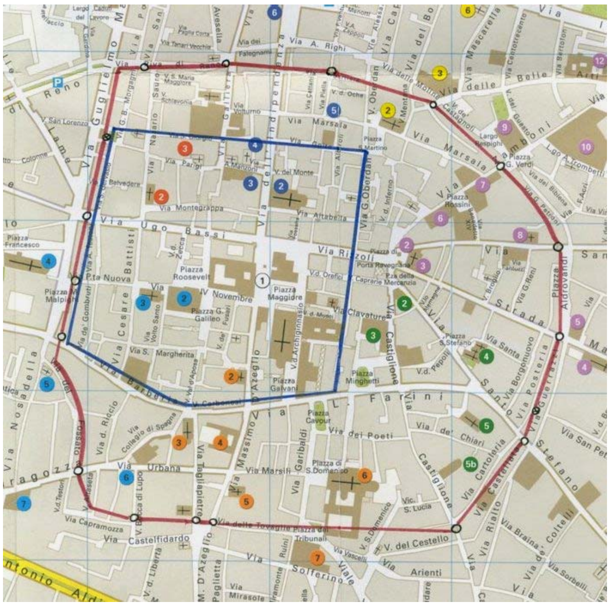


Fig. 1.3. A screenshot of the most inner part of Bologna centre. The Roman city is the part circled in blue, the early middle aged city is circled in red. The roads clearly show the characteristics described above (Source: archeobo.arti.beniculturali.it).

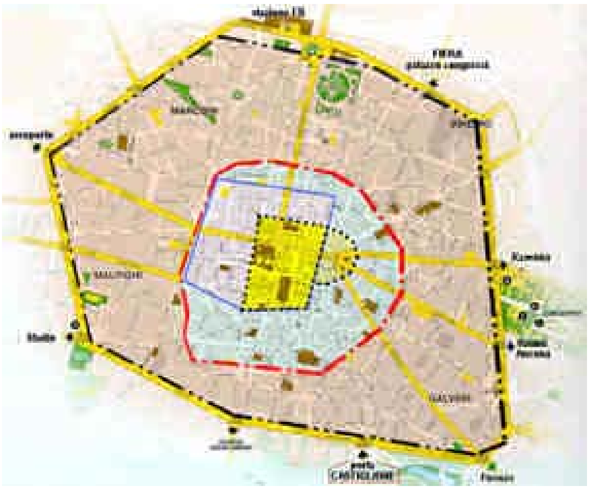


Fig.1.4. The development of Bologna historical city centre: the Roman city (contoured by the blue line in its final development), the early-middle aged city (contoured by the red line, the walls were built in year 1000), the outer walls (the black line). The outer walls were built in the thirteenth century and have been demolished at the beginning of the twentieth century. (Source: archeobo.arti.beniculturali.it)

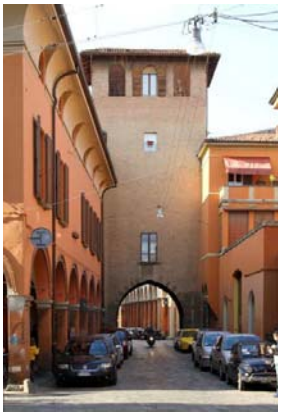


Fig. 1.5. An access gate of the Early Middle Aged walls conglobated in the buildings.

Then in the nineteenth century, because of the industrial revolution, a massive urbanization took place and the conformation of the cities has been completely altered. A first phase occurred in which besides all the industrial areas, grown outside the city walls, new poor and unhealthy houses were built for the factory workers.

After this first period of expansion, architects developed expansion plans devoted to rebuilt all the new quarters in a more healthy and efficient way. New residential areas were built outside the city walls.

The urban population continued to increase, residential quarters were built outside the city walls. Besides, the growth of the first fast means of transport, such as the automobile, created different needs of transport and therefore the new quarters had wider roads. In several cities, the walls were demolished to make place for the ring roads, and also in the historical city centres old houses were demolished to make place for new wide roads. In Paris, Napoleon III demolished several old parts of the city and replaced the narrow and often unhealthy roads with wide *boulevards*. Nowadays much historical and artistic patrimony is lost because of this “wild” demolition.

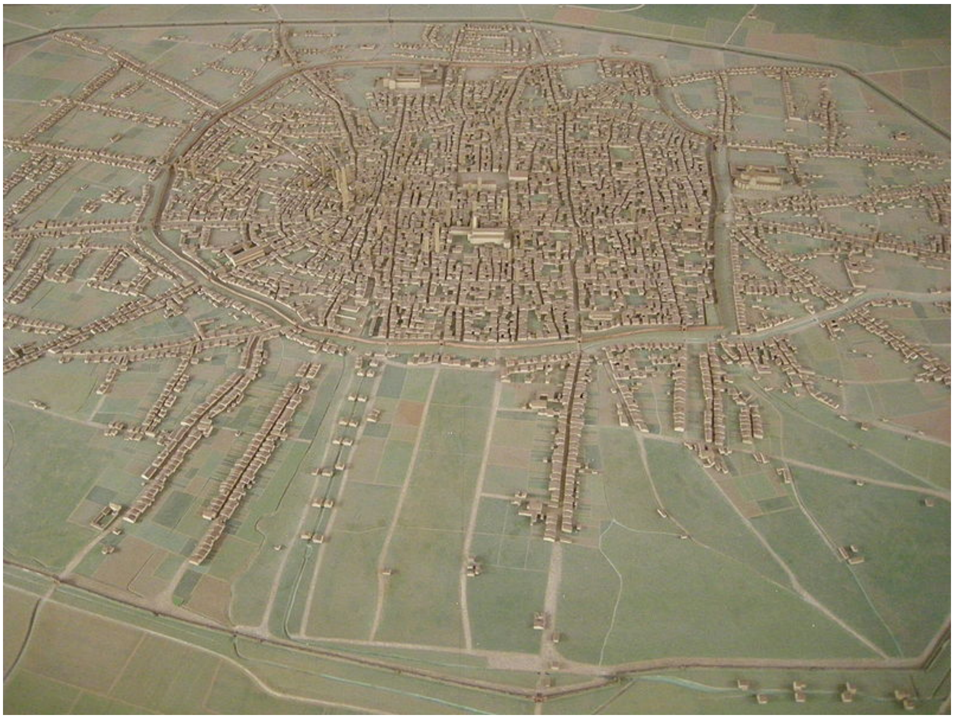


Fig. 1.6. A plastic, visible in the Civic Medieval museum of Bologna, which represents the development of the city in the Early Renaissance: the inner part of the city is inside the first city walls. As all the space inside was consumed (the free spaces refer to private gardens) some new quarters have been built outside the walls, along the main roads. A moat and an embankment are visible outside the urbanized area, in the places in which the new walls will be built.

This demolition however luckily regarded only some parts of historical city centres, while the rest of the city centre still has the original buildings, and the original roads, which are very narrow. Moreover, this demolition happened in several cities, such as London and Paris, but in several small cities and villages, some of which are located in central and southern Italy, the historical city centre is still intact. And the roads in historical city centres, for the reasons exposed above, are much narrow, their width is sometimes less than 2 metres, and have uneven pavements, like cobbled stones. Moreover some Italian cities grew on the hills, therefore roads are often characterized of high slopes (sometimes also 7–10% as registered in the historical centre of Genoa) and steps. Traditional public transport therefore cannot operate or operates with difficulty in such environments. On the other hand, city centres are characterized of high demand of transport, because: offices, activities and schools are concentrated there; and several tourist attractions, such as museums and monuments, are present. Moreover monuments and old houses can be seriously damaged by pollution and by the vibration created by heavy road traffic (like long buses). Therefore, new and greener modes of transport must be designed, in order: on one hand to satisfy the transport demand, and on the other hand to cope with all these infrastructural and physical limitations.

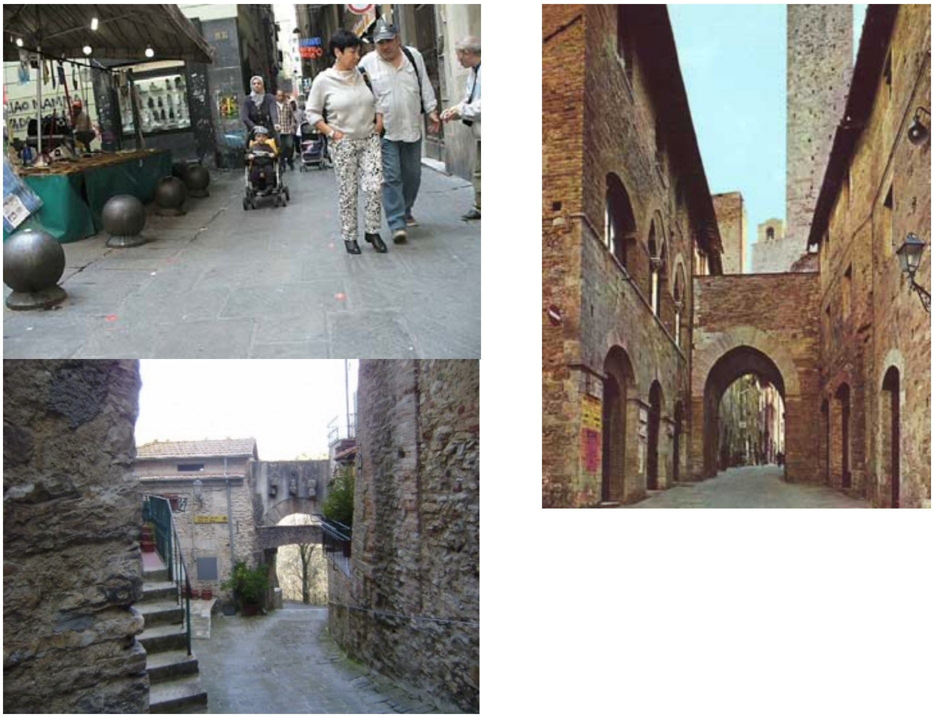


Fig. 1.7a-b-c. Roads and squares in historical city centres in Italy.

1.2. Current trends.

1.2.1. Massive growth of urbanization

The level of urbanization will grow dramatically in the next years. From a research performed by Yale University, in the next 18 years urbanized areas are expected to grow by 1.8 millions of km^2, and the cost of the necessary new infrastructures will be about 30,000 billions of dollars. This phenomenon will involve all continents; in Africa, the urbanized surface will be 600% more than in year 2000. In Italy, the areas mostly involved in the urbanization will be the Po Valley, Rome and Napoli.

1.2.2. The ageing of society.

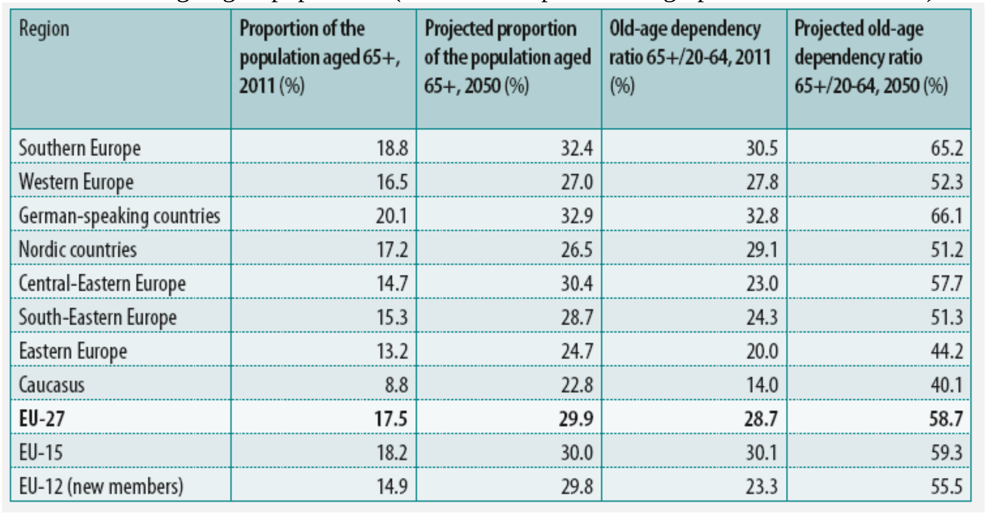
The mobility is equal to the distance that people can reach in a given time. The accessibility is equal to the opportunities which can be reached by people in a given time. The accessibility can be improved through an improvement of the mobility.

The number of aged people or of people with some impairment is high and it is growing with time. As a result, all future transport systems will have more and more to ensure accessibility to everybody, with particular attention to people's impairment. Several European projects, such as Ask-IT (www.ask-it.org), MEDIATE (www.mediate-project.eu), Access2All (www.access-to-all.eu), AENEAS (www.aeneas-project.eu), have been developed in order to understand the needs of all categories of users. In Italy in year 2004, 2,600,000 people older than 6 years suffered from disability, which accounted for 4.8% of the overall Italian population. About 46% of people with impairments have more than 80 years, and about 80% of people with impairments is older than 65 years.

Data about people's impairment have been collected from the European Demographic Data Sheet 2012. Three categories of impairments have been considered:

- difficulty in movement
- difficulty in everyday life functions, e.g. self care
- difficulty in communication, i.e. sight, hearing, speaking.

Table 1.4. The ageing of population (Source: European Demographic Data Sheet 2012).



Region	Proportion of the population aged 65+, 2011 (%)	Projected proportion of the population aged 65+, 2050 (%)	Old-age dependency ratio 65+/20-64, 2011 (%)	Projected old-age dependency ratio 65+/20-64, 2050 (%)
Southern Europe	18.8	32.4	30.5	65.2
Western Europe	16.5	27.0	27.8	52.3
German-speaking countries	20.1	32.9	32.8	66.1
Nordic countries	17.2	26.5	29.1	51.2
Central-Eastern Europe	14.7	30.4	23.0	57.7
South-Eastern Europe	15.3	28.7	24.3	51.3
Eastern Europe	13.2	24.7	20.0	44.2
Caucasus	8.8	22.8	14.0	40.1
EU-27	**17.5**	**29.9**	**28.7**	**58.7**
EU-15	18.2	30.0	30.1	59.3
EU-12 (new members)	14.9	29.8	23.3	55.5

About 700,000 people have difficulties in movement, which is equal to 1.3% of people older than 6 years. About 376,000 people, equal to 0.7% of people older than 6 years, have serious difficulties in performing self care. People with impairment related to communication, i.e. sight and hearing, are to 217,000, equal to 0.4%. About 1,200,000 people have disabilities in at least two of the three typologies exposed above, which accounts for 1.9% of the overall European population. Moreover, about 0.5% of people older than 6 years suffer from all the three categories of disabilities. On the whole, about 44% of people older than 80 years suffer from some form of disability.

The population is ageing, specially in Europe, as displayed in table 1.4. (Source: European Demographic Data Sheet 2012). Currently, in the European Union, about 17.5% of the population is older than 65 years but this percentage is expected to increase to about 30% in 2050. In tables 1.4 to 1.6 and in figure 1.8, some data about the ageing of the population are provided.

Table 1.5. Median age of the population (Source: European Demographic Data Sheet 2012)

Rank	Population median age, 2011 (years)		Rank	Projected population median age, 2050 (years)	
1	Germany	44.6		*Japan*	*56.0*
	Japan	*44.3*	*1*	Romania	53.9
2	Italy	43.5	2	Croatia	53.0
3	Greece	42.2	3	Albania	52.1
4	Finland	42.1	4	Latvia	52.0
5	Austria	42.0	5	Poland	51.7
	EU-27	**41.3**		**EU-27**	**48.0**
34	Macedonia. FYR	36.1	34	Sweden	43.7
35	Ireland	34.7	35	Turkey	43.3
36	Moldova	34.2	36	Ireland	43.2
37	Albania	31.0	37	United Kingdom	42.6
38	Turkey	29.3	38	Cyprus	40.1
				USA	*38.0*

Table 1.6. Old age dependency ratio. (Source: European Demographic Data Sheet 2012)

Rank	Old-age dependecy ratio, 2011 (years)		Rank	Projected old-age dependecy ratio, 2050 (years)	
	Japan	*39.5*		*Japan*	*81.1*
1	Germany	33.8	1	Italy	67.9
2	Italy	33.3	2	Germany	67.7
3	Sweden	31.6	3	Greece	67.3
4	Greece	31.4	4	Portugal	66.7
5	Portugal	29.5	5	Slovenia	66.0
	EU-27	**28.6**		**EU-27**	**58.7**
34	Russia	18.9	34	Lithuania	45.8
35	Slovakia	18.8	35	Cyprus	44.2
36	Macedonia. FYR	18.5	36	Russia	41.6
37	Moldova	15.2	37	Moldova	40.7
38	Turkey	12.3	38	Turkey	38.7
				USA	*37.3*

With reference to tables 1.4 – 1.6, the old-age dependency ratio is defined as the number of people aged 65 or older to the number of people aged 20 to 64:

$$\text{old age dependency ratio} = \frac{\text{number of people aged 65 or older}}{\text{number of people aged 20 to 64}}$$

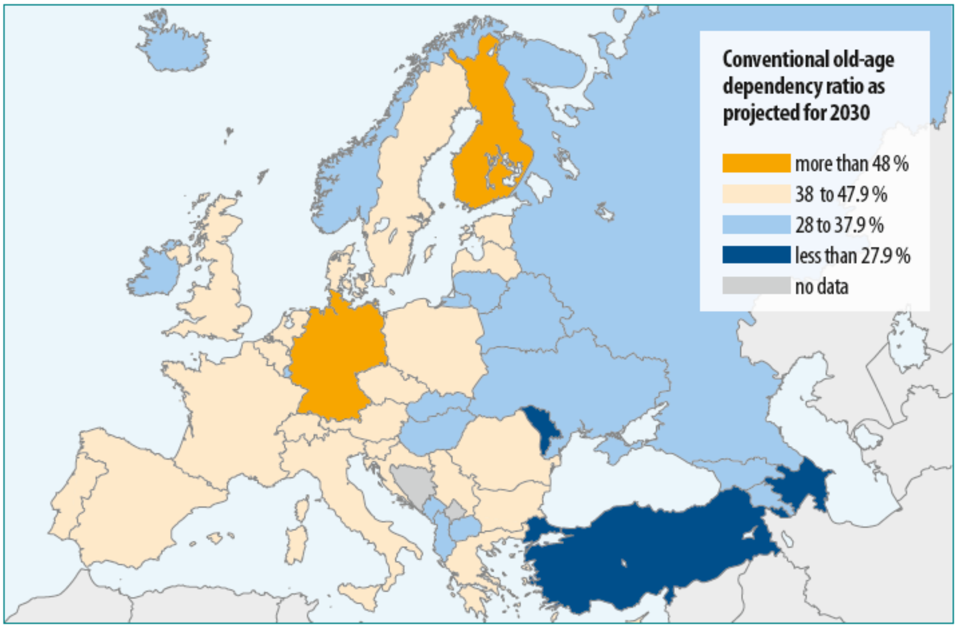


Fig. 1.8. Projection of the old-age dependency ratio for year 2030 in Europe. (Source: European Demographic Data Sheet 2012).

1.3. Policies and strategies to improve sustainable mobility in cities.

Several policies are currently being adopted by local administrations, in order to reduce the degree of congestion and pollution in urban areas. These measures regard: limit for the emission level of vehicles, measures to discourage private transport modes, improvement of conventional public transport, settlement of innovative public transport modes.

1.3.1. Policies aimed to reduce the emissions of vehicles

Several measures have been taken by local administrations in order to reduce the emission level of vehicles. Vehicles have been classified according to it. For vehicles running with petrol, the following classifications exist: non catalysed (pre Euro), Euro 0, Euro 1, Euro2, Euro3, Euro4, Euro5. Local administrations have prohibited the circulation for the majority of the day time to some classes of vehicles, such as pre Euro and Euro 0. When the concentration limits of PM10 are reached, local administration establish days of complete stop to private vehicles circulation, except for Euro4 and Euro5 petrol cars, vehicles running with gas, Diesel vehicles with anti-particulate filter. There is however great debate as the anti particulate filter of Diesel vehicles is effective for the biggest among PM10 particles; PM2.5 particles are not blocked by ant particulate filter, and these has shown to be the most dangerous for citizens' health.

1.3.2. Measures aimed to discourage private transport

Drivers are often not aware of the higher cost of private transport with respect to public transport: a car costs not only in terms of fuel, but also tyres, brakes, mechanical parts to replace, insurance, etc. Moreover they assign a high cost to the disutility of public transport, which for its nature shows little flexibility. The result is that in small and medium-sized Italian cities only a small percentage of commuters perform their trips through public transport. The low level of demand forces public administrations to reduce the offer of public transport. This decreases the quality of public transport, which again results in a further decrease of transport demand. Therefore, in order to shift a consistent level of demand to public transport, it is necessary both to improve public transport and to discourage the usage of private car. Several measures have been adopted by local administrations with this aim. These measures are normative, i.e. restrictions to private car in some zones of the city, or aimed to increase the cost of private transport, such as road pricing and park pricing.

1.3.2.1. Restriction of private transport in some zones of the cities

These measures consist on the establishment of the so-called "restricted traffic zones" and of pedestrian areas. Restricted traffic zones are areas restricted to private car, except for residents. In all the major Italian cities, the greatest part of the historical city centre is a restricted traffic zone. Pedestrian areas instead are zones restricted to everyone, except people having special permissions, and usually coincide with the most narrow roads with highest pedestrian flows, and with the most inner part of historical city centres.

Besides, some major roads of city centres and of the closest suburbs are completely dedicated to public transport, or have dedicated lanes. This policy is taken in order to decrease the travel time of buses with respect to private car.

1.3.2.2. Road pricing, park pricing

The road pricing measure limits the possibility of entering in some zones of a city provided of the payment of a fee. London city centre, for example, is subjected to road pricing.

A measure similar to road pricing is the park pricing: parking is charged and the amount of charge depends on the location of the car park. Some zones such as historical city centres, the closest suburbs, the areas in proximity of the main railway stations are usually the most expensive.

Usually the revenues from road pricing and from parking fees are used by public administrations to improve the quality of public transport.

The road pricing zone in London is shown in the figure 1.9.

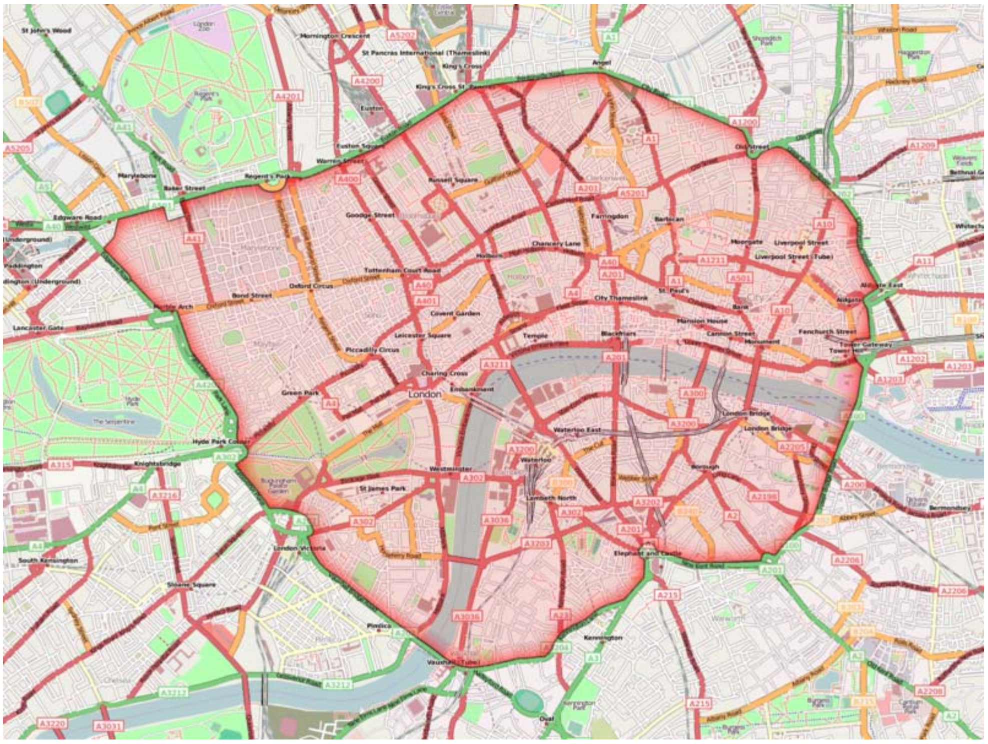


Fig. 1.9. London road pricing zone. (Source: Transport For London).

1.3.3. Improvement of public transport.

The measures aimed to discourage the usage of private car are not effective if a high quality public transport is not provided to users. These measures therefore may result only in the decrease of the transport demand and on the settlement of some activities and residents in other areas, and not in a real shift from private to public transport.

In order to perform a high quality public transport, all the segments of demand must be satisfied, specially people living in far and low populated residential areas and people aged or with some disability.

1.3.3.1. Improvement of conventional public transport.

In order to better capture percentages of transport demand, conventional public transport must be improved. Public transport is of high quality for users if it is:

- fast, i.e. transit times should be kept low;
- reliable, i.e. it should cross the stops and stations at the scheduled time. Reliability assume extreme importance when user trips involve different

means of transport: for example they have to take different bus lines, or tranship from train to bus or from bus to underground;

- capillar, i.e. it should reduce walking distances and therefore be capable to board the user as close as possible to his origin, and bring him as close as possible to his destination;
- frequent, at least 4 services by hour for urban areas; however if frequencies overcome 12 services by hour, there is no longer a benefit to users;
- with a reduced number of transhipments: much time is lost when users commute between different means of transport; moreover time spent waiting is perceived in a worse manner than time spent on board.

High quality of transport can be achieved through several measures; for example:

- build new lines of underground or improve the existing lines, when required
- develop bus lanes in some major roads interested with high traffic flows and provide public transport a priority phase in signalized intersection, in order to decrease the journey time. This also result in a decrease of management costs as fewer vehicles are necessary to perform the service
- in small and medium cities, where the demand is not enough for building an underground or other fast transit solutions, rapid bus lines, with only few stops, should developed.
- Build parking spaces in peripheral areas in proximity of public transport stops. Therefore commuters reach the urban area by private car and after commute to public transport.

Some small villages located in proximity of cities, or in isolated and low populated residential areas, where also the demand of transport is low, cannot be served by a high quality and frequent transport line. Therefore, for such situations on-demand services must be developed. In several small cities, some public transport lines have a main path and several areas around to serve on-demand. These lines have a high frequency, and deviate from the main path as they receive a request.

1.3.3.2. Car sharing.

Car sharing is a model of car rental where people rent cars for short periods of time, often by the hour. They are attractive to customers who make only occasional use of a vehicle, as well as others who would like occasional access to a vehicle of a different type than they use day-to-day. The organization renting the cars may be a commercial business or the users may be organized as a democratically controlled company, public agency, cooperative, ad hoc grouping. Today car sharing services are widely spread worldwide, with over a thousand cities globally where car sharing services are available.

The principle of car sharing is that individuals gain the benefits of private cars without the costs and responsibilities of ownership (Shaheen et al., 1998). Instead a household accesses a fleet of vehicles on an as-needed basis.

With reference to the usage of private car, car sharing has the following advantages (Shaheen et al., 1998):

1. Cost effectiveness. Car sharing is generally not cost-effective for commuting to a full-time job on a regular basis. Most car sharing advocates, operators and cooperating public agencies believe that those who do not drive daily or who drive less than 10,000 kilometers annually may find car sharing to be more cost-effective than car ownership (CityCarShare, 2007).
2. More environment friendly. Car sharing can help reduce congestion and pollution. Replacing private automobiles with shared ones directly reduces demand for parking spaces. The fact that only a certain number of cars can be in use at any one time may reduce traffic congestion at peak times. Moreover, often car sharing systems involve electric vehicles or vehicles running with gas; and shared vehicles are usually smaller than privately owned vehicles (Shaheen et al., 2009).

With reference to conventional public transport, car sharing shows the following advantages:

1. Flexibility. Users can reduce the distances to access the system and do not have to change different means of transport to reach their destination. Moreover, car sharing is ideal for leisure trips as the driver can decide where to go on his own.
2. Privacy. The user can travel alone or decide the travel mate, similarly to private car.
3. Better accessibility of low populated areas.
4. Car sharing provides better service than conventional public transport: for aged people or for people with some disability. In 2008 CityCarShare introduced the first wheelchair carrying car share vehicle, the Access Mobile, specifically designed as a fleet vehicle shared with wheelchair users.

Several cities have organized car sharing services at railway stations. This can be very useful as visitors do not have to learn the location of public transport lines and stops, but they simply reach their destination by taking the shared vehicle (www.icscarsharing.it).

The main problem related to car sharing regards the unbalancement of vehicles among the various locations. Indeed, it may easily occur the situation where in some stations there is a large number of vehicles available, while in others there are several users in queue. This problem, together with all the rebalancing techniques, will be treated in detail in the following sections.

1.3.3.3. Bike sharing.

A bike-sharing system is a service in which bicycles are made available for shared use to individuals who do not own them. Bicycle sharing systems are aimed to provide free or affordable access to bicycles for short-distance trips in an urban area as an alternative to motorized public transport or private vehicles, thereby reducing traffic congestion, noise, and air pollution. Bicycle sharing systems have also been cited as a way to solve the “last mile” problem and connect users to public transit networks (Shaheen, Guzman, Zhang, 2010).

Bike sharing systems can be divided into two general categories: public bike sharing programmes organized mostly by local community groups or non-profit organizations; and government-run bike sharing programmes implemented by governments, sometimes in a public-private partnership.

Public bike sharing programmes address some of the primary disadvantages to bicycle ownership, including loss from theft or vandalism, lack of parking or storage, and maintenance requirements. However, by limiting the number of places where bicycles can be rented or returned, the service itself essentially becomes a form of public transit, and has therefore been criticised as less convenient than a privately owned bicycle capable of point-to-point transport. Government-run bicycle sharing programmes can also prove costly to the public unless subsidised by commercial interests, typically in the form of advertising on stations or the bicycles themselves (Shaheen, Guzman, Zhang, 2010).

Bike-sharing systems have undergone changes which can be categorized into three key phases, or generations. These include:

- the first generation, called white bikes (or free bikes): free of charge;
- the second generation of coin-deposit systems, which is a simple bike rental;
- and the third generation, or information technology based systems.

Recent technological and operational improvements are also paving the way for a fourth generation, known as the demand-responsive, multimodal system.

As of May 2011 there were around 136 bike-sharing programmes to be launched in 165 cities around the world, made of an estimated fleet of 237,000 bicycles. The countries with the most systems are France (29), Spain (25), China (19), Italy (19), and Germany (5). (Shaheen and Guzman, 2011).

Further details about bike sharing systems will be provided in chapter 2.

1.4. The PICAV transport system.

It is in this scenario, that the proposed transport system is developed. This system has been developed with the aim to provide accessibility for all, especially for people with some mobility impairment, to the urban historical city centre, in particular in those areas where traditional public transport cannot operate. The proposed transport system is an on-demand system and it is based on the car-sharing concept.

The proposed transport system makes use of small personal electric vehicles, called PICAV, i.e. personal intelligent city accessible vehicles. They are one-person vehicles and are specifically designed to occupy as less space as possible but without compromising mechanical stability. Their inner part, the seating system and the controls to motion have been designed to cope with all people’s impairments. The vehicles’ dimension is about 1.1 m wide and 1.3 m long (Masood et al., 2012). A picture of the PICAV vehicle is provided in fig. 1.10 and 1.11. These vehicles have been designed for accessing any typology of environment, such as small roads with high pedestrian flows, uneven pavements and some indoor parts of buildings. They are able to swall steps and to adapt to various typology of slopes, both longitudinal and lateral.

The proposed transport system is planned to work in pedestrian-only environments and to easily interface with conventional public transport and private car. The stacks of available PICAV vehicles will be conveniently placed in locations all around pedestrian areas close to interchange points with public transportation, like bus and underground stops, railway stations, etc.; and with private car parks.

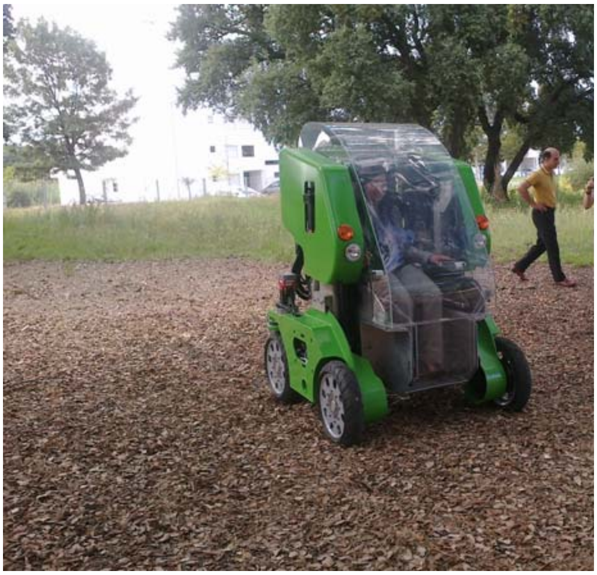


Fig. 1.10. The PICAV vehicle

1.4.1. The PICAV vehicle.

The PICAV vehicle is an electrically powered, automatic, one person vehicle, which ensures accessibility for everybody and some of its features are specifically designed for people whose mobility is restricted for different reasons, particularly (but not only) elderly and disabled people.

PICAV vehicles are specifically designed for urban pedestrian environments where usual public transport services cannot operate because of the width and slope of the infrastructures, uneven pavements and the interactions with high pedestrian flows.

The prototype of the vehicle has been developed in the European project PICAV, Personal Intelligent City Accessible Vehicle, FP7-SST-2007-RTD-1. Grant agreement no.: SCPS-GA -2009-233776. Starting date: 1st August 2009. End date: 30th September 2012. (http://www.dimec.unige.it/pmar/picav/). Partners involved:

- DIME (Coordinator) (Department of Mechanics and Machine Design), University of Genoa, Italy
- INRIA (National Institute for Research in Computer Science and Control), France.
- UCL (University College London), UK.
- University of Pisa, Italy
- TCB (Serviços Municipalizados de Transportes Colectivos do Barreiro), Portugal
- ZTS VVU KOSICE, Slovakia
- Mazel Ingenieros, Spain.

The PICAV vehicle shows the following specific characteristics (DIME, 2012):

- Size and Weight: all the dimensions of PICAV vehicle prototype were kept as small as possible (1.1 m wide and 1.3 m long) but compatible with ergonomic user needs and environmental infrastructure. The partnership tried to keep the vehicle mass as low as possible.
- Agility: dexterous operation in narrow roads and short corners; ability to overcome steps up to 200 mm; ascend and descend a maximum longitudinal incline of 25 degrees; a maximum tilt of 25 degrees without toppling; turning radius smaller than 1m.
- Zero emission of pollutants.
- Noise emission less than 45 dBA (for comparison, the legal maximum noise level for a small motorcycle is 77 dBA and 45 dBA is about the level of a quiet urban area without traffic of any kind and with few pedestrians walking around).
- Personal comfort and Ergonomics: the vehicle components and sub-assemblies were tested by representative sample of potential users. They were be asked to sit in the vehicle, to drive it in a given scenario and to give feedback. The level of satisfaction about comfort and easy driving also in transient conditions (breaking, turning, acceleration) was good.
- Safety: the vehicle maximum velocity is lower than 6 m/s thus no crash tests are foreseen. Stability is an important issue for safety. With reference to the agility performances, the vehicle guarantees roll and tilt in an angular range of ±12 degrees. It was verified that the user’s exit from the vehicle is possible in case of overturning in any direction. Safety issues were addressed considering robustness of the control and redundancy of sensing, actuation and control systems. The design of the vehicle exterior ensured that the surfaces are smooth with no protuberances that could injure passing pedestrians.
- Energy efficiency: The power system architecture and specific algorithms were developed to improve the vehicle energy efficiency.

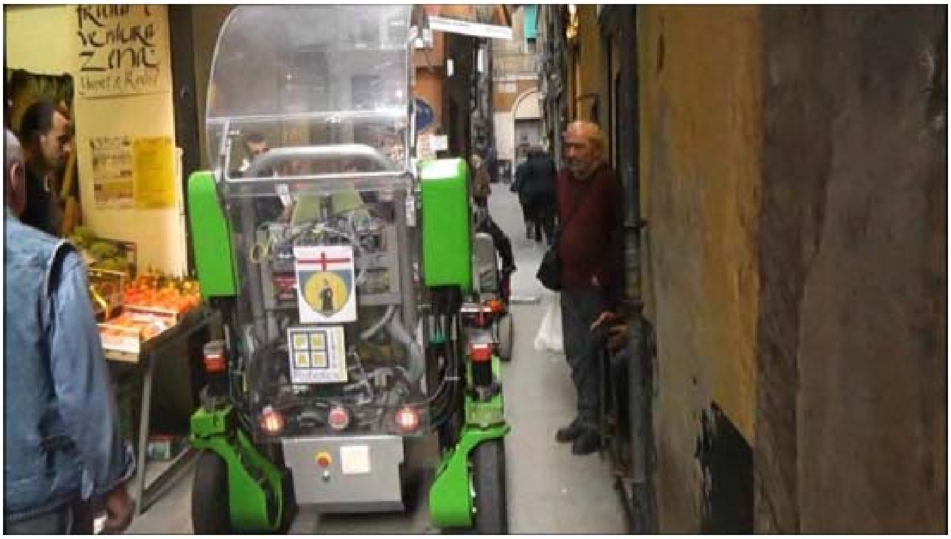


Fig. 1.11. The PICAV vehicle in a road in Genoa centre (DIME, 2012)

1.4.1.1. The power system

The battery system is made up of Lithium-polymer cells. Each cell has a nominal voltage of 3.2V and its capacity is 5Ah. The nominal current is 5A. The battery pack configuration is made by 15 blocks connected in serial and each block has 16 cells connected in parallel. When the battery is fully charged, the voltage is 54V. Nominal voltage is 48V and cut off voltage is 45V. We could storage all the energy in a volume of 20 litres with a weight of 40Kg. An internal view of the PICAV battery is shown in the figure 1.12.

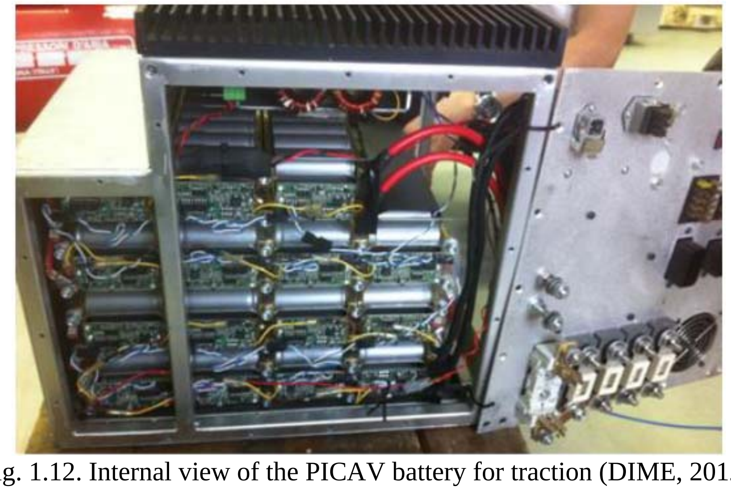


Fig. 1.12. Internal view of the PICAV battery for traction (DIME, 2012).

1.4.1.2. The traction and the roll control mechanism

The traction of the vehicle is made by actuators directly attached to the wheels, similarly to the new in-wheel motors available in the market. The vehicle has 4 driving wheels designed in such a way to allow the vehicle to climb steps.

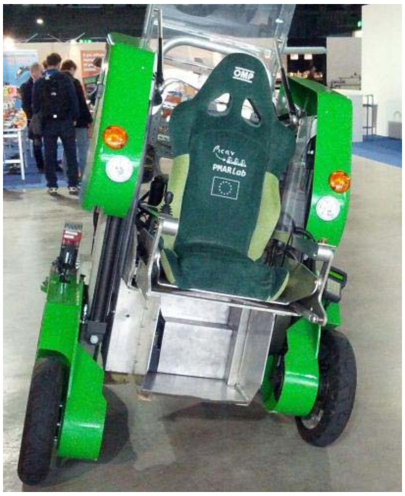


Fig. 1.13. The capability of the vehicle's wheels to allow the vehicle travel with two wheels in the sidewalk and two wheels on the road. Thanks to that, the user can be kept in a horizontal position (DIME, 2012).

The roll control mechanism of the PICAV vehicle is designed in a way so that the vehicle is capable of adjusting the height of the wheels module with respect to the terrain. Taking into account the urban scenarios in which part of the city is atop a hill, it has to be considered that the PICAV will circulate in roads with a lateral incline. The roll mechanism permits that with a lateral incline the body of the vehicle maintains horizontality and it becomes virtually impossible to turn over. Furthermore, the vehicle needs to be capable of circulating in different environments including pedestrian areas. PICAV is capable of going with one wheel module above the sidewalk and the other below on the road, and of maintaining at the same time the driver in a horizontal position, as shown in figure 1.13.

1.4.1.3. The seat

The seat module has been designed taking into account the state of the art on ergonomics of automotive seats and the needs of impaired user. It envelopes the user at the shoulders and legs; while the size ensures comfort for users of any height and size.

The motion of the seat is delivered by two actuators. A first linear actuator enables the forward movement of the seat inside the vehicle. The second one starts to work when the seat is completely out of the vehicle and raises the seat. When the user is leaning in the seat, the procedure is reversed accompanying the user in the inner drive position. This gives the possibility to mobility impaired users to be carried inside the vehicle in an autonomous way without having to use their muscles. The actuation of this motion is commanded by the user by the buttons located in the armrest. Figure 1.14 shows the functionality of the seat

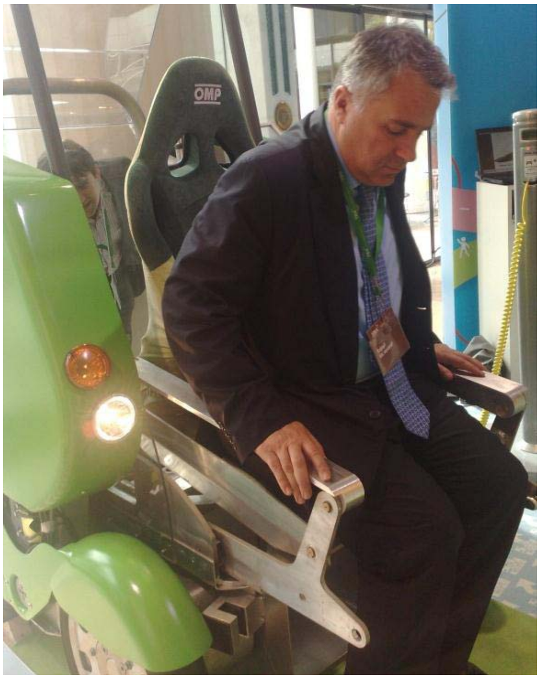


Fig. 1.14. The functionality of the seat. It accompanies the user when he raises and when he sits down, allowing mobility impaired people to sit and raise without using their muscles (DIME, 2012).

1.4.1.4. The sensors

Different sensors are used to provide driving assistance and automated drive. Three front laser scanners are used for the obstacle, steps and stair detection. Moreover, 4 ultrasonic sensors and a vision camera were used for the rear detection in the back driving maneuvers. Two inclinometers were used to keep the comfort in the driving. The stair detection system is composed of three laser sensors. The two laser sensors dedicated to the step detection have a vertical orientation in order to scan the altitude profile of the environment. For the obstacle detection, another was placed in the middle of the vehicle in horizontal position. The system has been designed as the vehicle is supposed to be able to cross over a stair and steps. The sensorial system installed on PICAV is shown in figure 1.15. The PICAV perception system is shown in figure 1.16.

1.4.1.5. The human machine interface

The human machine interface (HMI) is in charge of transmitting the command from the user to the vehicle. The HMI developed allows three main tasks: joystick, buttons and monitoring. The software is installed on an onboard tablet. The onboard tablet has installed a wifi-card, and through this connection the command can be read by other computers: in fact, the HMI can run in different external computers that have access. However, the simultaneous use of each task is prohibited

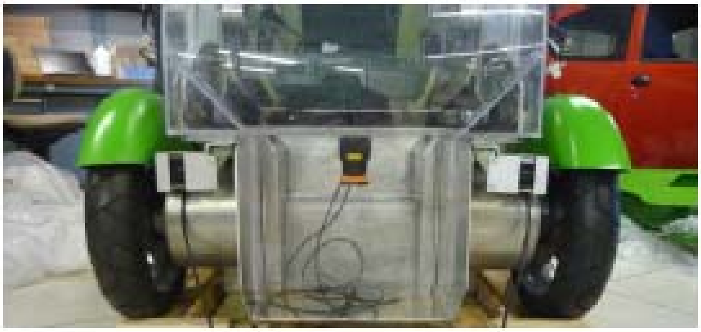


Fig. 1.15. The sensorial system installed on the PICAV (DIME, 2012).

Joystick. A touchscreen of the smartphones has been used to control the steering and the speed on the PICAV. However, from a standard computer (with a mouse) the joystick can also works properly. A low past filter is added in order to avoid jumps or peaks to the PICAV actuators. When the user lifts the finger from the touchscreen of the HMI, the filter smooths the signals.

Buttons. Figure 1.17 shows the six buttons used by the HMI. The different tasks defined are: chassis (up and down); cover (open and close); seat (forward and backward); seat (up and down); lights (on and off).

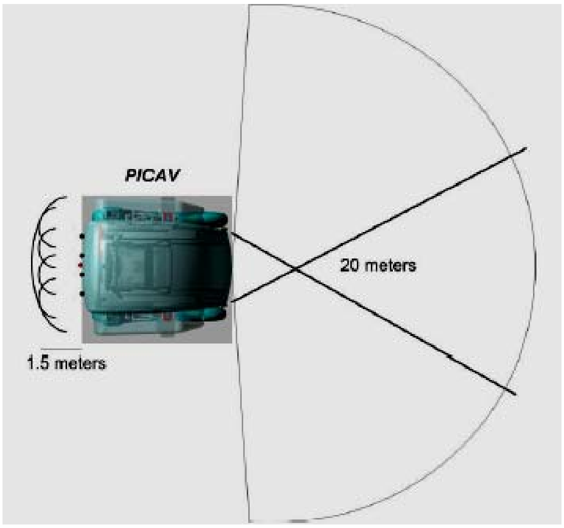


Fig. 1.16. The PICAV perception system through the sensors mounted on the vehicle (DIME, 2012).

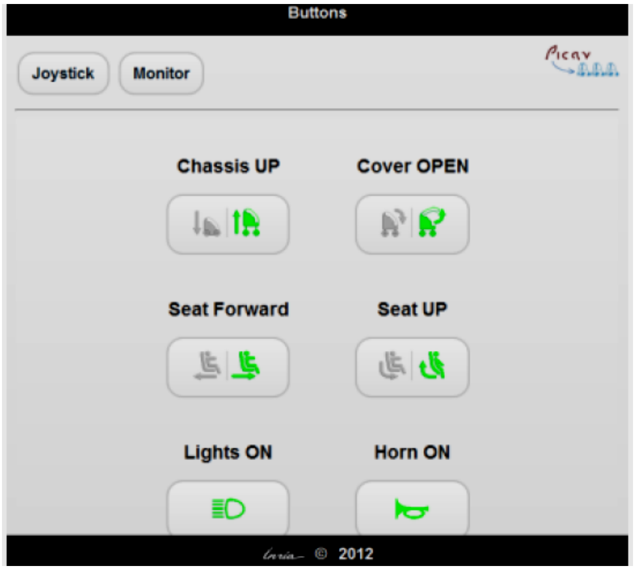


Fig. 1.17. The buttons of the human machine interface (DIME, 2012).

Chapter 2. State of the art on vehicle sharing systems

Introduction

In this chapter, the main aspects about car and bike sharing systems and their evolution are provided. Car and bike sharing systems have the advantage of providing users a high level of flexibility, which aims to be as similar as possible to privately-owned vehicles. On the other hand, the necessity to rebalance vehicles among stations arises. This is a key aspect for what regards vehicle sharing systems and in this chapter the focus is provided to that.

There are however other aspects related to car and bike sharing systems, that are not treated in this chapter. These aspects regard:

- membership and registration;
- booking techniques: for example online booking;
- check in and check out methodologies: for example through smart card;
- techniques for the identification of the users;
- techniques for payment;
- security for vehicles robbery;
- pricing systems and integration with pricing of conventional public transport;
- system management;
- techniques for the localization of vehicles;
- techniques for communication between vehicles, users and system manager.

The chapter is structured in the following way.

Section 1 provides an overview of car sharing systems and of their rebalancement techniques. The first subsection explains the main characteristics of car sharing systems. Car sharing systems can be classified in three generations according to the level of flexibility provided to users. Subsection 2 concerns the origin of car sharing systems and a general introduction of their main characteristics. Subsections 3, 4 and 5 show the main features and problems of the three generations of car sharing systems.

Section 2, describes the main features of new generation bike sharing systems and their main characteristics. Relocation for bike sharing systems does not constitute a problem because cycles can be successfully carried on vans in great quantity and therefore staff costs are kept low.

2.1. Car sharing systems

2.1.1. Overview

Car sharing is a model of car rental where people rent cars for short periods of time. "Carpooling" or "ride sharing" refers to the shared use of a car for a specific journey, in particular for commuting to work, often by people who each have a car but travel together to save costs. The principle of car sharing is that individuals gain the benefits of private cars without the costs and responsibilities of ownership (Shaheen et al. 1998). Instead a household accesses a fleet of vehicles on an as-needed basis. Car sharing may be thought of as organized short-term car rental. Car sharing has sprung up in different parts of the world and operations are organized in many different ways in different places. Sizes of organizations vary, from one shared car and only a handful of sharers, to organizations that serve a complete urban area.

Car sharing differs from traditional car rentals in the following ways (Shaheen et al., 1998):

- Car sharing is not limited by office hours.
- Reservation, pickup, and return is all self-service.
- Vehicles can be rented by the minute, by the hour, as well as by the day.
- Users are members and have been pre-approved to drive (background driving checks have been performed and a payment mechanism has been established).
- Vehicle locations are distributed throughout the service area, and often located for access by public transport.
- Fuel costs are included in the rates.
- Vehicles are not serviced (cleaning, fuelling) after each use, although certain programs such as Car2Go continuously clean and fuel their fleet

The technology of CSOs varies enormously, from simple manual systems using key boxes and log books to increasingly complex computer-based systems (e.g. partially automated and fully automated systems) with supporting software packages that handle a growing array of back office functions (Shaheen et al., 2005). The simplest CSOs have only one or two pick-up points, but more advanced systems allow cars to be picked up and dropped off at any available public parking space within a designated operating area.

While differing markedly in their objectives, size, business models, levels of ambition, technology and target markets, these programs share many features. The more established operations usually require a check of past driving records and a monthly or annual fee in order to become a member. The cost and maximum time a car may be used also varies (Shaheen et al., 2005).

To make a reservation, one can either make a reservation online, by phone, or by text messages depending on the company's flexibility. Then the company usually asks all the necessary information such as: the beginning time of the trip, the end time of the trip, the best place for picking up the car, which car is preferred.

There is a small card reader mounted on the windshield. Once the customers place their membership card on the reader, it will use what is called blink technology to activate the time and unlock the car. The reader will not work until it is time for that specific reservation. Depending on the company, the customer may be provided with a key to a lock box that contains the ignition key itself. Once the customer is set, they are off to their next destination.

Although members are often responsible for cleaning the car and filling up the tank when low, the car sharing company is generally responsible for the long-term maintenance of the vehicles. Members have to make sure that when they are finished, the car is ready for the next user.

Many car sharing companies, are now investing in plug-in hybrid electric vehicles (PHEV) and in fully electric vehicles.

2.1.2. History

The earliest origin of shared use vehicle systems is in 1948, in Zurich, which has been performed by a cooperative called "Sefage", but there was no known formal development of the concept in the next few years. By the 1960s as innovators, industrialists, cities, and public authorities studied the possibility of high-technology transportation, mainly computer-based small vehicle systems (almost all of them on separate guideways), it was possible to spot some early precursors to present-day service ideas and control technologies (Shaheen et al., 1998).

The early 1970s saw the first whole-system car share projects. A shared vehicle system, ProcoTip, has been settled in 1971 in Montpellier. The ProcoTip system lasted only about two years. A much more ambitious project called the Witkar was launched in Amsterdam in 1973. A sophisticated project based on small electric vehicles, electronic controls for reservations and return, and plans for a large number of stations covering the entire city, the project endured into the mid-1980s before finally being abandoned (Shaheen et al., 1998). All these experiences therefore have been unsuccessful.

However, in the 1980s several other initiatives have been launched, such as "Mobility Car-Sharing" in Switzerland, and "Stattauto" in Berlin. In these car sharing system usually members need to book cars beforehand and the time the car will be dropped off should be specified (fixed-period reservation); besides, generally cars must be returned to the same location where they were picked up (two-way trips). In 1990s, the car-sharing concept has become popular also in the U.S., where several pilot projects have been performed.

After, to overcome the barriers of traditional car-sharing systems, a new generation of car-sharing systems has been developed which provides the following specific services: instant access; open-ended reservation and one-way trips. Among these new systems, UCR IntelliShare at the University of California at Riverside (Barth and Todd , 2003; Barth at al., 2000), CarLinkI and CarLink II in Dublin-Pleasanton (Shaheen and Rodier, 2005); in Paris (Autolib'); and in Singapore (Honda ICVS). The features provided by the new generation systems on one hand provide users great flexibility, but on the other hand create a serious problem, which is the unbalancement of vehicles available at the various stations, therefore in some stations there is an excess of vehicles, while in others there is lack of vehicles.

The target of a relocation strategy is twofold: firstly to reduce management costs and secondly to provide users high flexibility and low waiting times.

Relocation strategies could be classified in two main categories: operator-based and user-based. Operator-based strategies resolve the balancement problem by operators that manually relocate a vehicle or a platoon of vehicles from stations having vehicles in excess to stations having lack of vehicles. In user based strategies, instead, relocations are performed by the trips of the transport system users. In this thesis, a new strategy in which some users accept to be told to which stations return the vehicle, according to the system needs, is shown.

The activities performed by the operators are: maintenance: the staff is performing vehicles maintenance such as refueling, cleaning, etc.; movement: the staff moves from the station in which is currently present to the station in which he is needed to perform the relocation operations; and relocation: the staff relocates the vehicle. The great majority of shared vehicle systems involve operator-based strategies. This category includes also some strategies which could be also partially user-based: some users may be available in performing few of the required relocations if motivated by a reduction in the transport price. Only a few partially user-based strategies have been developed. In IntelliShare the integration of operator-based strategies and partially user-based strategies has been considered (Barth and Todd, 2003). In the strategies belonging to the second category, all the relocations are performed by users and therefore operator’s costs could be saved.

Recently, some “third generation” car sharing systems, such as Car2Go (Firnkorn and Müller, 2011) and DriveNow, have been developed. In these systems, vehicles can be accessed also along the roads and be returned to the system at any point of the intervention area. They have been specifically designed for those residential areas where the population is low and sparse and therefore users may have a quite long walking distance to reach the closest station.

The performance of a relocation strategy is generally assessed as a function of users waiting times and number of relocations. Several authors, such as Shaheen et al (2009), focus on modal split issues: the capability of these new transport systems to attract users from private transport modes.

2.1.3. First generation car sharing systems

These car sharing systems, which are called *first generation*, require registration and *booking* in advance. Moreover, they show several drawbacks:

1. distances among stations: the shared car must always be left at the same station in which it has been taken, which often is uncomfortable to reach. Some CSOs however apply fares that allow to use the car for an evening and return it in the following morning without any added cost;
2. *single trips are impossible*: it is always necessary to return the vehicle at the same station or bay from which it has been taken;
3. *it is necessary to book* before accessing the shared vehicle;

4. possibility of not finding any vehicle of the required typology when needed;
5. non flexibility: *the user must return the vehicle at the end of the period booked*. This may constitute a problem if some impedances in the trips occur, such as traffic jams;
6. share of the car with other users: the user cannot leave in the car personal objects that he always needs but he has to take them with him.

In order to improve the degree of flexibility for users, new generation shared vehicle systems are proposed. These systems remove the constraint of booking in advance, and allow to perform single trips and return the vehicle as the user has finished his trip or trip chain. These systems, called *second generation* car sharing systems, will be described in the section 2.1.4.

2.1.3.1. Some examples of first generation car sharing systems

Today carsharing services are widely spread worldwide, with over a thousand cities where carsharing services are available. Existing services include:

1. Zipcar: (www.zipcar.com) developed in Spain (Barcellona), Canada (Toronto and Vancouver), UK (Bristol, Cambridge, London, Maidstone, Oxford), U.S. (Atlanta, Austin, Baltimore, Boston, Chicago, Los Angeles, Miami, New York, Philadelphia, Pittsburgh, Portland, San Francisco, Seattle, Washington). As of November 2012, Zipcar has 767,000 members and offers 11,000 vehicles throughout the North America and Europe, making it the world's leading carsharing network.
2. EasyMotion.pl (www.easymotion.pl) in Poznan (Poland).
3. City Car Club (www.citycarclub.co.uk), developed in the UK in the following cities: London, Bristol, Southampton, Leeds, York, Manchester, Sheffield, Birmingham, Glasgow, Edinburgh, Cardiff.
4. Ibilek (www.ibilek.es): developed in Spain, in Bilbao, Donostia, Vitoria.
5. Respiro (www.respiromadrid.es): developed in Madrid, Spain.
6. Stadtmobil (www.stadtmobil.de): developed in Germany in the following areas: Berlin, Hannover, Rhein-Ruhr, Rhein-Main, Rhein-Neckar, Karlsruhe, Stuttgart.
7. CityCarShare.org (www.citycarshare.org), in San Francisco area, U.S.
8. NTUC Incarne Car Ca-op, CitySpeed and WhizzCar in Singapore (Barth et al., 2006).
9. In Japan: ITS Mobility system in Osaka, Tourist Electric Vehicle System in Kobe, Ebina Eco-Park & Ride (for commuters of a railway station), Inagi EV-Car Sharing for residential areas, Minato–Mirai 21 in Yokohama (Barth et al., 2006).

However, some of these companies, such as Zipcar, allow users to book the car shortly in advance, and prolong the booking period if some impedances, such as queues or traffic jams, occur during the trip. But in all cases the vehicles must be returned in the same pick-up stations.

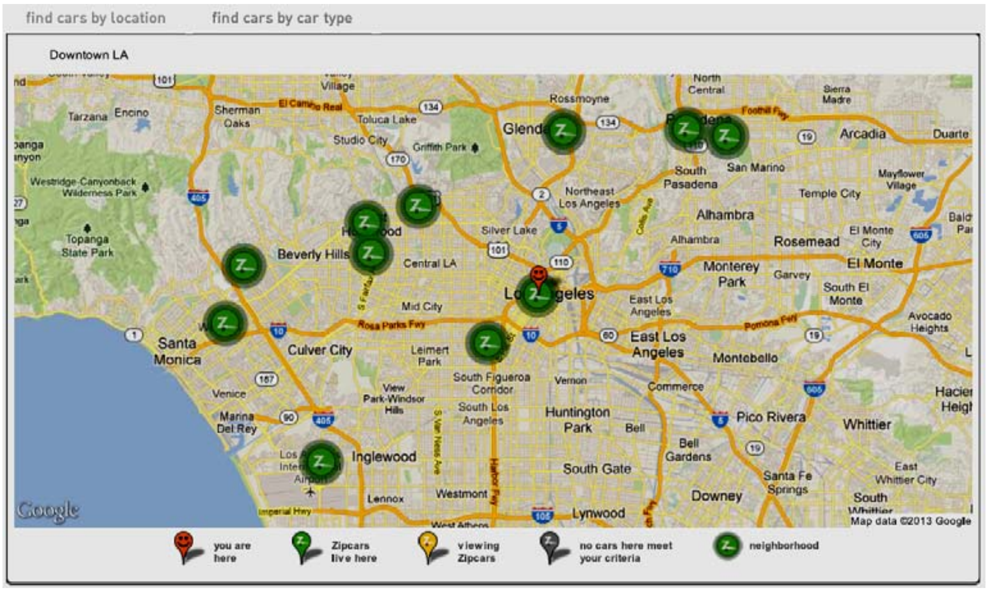


Fig. 2.1. The Zipcar map. It tells the user the state of each station in terms of number of vehicles available and number of free spaces. It also allows users to book and reserve a vehicle. This map refers to Los Angeles (www.zipcar.com).

In websites of the companies cited above, the location of the stations and their state, i.e. the availability of vehicles and of free spaces, is provided to the users. In figure 2.1, the application of Zipcar is shown, which allows users to check the stations' state and book and reserve a vehicle. In figures 2.2 - 2.5, the analogous applications for CityCarClub, Stadtmobil, Respiro and CityCarShare.org are available.

In Italy, several cities have a car sharing system. The Italian Ministry of Environment has launched a circuit called CarSharing Initiative (ICS, i.e. Iniziativa Car Sharing), where members registered to one of cities taking part in the initiative can make use of the car sharing in all the other cities. Currently the Italian cities taking part in this initiative are: Torino and its province, Alessandria, Milano and province, Monza and province, Brescia, Milano, Venezia, Padova, Bologna, Modena, Parma, Reggio Emilia, Rimini and province, Genova, Savona, Firenze and metropolitan area, Perugia, Pescara, Roma, Bari, Taranto, Palermo (www.icscarsharing.it). All cities of ICS have a web site in which the location of the stations is reported. However, the state of each station, i.e. the availability of vehicles or of free spaces, is never shown. With the initiative called "IoGuido", there are promotions like high discount in prices of car sharing systems, aimed to encourage people to visit cities by train + car sharing instead of private car.

Other Italian cities have minor car sharing systems, which do not take part in the initiative.

All Italian car sharing systems do not allow booking shortly in advance.

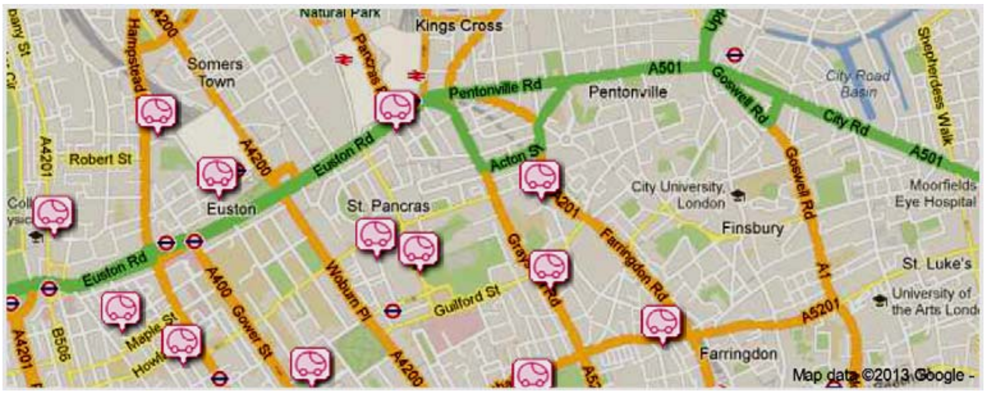


Fig. 2.2. The similar map for CityCarClub, London. Clicking on the station, informations about its state are provided. (www.citycarclub.co.uk).

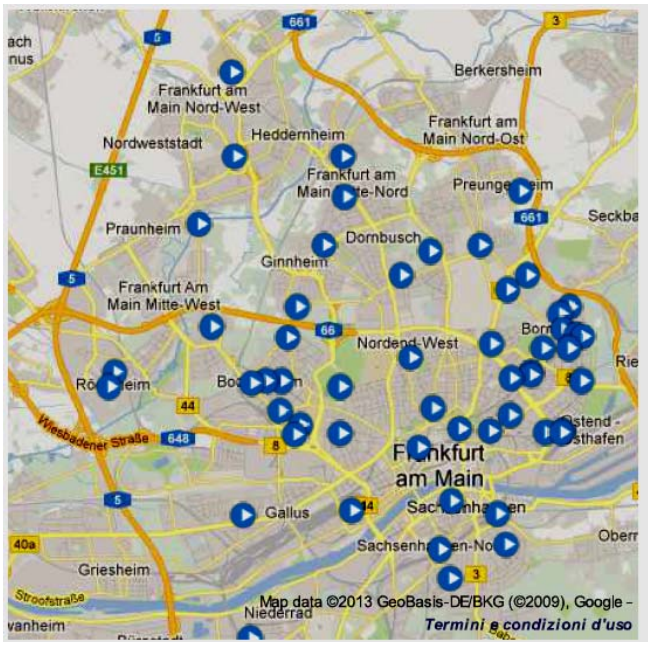


Fig. 2.3. The map of stations of Stadtmobil, in Frankfurt. Clicking on the station (the blue arrow) the information about the station’s state is provided. (www.stadtmobil.de).

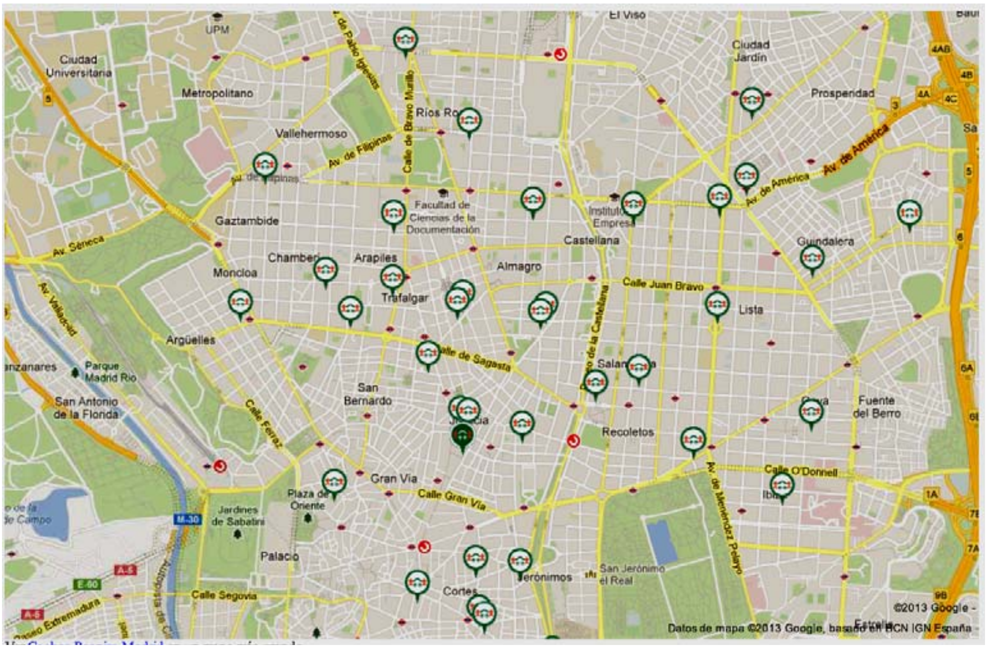


Fig. 2.4. The interactive map of Respiro's stations in Madrid. Again, clicking on the stations the information about them are provided. (www.respiromadrid.es).

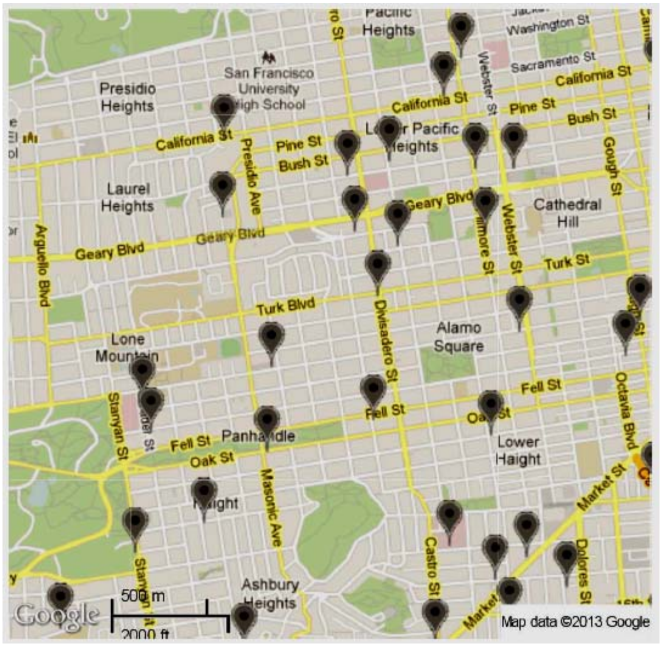


Fig. 2.5. The interactive map of CityCarShare.org stations in San Francisco. Again, clicking on the stations the information about them are provided. (www.citycarshare.org).

2.1.4. Second (new) generation car sharing systems.

New generation car sharing systems have been developed to overcome the barriers of traditional car-sharing systems. These systems provide the following specific services: instant access; open-ended reservation and one-way trips.

2.1.4.1. Main features of the new generation car sharing systems

As stated above, new generation systems provide users the following services:

Instant access: The system can be accessed by users without any need of a booking. The registration or however some form of identification of the users is necessary.

Open ended reservation: The user does not have to specify a return time when he accesses the vehicles. Therefore, he can return the vehicle at any time, when he has finished all his trips. This feature is extremely useful as, because of contingencies, it is usually impossible to forecast the time required to perform a given trip or trip chain. For example, if the vehicle is used to go shopping in the city centre, the duration of the trip depends on: the traffic, the time spent at each shop, etc.

One way trips: The user can return the vehicle in a station different from the pick-up one.

The features provided by the new generation systems on one hand provide users great flexibility, but on the other hand create a serious problem, which is the unbalancement of vehicles available at the various stations, therefore in some stations there is an excess of vehicles, while in others there is lack of vehicles.

Therefore, relocation is necessary.

Relocation strategies can be distinguished in two main groupings: operator based and user based. Operator-based strategies resolve the balancement problem by operators that manually relocate a vehicle or a platoon of vehicles from stations having vehicles in excess to stations having lack of vehicles. In user based strategies, instead, relocations are performed by the trips of the transport system users.

2.1.4.2. Some examples of new generation car sharing systems

Several pilot projects have been developed at the end of 1990s and beginning of 2000s in the U.S. In particular, it is worth to mention:

- The Coachella Valley system (Barth and Todd, 1999): it has been settled in a Californian holiday resort.
- UCR IntelliShare (Barth and Todd, 2003), settled at the University of California at Riverside.
- CarLink I, settled in the Dublin Pleasanton Bay Area, and its continuation CarLink II, settled in Palo Alto, California (Shaheen and Rodier, 2005).

Another new generation car sharing system, Honda ICVS, has been established in Singapore. However, it has been recently removed due to high staff costs.

Finally, another system, Praxitéle, has been developed as a pilot project in Paris in year 2000, and now another system, called Autolib’ (www.autolib.eu), is operating in Paris and Ile de France, and recently it has been settled also in Lyon.

2.1.4.3. Operator based relocation strategies

2.1.4.3.1.Where and when a relocation is required?

The relocation is required when a critical situation occurs. A critical situation occurs when the actual number of vehicles idle at a station equals one of the station’s critical thresholds. Two thresholds could be defined for each station (Barth and Todd, 1999):

- the low critical threshold. If the number of vehicles in a given parking lot goes below the low critical threshold, the station is in shortage of vehicles and some users may be in queue at the current time instant. This situation is referred as ZVT, i.e. zero vehicle time (Kek et al., 2006). When this condition takes place, a request for a vehicle is generated.
- the high critical threshold. If the number of vehicles in a given parking lot at a given time instant goes above the high critical threshold, the station has reached its capacity. This situation is referred as FPT, i.e. full port time (Kek et al., 2006). When this condition takes place, users who want to return the vehicles to the congested station need to be redirected to other stations.

The high critical threshold is upper limited by the space available in the station. Some authors (Kek et al., 2006) calibrate the high critical threshold in such a way to minimize the space occupancy. The low critical threshold could be assumed constant in time or a function of time.

2.1.4.3.2. Which is the supporting station?

If a ZVT situation takes place, a vehicle needs to be allocated to the station in short supply. The point is: which is the station which provides it? Kek et al. (2006) introduce a new threshold: the low buffer threshold which is the minimum number of vehicles that a station needs to have in order to be able to send vehicles. According to these authors, the vehicle request could be addressed only to stations where the number of vehicles is above the low buffer threshold. Among these stations, the providing station could be selected according to several criteria: the nearest station (shortest time criterion), or the station having the highest number of vehicles (inventory balancing).

If a FPT situation takes place, vehicles in excess at the station need to be moved to another station. The point is: which station is able to accept them? Kek et al. (2006) introduce a new threshold: the high buffer threshold which is the maximum number of vehicles that a station can have in order to be able to accept new vehicles. According to these authors, vehicles could be sent only to stations where the number of vehicles is below the high buffer threshold. Among these stations, the accepting station could be selected according several criteria: the nearest one, the closer one (shortest time criterion), or the station having the lowest number of vehicles (inventory balancing).

2.1.4.3.3. Performance of the criteria used for selecting the supporting station
The shortest time criterion relates mainly to service levels, while the inventory balancing mainly focuses on cost efficiency. Therefore, an appropriate choice of relocation technique should be made according to the current system situation. In periods of low usage, the most appropriate relocation technique is by inventory balancing. In periods of high usage, then the shortest time technique performs best.

In reality, the results of these techniques depend also on several set-ups, such as the number of stations, that should be kept as little as possible in order to reduce the number of relocations. Also the number of parking spaces at each station should be quite big in order to maintain the station capacity high, and therefore again to reduce the number of relocations.

The application of these relocation techniques to existing systems has shown that ZVT occurs more frequently than FPT and that individually changing any of the above parameters does not significantly improve system performance, because of the high interrelation among each other.

Other two parameters that should be considered, besides the relocation technique, are:

- The vehicle-to-trip ratio, i.e. the ratio between the fleet dimension and the total number of trips performed in a day. This ratio provides a coefficient of vehicles usage: as this ratio decreases, the utilization of vehicles increases. Barth and Todd (1999) have reported that for the majority of existing new generation car sharing systems, the vehicle-to-trip ratio assumes a value between 0.03 and 0.06.
- The vehicle-to-trip-station ratio, i.e. being v the number of vehicles composing the fleet, s the number of stations, and t the number of trips per day, this ratio is equal to v / (t s) . A lower ratio means that vehicles have been utilized more heavily, or are spread out in more stations (Kek et al., 2006).

The concepts of shortest time and inventory balancing have been implemented in Honda ICVS in Singapore, which is an operator-based car-sharing system. For this system, through simulation, the inventory balancing and shortest time criteria have been studied under various operating parameters, e.g. staff strength, number of car park lots in each station and threshold values (Kek et al., 2006). The interesting outcome is that a reduction of resources usually is expected to worsen the service levels, i.e. number of situations occurrences ZVT and FPT. But in reality when this reduction is balanced with the right combination of relocation techniques and operating parameters, performance indicators can even be improved. In particular, the use of inventory balancing technique in situations of low resources is able to keep low the number of relocations while maintaining satisfactory levels of ZVT and FTP. The use of the shortest time technique instead brings to an increase of the number of relocations, but also on the other hand some improvements in ZVT, while FTP levels are maintained.

All authors agree that to exceed in the number of parking spaces at each station is not good because of the amount of space consumed, therefore the ratio between the maximum number of idle vehicles at a station and the number of parking spaces should be kept close to 1. But on the other hand to reduce in an excessive way the number of parking spaces makes the two critical thresholds nearer to each other, therefore the number of relocations increases relevantly.

2.1.4.3.4. Demand forecasting

In general, the operator based relocation is performed by an operator who manually relocates vehicles among the various stations. Therefore it should be very useful to know in advance the need of vehicles at the various stations and therefore optimize the operator’s route.

In order to perform the most efficient relocation procedure, Barth and Todd (1999) have hypothesized two techniques: historical predictive relocation and exact predictive relocation.

The historical predictive relocation takes into account the trips that generally occur day by day in a given period between each pair of stations and arranges the relocations by forecasting the demand. It is worth to underline that in this scenario all relocations are performed by some operators and therefore it is extremely important to optimize the operators’ trips, in order to keep on one hand the staff reduced, and to satisfy on the other hand the customers’ needs.

The exact predictive relocation tries to determine the transport demand, on hourly basis, between each pair of stations in the different periods of the day.

In fig. 2.6, the performance of the three techniques for demand forecasting is shown. In fig. 2.7, the total average wait time and the number of relocations versus the vehicle to trip ratio is shown.

The non predictive relocation is based on the thresholds mentioned above.

2.1.4.3.5. How is the relocation performed?

The two relocation techniques which have been analyzed in the IntelliShare (Barth and Todd, 2003) are towing and ridesharing. In the towing technique, vehicles are towed from a location to another using a dedicated towing vehicle or simply another vehicle which is part of the system. Towing can be both mechanical, through tow bars, or electronic, through some sensors. This is an operator-based technique but, thanks to improvements in “vehicle intelligence” and vehicle’s sensors, in a short time vehicles will be able to tow on their own, without the need of an operator. Ridesharing is when separate drivers take separate vehicles in some ride, and the same vehicle in some other ride. Drivers could be operators or eventually users. Therefore this technique could be operator-based or partially user-based. In the first case, some system operators relocate vehicles through sharing the ride or splitting into different vehicles. If the system operator needs to get to another station without moving a vehicle, a small scooter is available to travel between stations. This scooter can be mounted on the towing hitch. The towing technique through a towbar is shown in figure 2.8. Ridesharing can be performed also via a regular user trip. Therefore, if two or more users travel from a station with a shortage of vehicles to a station having an excess of vehicles, they are encouraged, through high price reduction, to joint together in the same vehicle (trip joining), while in the opposite case they are encouraged to split into different vehicles (trip splitting). Also in the reservation process, users are encouraged in telling in advance if they accept to joint their trip or to split their trip. In fig. 2.9, the map of UCR Intellishare is reported.

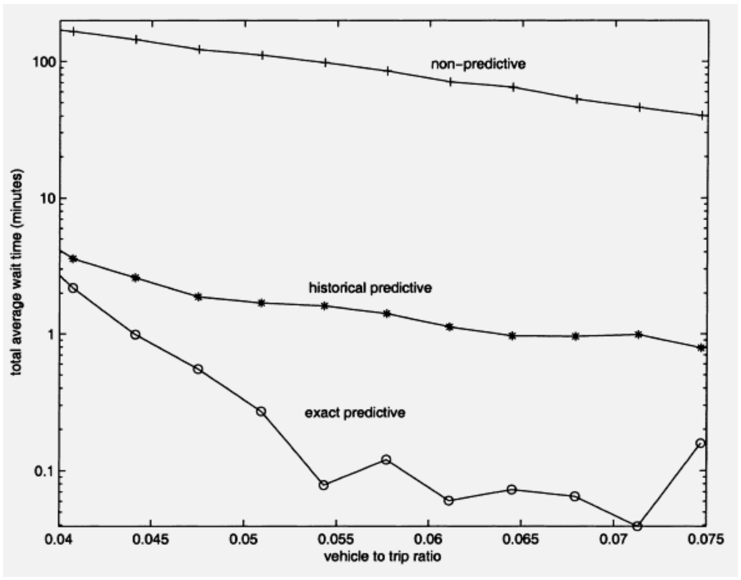


Fig. 2.6. The performance of the three techniques for demand forecasting: the case study is the Coachella Valley in California: non predictive relocation, historical predictive relocation, exact predictive relocation. In the x axis the vehicle to trip ratio is reported. In the y axis the total average waiting time (Barth and Todd, 1999).

2.1.4.3.6. Performance of the ridesharing relocation techniques

The effectiveness of the two techniques of trip splitting and trip joining on the number of relocations has been analyzed Barth et al (2004). The evaluation procedure by Barth et al (2004) refers to the Campus of the University of California at Riverside. The relocation strategy is base on constant critical thresholds. The number of vehicles has been varied between 22 and 30. The overall travel demand volume is about 200 trips / day. The percentage of users who accept to rideshare has been assumed equal to 100% because users, being university students, have no problem in ridesharing, even less if this leads to a discount on the transport cost. In this evaluation, the main interest was to assess the number of relocations necessary during the day to keep the system balanced. The simulation results show that trip joining reduces the number of relocations by 11%, trip splitting by 26%, and the two techniques implemented together by 42%. The analyzed transport system performed very well. However, if the willingness to rideshare is less, the performance drops. In the case of Honda ICVS in Singapore, the willingness to rideshare has been assessed nearly 0%, because Asiatics evaluate much more privacy than monetary cost. In fig. 2.10, the performance of ridesharing in UCR IntelliShare is reported.

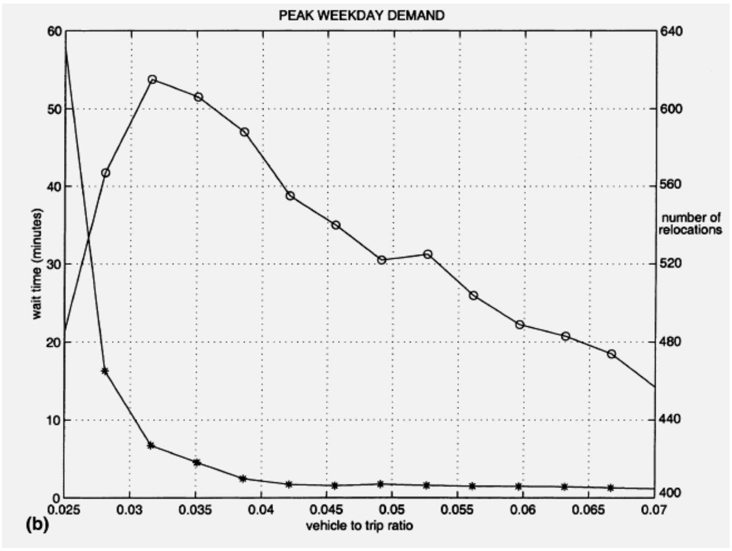


Fig. 2.7. Values of wait time and number of relocation with reference to the vehicle to trip ratio. The case study is the Coachella Valley in California. In the x axis the vehicle to trip ratio is reported. In the y axis the total average waiting time (Barth and Todd, 1999).

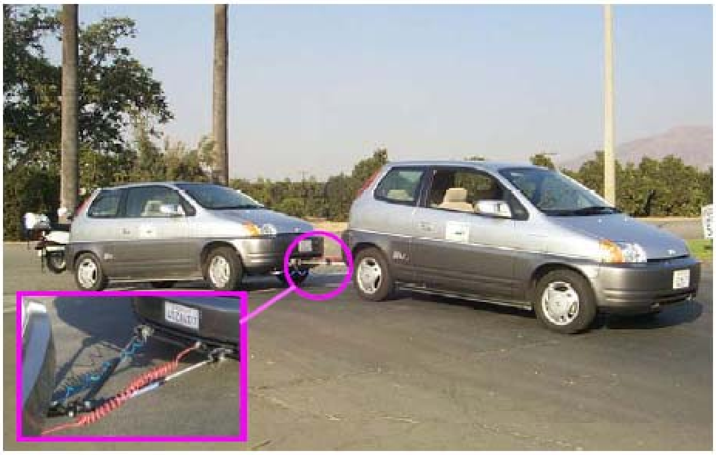


Fig. 2.8. The towing technique, in a mechanical way, through a tow-bar (Barth et al., 2004).

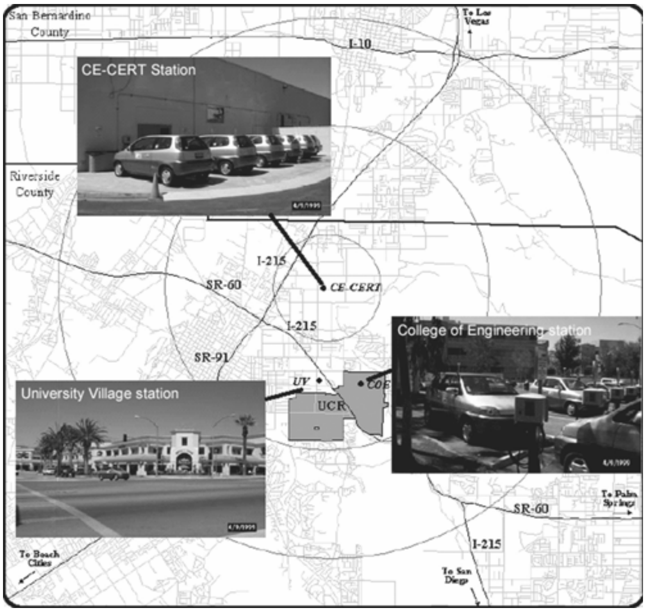


Fig. 2.9. A map of UCR IntelliShare, with the location of the stations (Barth et al., 2004).

2.1.4.4. User based strategies

All the management strategies exposed above are based completely or partially on operators. In this thesis, a relocation strategy fully based on users is proposed. Relocations occur when a user is available in performing it. This type of strategy, works if we make on users the hypothesis of flexibility, i.e. that they accept to be told by the system manager to which station return the vehicle. All details about the proposed user based strategies will be provided in section 3.2.1.

2.1.4.5. Autolib'

Differently from all the other mentioned systems, Autolib' (www.autolib.eu) does not have a relocation system: the user is provided with an application, in which the location of stations and the state of each station is shown. This application helps the user for accessing and returning the vehicle:

- For what regards accessing the vehicle, the user is informed about the state of each station, in terms of number of vehicles available, therefore he reserves the desired vehicle and reaches the station on his own.
- Regarding returning the vehicle, the user is informed about the free space available. If at the destination station there is no longer space available, he returns the vehicle at another station.

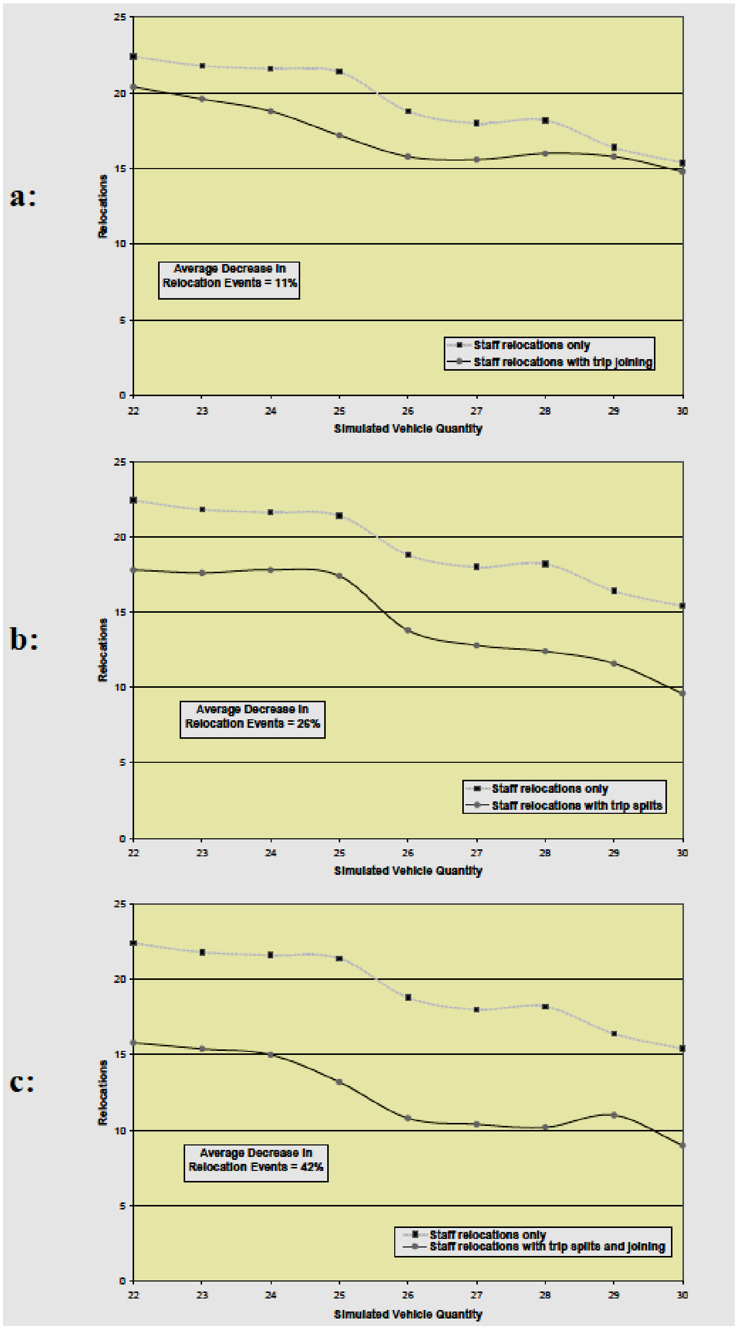
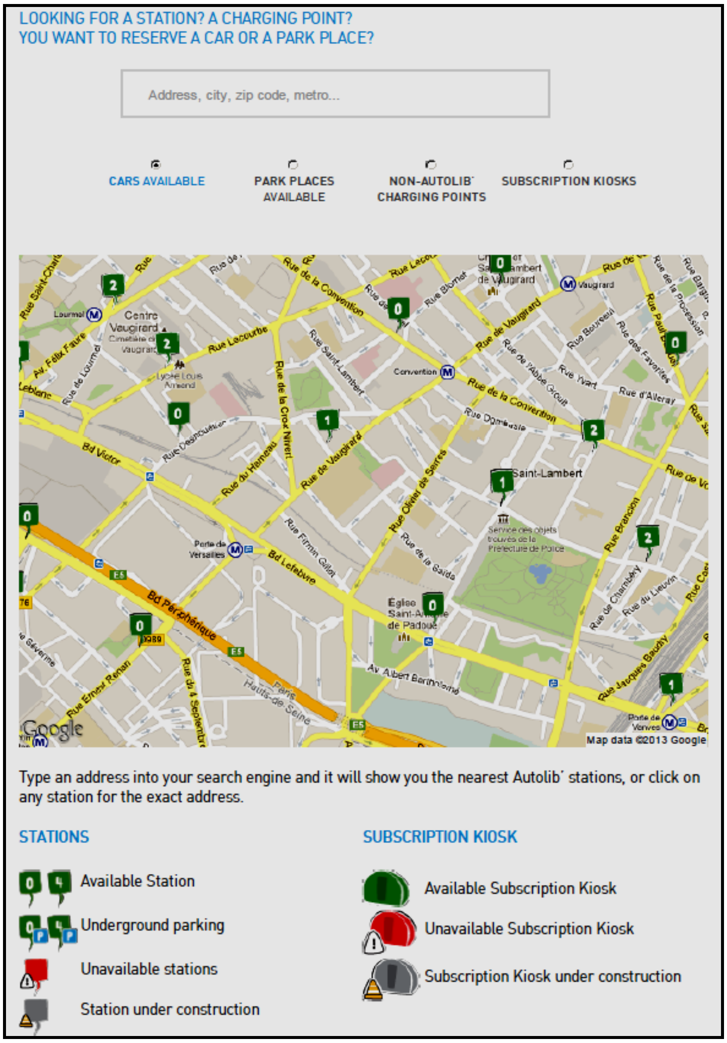


Fig. 2.10. Performance of UCR IntelliShare relocation techniques: (a) trip joining; (b) trip splitting; (c) both. In the x axis the fleet dimension is reported, in the y axis the number of required relocations (Barth et al., 2004).

This management scheme avoids the relocations. However, it creates a disutility to users, which is the necessity to cover major distances on their own, and it results more expensive for installation costs, because, as clearly shown in the website of Autolib', the stations are quite close, with a distance of about 600 – 800 metres, and the number of vehicles is huge. Moreover, this system cannot work if the demand is low, as or the offering in terms of vehicles and stations is greater than the demand, or users have to travel for quite long distances to reach the nearest available vehicle. Furthermore, as reservation and booking is performed through this application, the access to the Autolib' system may result quite difficult for an aged person. In fig. 2.11, the interactive map of Autolib' is reported.


LOOKING FOR A STATION? A CHARGING POINT?
YOU WANT TO RESERVE A CAR OR A PARK PLACE?

Address, city, zip code, metro...

[✓ CARS AVAILABLE] PARK PLACES AVAILABLE NON-AUTOLIB' CHARGING POINTS SUBSCRIPTION KIOSKS

Type an address into your search engine and it will show you the nearest Autolib' stations, or click on any station for the exact address.

STATIONS
Available Station
Underground parking
Unavailable stations
Station under construction

SUBSCRIPTION KIOSK
Available Subscription Kiosk
Unavailable Subscription Kiosk
Subscription Kiosk under construction

Fig. 2.11. The Autolib' stations map (https://www.autolib.eu/stations/).

2.1.5. Third generation car sharing systems

In Ciari et al (2009) and in Schwieger (2003), the adequate level of capillarity of a car-sharing system is discussed. The capillarity is meant by these authors as the degree of diffusion of vehicles within the application area of the transport system, therefore vehicles are available not only at the stations but also along the roads. A higher capillarity of the system can better capture all the potential demand by making the shared system more competitive and more similar to private car. This aspect is particularly suitable for residential areas. A high degree of capillarity results also in a less amount of space needed for the stations, because some vehicles, as stated before, are not placed at the stations but along the streets. This can be bring to the farthest case in which stations are only meant to vehicles maintenance, such as refuelling, and vehicles are available at any point of the intervention area. Capillarity should not be confused with the scale, in terms of number of vehicles composing the fleet. It is clear that an increase in the number of stations better satisfies the user needs but it increases the number of required relocations.

Several newest car sharing systems have been developed according to this concept. In such systems vehciles can be accessed and returned at any point of the area. These systems are called of *third generation*.

Several third generation car sharing systems have been applied on the field. In all these systems vehicles can be accessed with only a reservation performed just a couple of minutes before accessing the vehicle; moreover the vehicle can be returned at any point of the area. Vehicles can also exit the intervention area, in Car2Go can also be used abroad. However, they must be returned in the intervention area. No relocation is necessary. Users have real time information on the position of the vehicles, through an application for computers and i-phones: they reserve the vehicle they prefer and reach the vehicle’s position on their own.

In such systems, as stated above, relocation is not necessary. However, this is achieved because:

1. the user must reach the vehicle’s position on this own; this can also constitute a high discomfort if the nearest vehicle is quite far;
2. the fleet dimension is kept very high to satisfy the users’ demand, and the utilization rate of each vehicle is low, much lower than in the transport system proposed in the thesis.

2.1.5.1. Some examples of third generation car sharing systems

Third generation relocation systems are much diffused nowadays. The most important are Car2Go, DriveNow and Greenwheels. In all these systems, an application is available to users (both for computers and mobile phones) in which the position of vehicles, fuel stations, registration kiosks, etc., is shown on the map. These applications for what regards Car2Go in Ulm and DriveNow are respectively shown in figures 2.12 and 2.13.

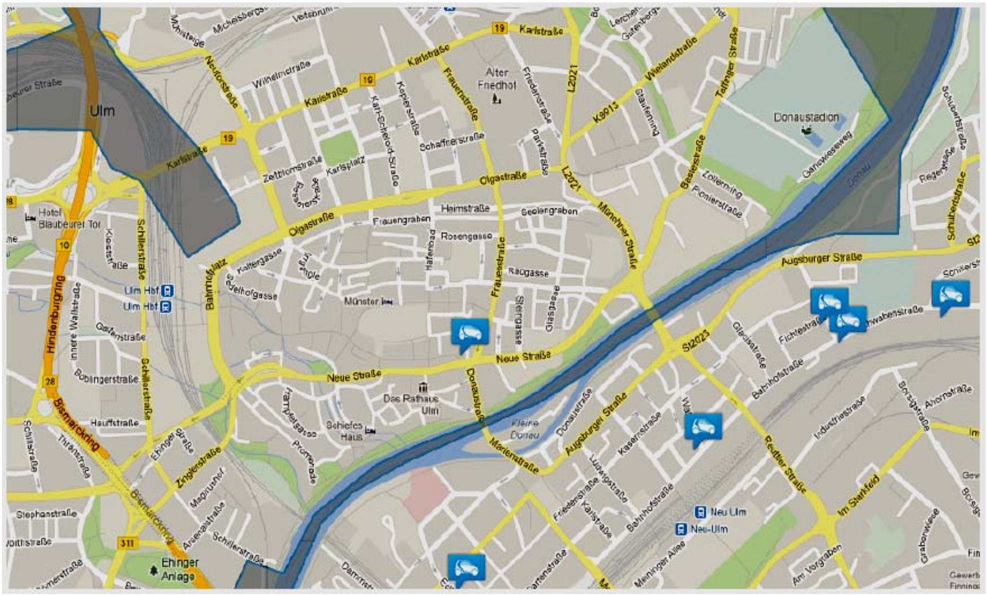
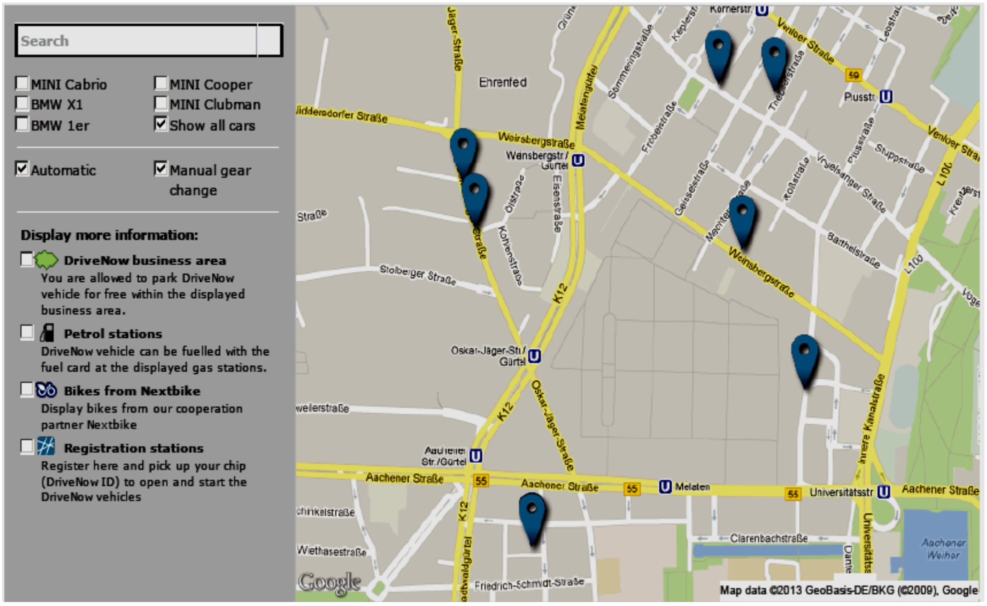


Figure 2.12. A screenshot of Car2Go finder (www.car2go.com).



Search

☐ MINI Cabrio ☐ MINI Cooper
☐ BMW X1 ☐ MINI Clubman
☐ BMW 1er ☑ Show all cars

☑ Automatic ☑ Manual gear change

Display more information:

☐ **DriveNow business area**
You are allowed to park DriveNow vehicle for free within the displayed business area.

☐ **Petrol stations**
DriveNow vehicle can be fuelled with the fuel card at the displayed gas stations.

☐ **Bikes from Nextbike**
Display bikes from our cooperation partner Nextbike

☐ **Registration stations**
Register here and pick up your chip (DriveNow ID) to open and start the DriveNow vehicles

Map data ©2013 GeoBasis-DE/BKG (©2009), Google

Fig. 2.13. A screenshot of DriveNow finder (www.drive-now.com).

Car2go (www.car2go.com) has been initially settled in Ulm, Germany, and now is diffused in other cities in Germany, The Netherlands, England, U.S. and Canada: Amsterdam, Austin, Berlin, Calgary, Dusseldorf, Koln, Hamburg, London, Miami, Portland, San Diego, Seattle, Stuttgart, Toronto, Vancouver, Vienna, Washington.

DriveNow (www.drive-now.com) is diffused in particular in Munchen, Berlin, Dusseldorf, Koln, San Francisco.

Greenwheels (www.greenwheels.nl) instead is diffused all over The Netherlands.

2.2. Bike sharing systems and their relocation procedures

As reported in chapter 1, four stages, or “generations”, of bike sharing programmes could be seen:

- first generation, called white bikes (or free bikes): free of charge, bicycles can be rented for free; it is a simple bike rental;
- second generation of coin-deposit systems, which is also a simple bike rental;
- third generation, or information technology based systems: the state of each station is available, the bicycle can be accessed through a card or a magnetic key;
- fourth generation, known as the demand-responsive, multimodal system: it is developed in order to successfully interact with the other means of transport.

In the first two generation of bike sharing systems, the cycles must be returned at the pick-up station. In the last two generations, bicycles often can be returned at any station of the area.

Bike sharing systems do not show the same problems as car sharing, for what regards relocation. Because of the little space occupied by the bicycle, the relocation is performed efficiently by a van.

Bike sharing systems are now much diffused. Several bike sharing schemes exist in Italy. However, they are not much popular, although the Milan bike sharing system performs very well. Instead one of the most popular bike sharing systems is the London Cycle Hire Scheme.

2.2.1. Bike sharing in Italy.

Nowadays, in Italy there are about 130 bike sharing systems, mainly diffused in the municipalities of the north and of the centre. The regions in which bike sharing are mainly diffused are: Emilia Romagna (19), Veneto (16), Piemonte (15) and Lombardia (13). (Ceccarelli, 2010). These are third generation bike sharing systems.

These systems are distinguished in two typologies according to the typology of access:

- mechanical access through a key
- magnetic access through a card.

In mechanical access systems, the bicycle must be returned at the same pick-up place, but no time limit for returning is given to users. The user is provided with a key that he has to consign when he accesses the bike. When the user finishes his ride, he can have the key back. The systems through keys are often for free.

In systems accessible through magnetic card, the bicycle can be returned in any other parking space. In all these systems, short rides are encouraged: in all Italian municipalities excluded Rome, the fare allows the first half an hour of usage for free. In Milano, the maximum limit of usage is four hours.

In Italy, the access through keys is provided by the agency “C’entro in bici” (fig. 2.14); the access through magnetic card is provided by “Bicincittà” (fig. 2.15). *C’entro in bici* is mainly diffused in Emilia-Romagna and Veneto and it regards 2/3 of the total number of bike sharing settlements. *Bicincittà* regards 1/3 of the total bike sharing settlements and is mainly diffused in Piemonte and Lombardy.

The biggest bike sharing system in Italy is diffused in Milano (called BIKEMI) (www.bikemi.com), which has been launched in November 2008 and now is composed of 1300 bicycles distributed in 100 stations in the historical city centre of Milano (Ceccarelli, 2010). Now is under development its expansion in the peripheral areas, in particular university campuses and the railway and bus stations mostly used by commuters. This expansion plan forecasts 170 new stations and 5000 bicycles. All new stations will have 33 places for bicycles. An i-phone map allows the user to see the nearest station of the bike sharing and the availability of the bicycles.

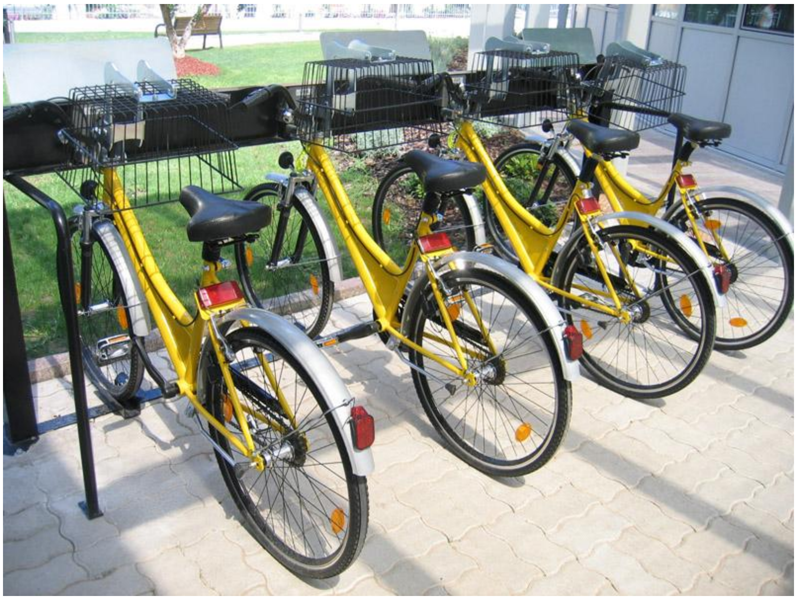


Fig. 2.14. A station of “C’entro in bici”, with mechanical access through a key (Ceccarelli, 2010).

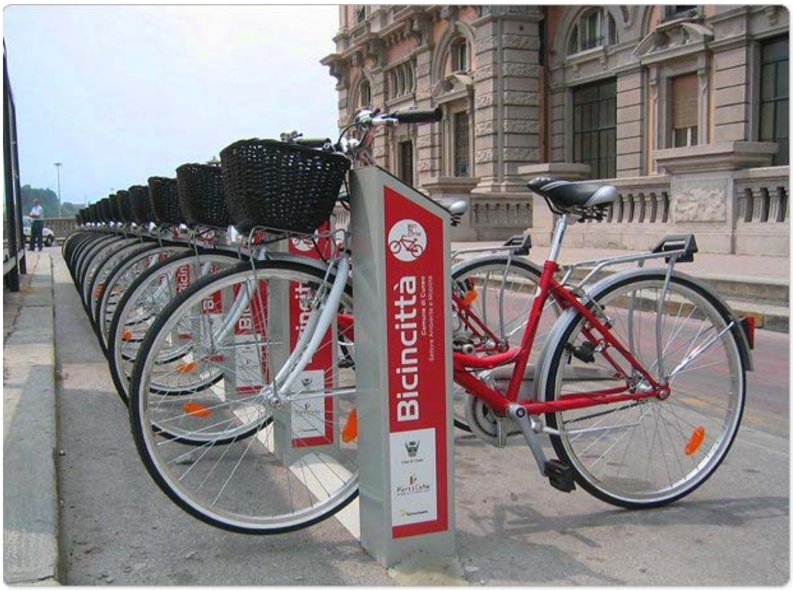


Fig. 2.15. A station of the bike sharing programme Bicincittà (Ceccarelli, 2010).

In Genoa, in April 2009, the system MOBIKE has been launched, and it consists on 55 bicycles distributed in 6 stations. It regards mainly bicycles with assisted driving because of the topography of the territory, with high slopes.

Now in Italy the most important bike sharing systems (of third generation) are diffused in: Milano, Brescia (200 bicycles), Ravenna (140), La Spezia (135), Bergamo (120), Trento (88), Genoa (55) (Ceccarelli, 2010).

For what regards the other cities, there are several bike sharing systems of second generation, i.e. bike rentals, with very little dimensions. This is mainly due to the peculiarity of Italian territory: several small cities. This situation does not allow big enterprises to make fruitful investiments.

2.2.2. The Barclays Cycle Hire.

Barclays Cycle Hire (BCH) is a public bicycle sharing scheme, launched on 30 July 2010 in London, England.

The initial feasibility of the scheme was assessed in “Feasibility study” (2008). This report showed that London was ready for this kind of scheme. In fact the number of cyclists on London’s roads is increasing and the number of injuries and deaths to cyclists is reducing. Several other bike sharing systems are already developed in Europe: in order to be competitive the London Cycle Hire Scheme should have:

- A deposit mechanism
- An annual subscription or registration process
- A strategic pricing structure
- A smartcard system

- Innovative docking stations to make use of available space
- Very secure and easy-to-use docking stations
- Robust bicycles
- Minimum use of vehicles to re-distribute bicycles
- Simple maintenance
- A visible and easily identifiable scheme
- Availability for tourists

Initially, BCH required initial payment of registration and membership fees to be paid in exchange for an electronic access key, but from the beginning of 2011 this was changed to allow casual cycle hires by non-members who have a valid credit or debit card.

2.2.2.1. Operation

Regular users of the scheme can register on the TfL website and sign up for one of three levels of access: daily, weekly or yearly. Users are then sent a key which they must activate before they use it for the first time. A key costs £3, and up to four can be registered under a single account. Scheme members insert the membership key into a docking point key slot; an amber light indicates that the account is being verified, then a green light indicates that the cycle can be undocked.

From 3 December 2010 the scheme has also been made available to casual users who have not registered with TfL. It is sufficient to have a credit card: on the docking station, on the terminal, are reported all the procedures to follow. Once the user has purchased his access period (for either 24 hours or seven days) the first 30 minutes of any journey will be free of usage charges (www.tfl.gov.uk). An interface for booking a cycle is shown in fig. 2.16.

2.2.2.2. Docking stations

Docking stations consist of a terminal and docking points where users pick up and return cycles. The terminal at each docking station contains a screen allowing users to:

- hire a cycle if the user does not have a key;
- print a record of their journey;
- find other nearby docking stations – if one is full or empty;
- get extra time if they need to return the cycle to another docking station; and
- see a local street map, review scheme costs, the code of conduct and translated information.

During high load hours the bikes are moved from the busiest stations to the emptiest using electric vehicles with zero CO_2 emissions. There are a number of applications for mobile phones to help users find the nearest station. A docking station is shown in figure 2.17. The interactive map of the stations in London is shown in figure 2.18. Clicking on the station, you can receive the information about the station's state: number of cycles available, number of free spaces.

2.2.2.3. Relocations

As shown above, relocation does not constitute a problem, bicycles of the BCH are relocated through a van from stations having bicycles in excess to stations having lack of vehicles. Despite this is an operator based scheme, this relocation is quite cheap as a high number of vehicles can be carried in a single relocation trip.

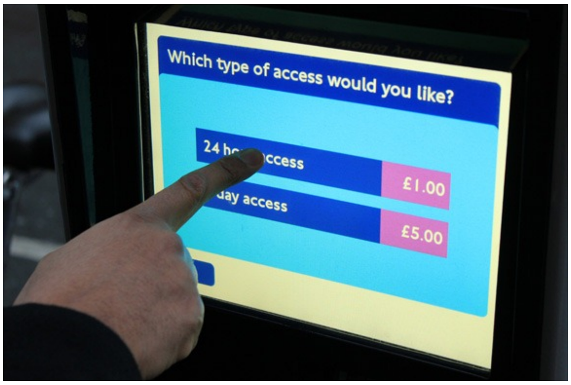


Fig. 2.16. The booking interface at a docking station (www.tfl.gov.uk).

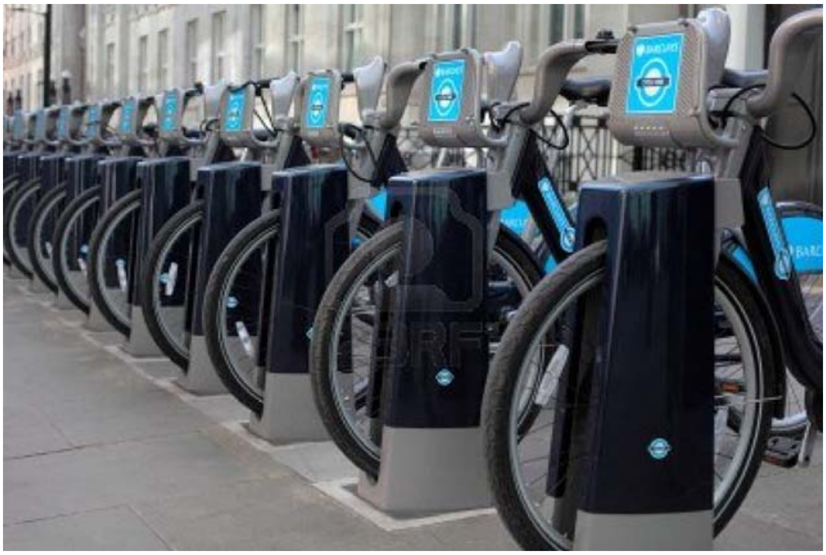


Fig. 2.17. A docking station in the Barclays Cicle Hire (www.tfl.gov.uk).

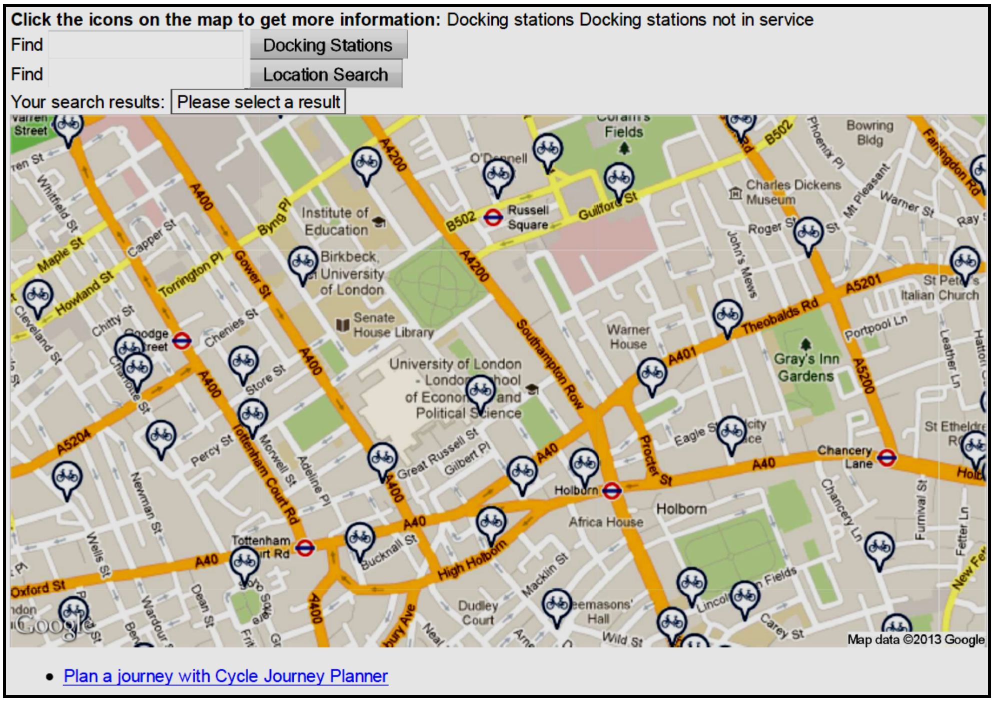
2.2.2.4. Costs

Users of the scheme must pay both an access fee and usage charges.

The access fee costs £2 for a 24-hour access, £10 for a seven-day access and £90 for an annual access. The usage charges are summarized in table 2.1. Source of data: www.tfl.gov.uk. The first half an hour of usage is free.

Table 2.1. Usage charges of the Barclays Cycle Hire scheme.

Time period between picking up and returning a cycle	Charge (£)
Up to 30 minutes	0
Up to an hour	1
Up to an hour and a half	4
Up to 2 hours	6
Up to 2 and a half hours	10
Up to 3 hours	15
Up to 6 hours	35
Up to 24 hours (maximum)	50


Click the icons on the map to get more information: Docking stations Docking stations not in service

Find [Docking Stations]

Find [Location Search]

Your search results: Please select a result

- Plan a journey with Cycle Journey Planner

Fig. 2.18. A screenshot of the BCH map in London city centre (www.tfl.gov.uk).

Chapter 3. The proposed transport system.

Introduction

The proposed transport system is a new multimodal shared use vehicle system for urban pedestrian environments and it involves a fleet of automated personal vehicles. PICAV vehicles are specifically designed for urban pedestrian environments where usual public transport services cannot operate because of the width and slope of the infrastructures, uneven pavements and interactions with high pedestrian flows.

The proposed car-sharing system overcomes some restrictions of traditional car-sharing (first generation) where members need to book cars beforehand and the time the car will be dropped off should be specified (fixed-period reservation); besides, cars must be returned to the same station where they were picked up (two-way trips).

The proposed system provides users with more flexibility thanks to:

- instant access: users can access directly to an available vehicle, without the need to make a reservation;
- open-ended reservation: users can keep the PICAV vehicle as long as needed;
- one way trips: users can drop the vehicle off at any station.

The main problem of the proposed car sharing system is that it may quickly become imbalanced with respect to the number of vehicles at the stations. Due to uneven demand, some stations during the day may end up with an excess of vehicles whereas other stations may end up with none.

New generation car sharing systems often resolve the unbalancement problem through operator based relocation. But operator based relocation has shown to be extremely expensive in terms of staff and management costs, therefore some systems have turned out into a failure, while others only remained pilot projects and have never been settled on a wide scale.

Three new relocation strategies have been studied.

This chapter is organized in the following way. Paragraph 3.1 summarizes the main characteristics of the proposed transport system. Paragraph 3.2 describes the three proposed relocation strategies that aim to solve the unbalancement problem of the proposed transport system.

This part of the thesis has been published in several papers: Cepolina and Farina (2012a), Cepolina et al. (2011a), Cepolina and Farina (submitted to TRB).

3.1. Main characteristics of the proposed transport system

3.1.1. The road network representation.

Trips by PICAV can have origin or destination either in a station or in a unit.

Stations

Stations are distributed at different locations through an area, and play the role of origin and destination of user trips and battery charging points. The stations are conveniently placed in locations close to both intermodal exchange points and attractors. Intermodal exchange points may be for example: bus stops, underground stations, railway stations, parking areas. Attractors are points where long term activities can take place: for instance, museum, offices, schools, etc. Each station is characterised by an attractivity that is a function of the number and the typology of the attractors within the station's influence area.

Units

For simplicity of representation, the roads in the intervention area have been divided in sections of a given length. Each road section is represented through a node, called unit, placed in a baricentric position of the road section. All the information regarding the road section, i.e. transport demand, number of users waiting, number of vehicles parked, is assigned to the unit. Each unit is also characterised by an attractivity. The unit's attractivity is a function of the number, typology and attributes of the attractors localized in the related section. In this case attractors are places where a short term activity takes place: for example, shops. Units can both play the role of localization of short term stops, and of origin and destination of the trips. Units are not placed on the border of the intervention area.

A trip by PICAV could be a travel on board the PICAV vehicle (single task trip) or a sequence of travels on board and activities that require short term parking (less than 1 hour) along the street (multi task trip).

If during a trip the user makes a stop that lasts more than 1h, the vehicle will need to be returned. If the stop duration is less or equal to 1h a short-term parking is permitted and the vehicle does not have to be returned.

3.1.2. Trip typologies.

We consider four types of trips in the intervention area:

Type A is a multi-task trip: the user has a sequence of short term activities to perform in the intervention area, with the origin and the destination of the trip on the border of the intervention area. The user reaches the intervention area border on his own, he picks-up a PICAV vehicle, he makes an activity travel pattern with short term stops (like shopping, visits to: banks, post office, etc.), he returns the vehicle and he goes back home again on his own (e.g. by public transport, by car, etc.).

Type B is a single-task trip, with either origin or destination on the border of the intervention area. The user has to perform an activity that requires a long stop in the intervention area. The user arrives on the intervention area border on his own, he picks up a vehicle at a station, he reaches to the station closest to his activity place and returns the vehicle there and he reaches the activity place by foot. Type B trip is also the return trip: the user has finished his activity within the pedestrian area and wants to go back home. The user thus goes by foot to the station closest to his activity place, he picks-up a vehicle, he returns the vehicle on the pedestrian area border in a station and goes back home on his own. In the third management strategy, the user calls the system manager, requires a vehicle, then enters the vehicle and reaches the area border by PICAV.

The trip typology D is a single-task having both origin and destination inside the intervention area.

The trip typology E is a trip chain having both origin and destination inside the intervention area.

The various trip typologies are shown in the figure 3.1.

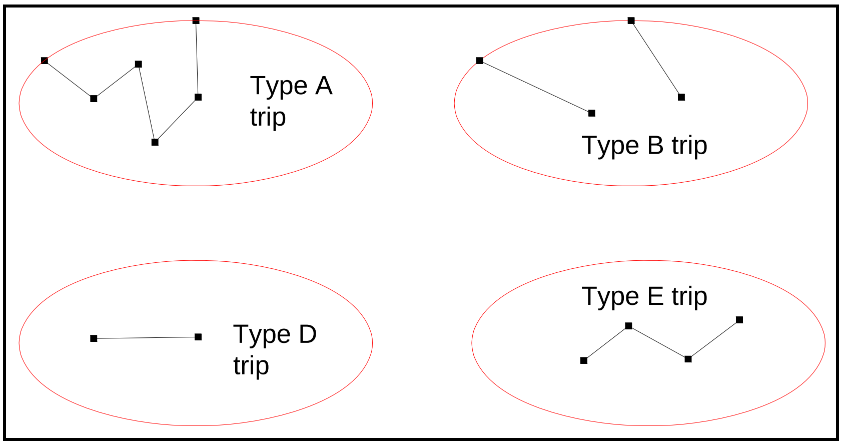


Fig. 3.1. The various trip typologies: type A (above left), type B (above right), type D (below left), type E (below right).

3.1.3. Communication and system management.

The PICAV vehicles are networked and can communicate through a cellular GPRS technique with each other, with the system manager, with the city infrastructure and with public transport in the surrounding area, which allows a high level of intermodal integration. The communication among vehicles is important for users who travel together. The communication between vehicles and system manager is important:

- for relocations,
- to provide the user real time data about: waiting times at public transport stops, traffic flow values, unpredictable events (e.g. street closures etc.).
- for booking, check in and check out of vehicles.

The system manager deals with all the activities of the system management, such as:

- reservation of vehicles, check in and check out;
- communication with the PICAV user;
- make automated vehicles relocate in the required time;
- make automated vehicles reach the user's position;
- constantly monitor the state of each station: the number of vehicles available, the number of users in queue, the number of vehicles whose charge level is less than the minimum: it is necessary for performing the relocations;
- constantly monitor the position and the state of each vehicle: available, unavailable but in charge, occupied by user, relocating.

3.1.4. Maintenance and refuelling.

The activities of vehicles maintenance regard: cleaning and refuelling. For what regards refuelling, vehicles are electric and therefore refuelling consists on battery recharge. Two battery charging techniques have been hypothesized.

1. The first technique is called "opportunity charging": the vehicle is put in charge whenever and wherever power is available. As a vehicle reaches a station, it is put in charge. This technique, as will be exposed more in detail in the following sections, is very efficient as it allows vehicles to always maintain a high level of charge and therefore all the fleet is available in the peak demand periods. However, this technique is good as the vehicle has a battery without memory effect.
2. The second technique requires that the battery is used always in its interval of maximum performance: i.e. the charge level should never go under 40%, and when the 80% of charge is reached, the battery is unplugged, as to reach the 100% of recharge a long amount of time is needed. This technique is not efficient as the previous one, however it is the best from the point of view of the battery.

3.1.5. Night rebalancement

At the end of a working day, vehicles must be redistributed among the stations. Given the PICAV vehicles characteristics, they will be able to relocate themselves among the stations and therefore perform by themselves this rebalancement

3.2. Relocation strategies

3.2.1. The first management strategy: flexible users

The first management strategy admits the existence of a system supervisor who is in charge of addressing at least part of the users (flexible users) to specific stations.

Vehicles can be accessed only at stations and they need to be driven by a person (they do not move in a fully automatic way). A fully user based relocation strategy is proposed. A system supervisor is in charge of addressing a subsection of users, who are called flexible users, to specific stations. A *flexible user* is a person that has a final destination outside the pedestrian area and reaches it by public transport; therefore a flexible user has a set of stations that are equally suitable for returning the vehicle. We call this a *user's choice set*. It includes all the stations close to inter-modal exchange points where public transport services, which are suitable to reach the user's final destination, stop. The flexible user agrees to return the vehicle to any of the stations within his choice set, because from these stations he can easily reach his final destination.

When a flexible user finishes his mission in the pedestrian area, he calls the system supervisor and asks where he has to return the vehicle. The supervisor makes a decision according to the following informations:

- the *choice set* of the flexible user;
- the *current waiting times* at the stations within the user's choice set;
- the *travel times* between the user's current position and the stations within his choice set.

The supervisor assigns a station to the user that results good from both the point of view of the flexible user (since it belongs to the user's choice set) and of the transport system management. More in detail, the destination station is the following:

- if there are some users in queue in the stations belonging to the choice set, it has the greatest ratio between: the maximum waiting time, and the distance from the user's current position;
- if there are no users in queue in the station belonging to the choice set, it has the lowest product between: the number of vehicles available, and the user's current position.

In this management strategy, when his trip is over, the user calls the system supervisor and receive the indication of the destination station within his choice set. The call to the supervisor takes place:

- if the user performs a single mission trip from inside the intervention area to the border, he calls the supervisor as he enters the vehicle;
- if the user performs a trip chain with destination on the border of the intervention area, he calls the supervisor as he departs from the last stop of the trip chain.

The flexible user is supposed to return the vehicle to the station chosen by the supervisor. This schema should help in keeping a internal balance between the number of the vehicles at the stations, without the need to have external operators that, according with relocation mechanisms, transport vehicles from a station with an excess of vehicles to the station in short supply. This scheme can work in contexts where the public transport is mainly used for reaching the pedestrian area. In fact in this case the user can exit the pedestrian area close to any stop where suitable public transport lines pass. In this case the number of flexible users is supposed to be consistent. In this management scheme, the transport system characteristics are: the fleet dimension, and its distribution among stations at the beginning of the simulation period. Both parameters are described through one vector, whose number of elements equals the number of stations, and the value of each element is the number of vehicles at the given station at the beginning of the simulation time. The fleet dimension is therefore the sum of all the vector's elements. These values are assessed through an optimization algorithm.

This first management strategy performs very well and heavily reduces waiting times and costs with the drawback of a slight disutility to users. But this strategy works if the demand is balanced, i.e. if during the day the number of users who have a given station as origin is the same as of users who have this station as destination. Moreover, all the stations of the area must belong to at least one choice set of flexible users. For example, Genoa case study, which will be described in chapter 7, is a problem because the two inner stations do not belong to any choice set. If therefore the demand is unbalanced and some stations do not belong to any choice set, the waiting time increases heavily and the whole system collapses.

It is therefore necessary to design other management strategies who do not have this drawback.

3.2.2. The second management strategy: automated vehicles

In the second management strategy, vehicles can be accessed only at stations and are assumed to be able to move in a fully automatic way. A fully vehicle based relocation strategy is proposed: vehicles automatically relocate among the stations according to the indications of a system supervisor.

This management scheme, despite vehicles have to travel at a much lower speed, results to be very successful, and the costs of the staff necessary to relocate vehicles are drastically cut. However, it is still not applicable, as there is a legal problem of determining the responsible in case the automatic vehicle has an accident.

When relocations are required, unused vehicles are redirected from a station to another, according to the system needs and the actual waiting times at the stations. A relocation is required when a critical situation occurs. A critical situation occurs either when a station is in shortage of vehicles, therefore users may need to queue; or when the station has reached the capacity, therefore arriving vehicles need to be redirected to a close station, because in the destination one there is not space any more. The first critical situation is called ZVT, i.e. zero vehicle time; the second one instead is called FPT, i.e. full port time.

As stated in chapter 2, in the same way as operator based procedures, in order to control the insurgence of these two critical situations, two thresholds have been introduced, respectively the low critical threshold and the high critical threshold. Moreover, in order to determine the supporting station, other two thresholds are introduced: the low buffer threshold and the high buffer threshold.

When a ZVT situation takes place, a request for a vehicle is generated. The vehicle request is addressed only to stations where the number of vehicles is above the low buffer threshold. The low buffer threshold is the minimum number of vehicles that a station needs to have in order to be able to send vehicles. Among the stations to which the vehicle request could be addressed, the providing station is selected according to two criteria: the closest station (shortest time criterion) and the station having the highest number of vehicles (inventory balancing). The shortest time criterion relates mainly to service levels, while the inventory balancing mainly focuses on cost efficiency. Therefore, an appropriate choice of relocation technique should be made according to the current system situation: in periods of low usage, the most appropriate relocation technique is by inventory balancing whilst in periods of high usage, then the shortest time technique performs best.

If the number of vehicles in a given station at a given time instant goes above the station's high critical threshold, a request for relocating the vehicle is generated. The vehicle request is addressed only to stations where the number of vehicles is below the high buffer threshold. The high buffer threshold is the maximum number of vehicles that a station needs to have in order to be able to receive vehicles. Among the possible receiving stations, the supporting station is again selected according to the shortest time and to the inventory balancing criteria. If the shortest time criterion is chosen, the supporting station is the closest; if the inventory balancing is chosen, the supporting station is the one having the least number of vehicles.

However, despite exceptional cases in which one or two stations have a particular low capacity for infrastructural reasons, the FPT situation is always avoided through avoiding the ZVT.

In this management scheme, the transport system characteristics are:

- the fleet dimension: the fleet is assumed equally distributed among stations at the beginning of the simulation day
- the low critical thresholds: they are given in the form of an array. Each element of the array refers to a station and it is the value of the low critical threshold for the given station. Low critical thresholds are assumed constant with time, i.e. they are the same in all the simulation period.
- the low buffer thresholds: they are given in the same form as the low critical thresholds.

The dimension of the two above mentioned arrays equals the number of stations in the area, the value of each vector component is the station's low critical threshold or low buffer threshold. The transport system performance is considerably affected by the fleet dimension and by these two vectors, therefore their values should be carefully selected. The transport system performance is not affected by the high critical and high buffer thresholds, apart exceptional cases. Indeed, the high critical threshold is uniquely fixed through the station capacity. Moreover, a relocation strategy based on ZVT occurrences, automatically guarantees that FPT occurrences do not take place.

3.2.3. The third management strategy: capillarity

In this scenario vehicles are available not only at stations but also along the roads. A trip by PICAV may have, as origin or destination, either a station or any position along the roads within the intervention area. When the origin of the user's trip is not a station, a PICAV reaches the user in a fully automatic way.

A fully vehicle based relocation strategy is proposed. Relocations are required:

1. when the number of vehicles available at stations is below the low critical thresholds (ZVT situation). In this case, the request for a vehicle could be addressed:
    a. firstly to the stations where the number of vehicles is above the low buffer threshold (as described in chapter 2); among these stations, the providing one is selected according to the shortest time or the inventory balancing criteria.
    b. otherwise, to the vehicles parked along the road. In this last case, the nearest vehicle automatically relocates towards the station in shortage.
2. when the origin of the user's trip is not a station. In this case, the system supervisor assigns to the user the *vehicle nearest to the user's position*. If the nearest vehicle is in a station, it can be provided only if the number of vehicles available in the station is greater than the low critical threshold of the station. Results from simulation experiments show that if we refer to the low buffer threshold instead of the low critical threshold situations where all the parked vehicles are at stations occur very often as stations are never capable to provide vehicles.

At the end of their trips, the user can leave the vehicle at any position along the roads within the intervention area. When a vehicle is returned, if the level of battery charge is below the minimum charge level, the vehicle automatically reaches the nearest station to recharge the battery. As soon as it reaches the minimum charge level, it becomes available and if not required, continues the charging process.

3.2.4. The transport system modeling and simulation

To test the proposed transport system and particularly the three relocation strategies, a micro simulator has been developed.

The micro simulator receives in input: a simulation time period, a road network, a PICAV fleet, the PICAV transport demand and the parameters related to the relocation strategies (thresholds). The simulator allows to track the second-by-second activity of each user, as well as the second-by-second activity of each PICAV vehicle. The micro simulator gives in output the transport system performances, in terms of level of service (LOS) provided to users and in terms of efficiency from the management point of view. A full description of the micro simulator is provided in chapter 4.

The micro simulator aims to be a useful tool for decision makers to evaluate what if scenarios: for example, what can be the impacts on the proposed systems of peaks of demand, temporary road closures, station closures, a change in the fleet dimension or in its distribution among stations.

Chapter 4. Modelling and simulation of the proposed transport system.

4.1. Introduction on modelling and micro simulation

Simulation can be defined as the representation of reality through informatic models. The simulation is mainly used as a support to decision problems, in order: to evaluate the effects of major modifies to existing systems, and to project new systems (Law and Kelton, 1991).

The usage of models for decisions support has always registered great development. The first typology of such models are the scale models: they reproduce the reality at a reduced scale, they are very effective but also extremely expensive. An example of such models may be a plastic of a building. Other models are the logic-mathematical ones, which anyway become extremely expensive as the system becomes more complex.

The main targets of the simulation are:

- To achieve a better comprehension of the system. Indeed several choices performed in our system are the result of non satisfying results of the simulation.
- What if analysis on the system, e.g. to check what happens if I modify the system.
- To compare alternative decisions with each other.

The real world which is subject of the survey is represented through the model, which is a real-world representation of the reality that we want to study. The first relevant choice is to determine the system borders, and therefore to decide which is worth to consider and which is worth to exclude in the future modelization. An incorrect choice of the system borders may result in relevant changes in the simulation output data. Outside the system borders is the environment, which however have some relations with the system (Law and Kelton, 1991).

The system is characterized by some elements, called entities, which interact with each other. The entities are described through attributes. For example, if the system to model is a petrol service station, we have the entity “vehicle”, which has attributes: identifier, arrival time at the service station, begin refurnishing time, exit time from the station. If the purpose of the simulation is to project a new petrol station and therefore to decide the number of pumps to install, the colour and the mark of the car are not relevant and therefore they are not going to be attributes.

The state of the system or of an entity is determined according to the values assigned to attributes. In the previous example, the state of the system is determined as the number of free pumps, and the state of each vehicle may be 1 if it is refurnishing and 0 if it is in queue.

The simulation may be deterministic or stochastic, discrete or continuum. It is deterministic if the model evolution in time is determined in an unique way from its initial conditions. It is stochastic if instead in the model random variables are present.

The simulation is continuum if it describes the system through a system of equations to resolve numerically. It is discrete if it is realized through discrete-event systems which modify their state at discrete time instants. Examples of discrete simulation are: queue systems, Monte Carlo simulation, complex systems on large scale.

The simulation may be event-based, activity-based, process-based and agent-based.

In the event-based simulation, each event corresponds to a change in the state of a component (object) of the system. In the case of petrol station, an event may be the beginning of the service. The activity is comprised between two events: for example, the waiting in queue, or the refurnishing at the pump, are considered as activities. Actually, often activities are confounded with the beginning or the ending of themselves, therefore often they coincide with the events. Elementary activities can be considered of two typologies: the programmed activities and the conditioned activities. The conditioned activities take place when some logical conditions take place. Programmed activities instead must take place in prearranged times, independently of the state of the system. An example of a programmed activity may be the opening or closing of a pump at a petrol station at a given time. Conditioned activities are scanned in each instant of the simulated time, while programmed activities are scanned only once.

A process is a sequence of activities and events: all the events and the operations of an entity are joined together in an unique sequence called process. An example of process may be the sequence of activities of a refurbishment at a pump: arrival, waiting in queue, refurbishment, waiting for getting into the main road, exit of the system.

The agent-based simulation is the most recent, and it is considered the best modelling of a classical problem of simulation: the prey and predator one. In the agent-based simulation, the system is characterized of agents which interact among each other and make it evolve. An agent is an entity with targets and properties and most of all with the capability of taking decisions. The agent interacts with the system and with the other agents, it is autonomous, flexible and it has memory. The system instead represents the entities which have an effect on the agents, which are the effective object of our study. The time advances in regular steps and at each time step the actions of each agent are executed. In the prey – predator model, for example, at each iteration in a casual way the direction and the length of the movement of each agent is determined. If in a given instant the prey is close to the predator position, at the following time step the prey obviously exits the system. The last characteristic of agent-based simulation is that an agent, according to its fertility, can give birth to other agents of the same typology.

The main steps of a simulation modelling are:

1. Analysis of the problem.

2. Definition of the targets. The simulation is a representation of the reality and so it chooses which aspects to model and which to neglect. Some aspects can be neglected as they are not under study, others can be neglected because to exceed in the level of detail can be counterproductive. It is indeed necessary, every time we model something, to keep always the contact with the reality and with the system we are modelling.
3. Conceptualization of the model. This aspect has already been introduced in the previous. In this phase we choose what to represent and to simulate of the system and the model is realized in practice.
4. It is opportune to develop the model before collecting data. A preliminary collection of information is necessary for the phase 1 (problem analysis) in order to avoid to misunderstand the problem because of the too few informations. The real data collection must be performed after developing the model of the system, given that data collection may be also very onerous from both economic and of working hours point of view.
5. Development of computer program.
6. Verification that the model has been correctly implemented, i.e. that there isn't any programming bug. The verification phase is extremely important, it is usually required that another person, different from the programmer, verifies the code and cooperates strictly with the programmer in order to correct the error presents.
7. Calibration of the model's parameters
8. Validation: i.e. to verify that the model correctly represents the real world system. The validation is possible if we have a real world system. In this case, some approaches to validation may be: comparison of experimental and real data, evaluation of system's experts, animation.

4.2. Object-oriented programming

Object oriented programming has been chosen to develop the simulator because it has several advantages, which will be described herewith. The language chosen is Python 2.6. The Python language is very simple and friendly, the choice of the version 2.6 is reasonable because of the great number of packages available. In the following, details about the object oriented programming will be provided.

Object oriented programming is the newest and nowadays more popular programming technique. It is the result of several years of research and improvement of the programming methodology. Before the born of object-oriented programming, structured and procedural programming was used, but they have shown several disadvantages, in particular the high quantity of code necessary to modelize complex systems, and the difficulty to modify the code in a second time, if e.g. something in the system has changed or new targets of the programme have been defined.

4.2.1. From non structured programming to object oriented

In non structured programming, the code is constitute of an unique block, called “main”, from which data are manipulated in a fully sequential manner. All data are represented only through global variables, and therefore are located in the memory for all the time in which the code remains under execution. This programming technique is absolutely limited and full of disadvantages: the code may be repeated several times and the system employs a high amount of resources.

In procedural programming, the parts of code which are repeated several times are grouped and recalled every time the necessity arises. A procedure is considered as a sub programme which plays a pre determined role. The main code still exists but it mainly consists on the invocation to the procedures defined in the programme. Procedural programming is better because the code is more reduced and therefore the number of possible mistakes is also smaller.

Modular programming allows to re-use the procedures created by a programme in order to allow other programmes to make use of them. Therefore, the idea was to group the procedures having a common purpose, e.g. procedures for executing a particular mathematical formula, in separated modules.

4.2.2. Object oriented programming vs. procedural programming

In procedural languages, i.e. non-object oriented languages, programming is mainly oriented to the action, and the function is the programming unit. The methodology is the functional decomposition. In object oriented languages, programming is oriented to the object. The programming unit is the class and the methodology is object oriented decomposition.

Non-object oriented programs may be a long list of statements (or commands). More complex programs group smaller sections of these statements into functions or subroutines, each of which performs a particular task. With designs of this sort, it is common for some of the program’s data to be global, i.e. accessible from any part of the program. In object-oriented approach the data are not directly accessible by the rest of the program. Data are indeed accessible by calling specially written functions, commonly called methods. These act as the intermediaries for retrieving or modifying the data they control.

The main components of object-oriented programming are: classes, objects and methods. In object oriented programming concepts are represented as “objects” that have: data fields, i.e. the attributes that describe the object; and some associated procedures, i.e. the methods. Objects, which are instances of classes, interact with each other. An object-oriented program may be viewed as a collection of interacting objects, opposed to the conventional model, in which a program is seen as a list of tasks (or subroutines) to perform (Lewis and Loftus, 2008).

4.2.3. Classes, objects, attributes and methods.

Objects have properties (data) and methods (procedures), which operate on the data of the object itself.

Object oriented programming has the purpose of formalizing the objects of the real world and of building through them a virtual world. The main concept regards the class: objects of the same class have the same characteristics. For example, if we are trying to modelize a petrol pump, we create a class "car" which has for objects all the cars that have entered the system. An other class is "servant", which has for objects all the people who serve at the pump.

The part of the real world which is reconstructed virtually through objects is called "applicative domain".

A class is the method through which an object is created and identified. For what regards its informatic "constitution", a class is a data type, like integers and strings. Objects are instead the entities of a programme which interact with each other. Objects are created during the execution of the algorithm and each object is part of a class. A class can create several objects, each object of the class is distinguished from the others though they belong to the same typology. An object is a class instance. To create an object, the instantiation of a class is performed. In this phase, a part of the memory is reserved to keep the values of the object's attributes.

The set of values of an object's attributes is called state of an object and generally varies during the simulation time. An object's attribute, also called member variable, describes the main characteristics and properties of the object itself. For example, in the case of a petrol pump, an attribute of the object pump is the state, i.e. free or occupied.

A method, also called member function, is an action that an object can perform. The declaration of a function is composed of: name of the method, input data, output data. A method may be for example, end service at the pump.

In order to communicate with each other, the objects can make use of methods; an object can also invoke the method of another object. Methods can be distinguished among: constructors, distructors, accessors, mutators.

The constructor is a particular method which is invoked when an object is created and which contains all the instructions which are necessary for its initialization. In the simulator of this thesis, for example, the method "presence picav" provides to the object vehicle just created all the necessary informations, such as: station in which it is parked, battery charging level, state = empty, etc.

Distructors play the role of destroying an object. However, several object oriented programmes, such as Java, C++ and Python, do not have distructor methods, but the object is automatically removed when it exits from the context.

Mutator methods make a modification on the state of an object. Mutator methods can be: public, protected and private. Public are those which can be accessed by any class and object. Protected mutators allow the access only to classes and objects whose typology is a sub class of the mutator's class. Private mutators allow the access only to the objects of the class itself in which the mutator is used.

4.2.4. Abstraction

Programming languages have evolved in such a way that source codes could abstract completely from the way they are executed. More in detail, a programme is firstly written, then compiled, i.e. the code is translated in the machine language. The first languages, such as Cobol, were much close to the machine language. The newest programmes, such as Java and C++, have an approach of problem solving as close as possible to real world – and therefore much different from the machine language.

4.2.5. Relations among classes.

A class which does not interface with the other classes is not significant in object oriented programming. Objects interact with each other using the exchange of messages to require the execution of a specific method. This communication allows to identify inside the programme a series of relations among classes whose documentation results much useful. For example, if two classes show a similar behaviour, they can be conglobated in another class of higher level.

The most common relations among classes can be use relationship, containment relationship, inheritance relationship:

4.2.5.1. Use relationship.

Use relationship is the most intuitive and diffused typology of relationship. In particular, a class A uses a class B if an object belonging to the class A uses a method of an object of class B, and sends to it messages. More in detail, in use relationship it is possible to pass from the objects of class A to objects of class B simply referring to an object. For example, an object “car” is associated to an object “petrol pump” because the car refurbishes at the pump, and the relationship in this case is “refurbishment”.

4.2.5.2. Containment.

The relationship like containment is the following: an object belonging to class A contains an object of the class B if B is an attribute of A. For example, a class “service station” contains the class “pump” because the class “service station” is composed of pumps. Composition is stronger than containment, because it allows a class contained to be part of only one class container.

4.2.5.3. Inheritance.

The inheritance is used in the phase of development and structuration of the software and it allows to derive new classes departing from the classes already defined and therefore it realizes a class hierarchy. A derived class, called subclass or “son” keeps the methods and the attributes of the base class (or “father”). In practice, the class son is able to perform all the instructions that a class father is able to, plus others that the class father cannot perform. Moreover, it can define its own methods and attributes, and redefine the coding of some inherited methods through the overriding mechanism. When a class inherits from only one class, we have single inheritance. Otherwise, we have multiple inheritance. Python allows also multiple inheritance. The inheritance is used to extend and re use some parts of a code.

4.2.6. Encapsulation.

Besides inheritance, also encapsulation is a key aspect of object oriented programming. Encapsulation regards putting together in an object all data and actions that regard a given component. Another form of encapsulation is subdivide an application in several smaller objects.

A similar concept to encapsulation is information hiding. It allows to access to an object's informations only through the object's methods. This is performed only if we never allow having public attributes, despite particular exceptions for attributes of classes and constants. In order to access to attributes from externally, public methods are inserted which can be called to set or require the attribute value.

4.2.7. Polymorphism.

Polymorphism allows an object to assume several shapes. Polymorphism refers to the attitude of an object of showing several implementations for a single functionality. For example, we have a software system which intends to draw geometric figures. In this case, we have defined a class "father" called "geometric_figure" and three derived classes "rectangle", "circle", "oval". All these classes have the same method "draw_figure( )" Without polymorphism, it is necessary to perform a switch construct, which distinguishes the various cases of the various geometrical figures. Through polymorphism, all is reduced to the automatic recall of the method draw_figure( ) of the class "geometric_figure". If we want to draw a triangle, with polymorphism it is sufficient to create a new class "triangle" with the method draw_figure( ) and the rest comes out automatically.

4.2.8. Cohesion and coupling.

Coupling refers to the connections existing between two different classes of a programme. If two classes have several details in common, they are strongly coupled. Cohesion is an information on the quantity and heterogeneity of tasks whose a class or a method is responsible. In general, through the cohesion it is possible to establish the tasks that a class is designed for. If a class or method is responsible for only one task, the method has a strong cohesion.

In the simulator, each task is performed by a different method. However, some classes are strongly coupled, and this has been unavoidable given the specific output requested.

4.3. The modelization of the proposed transport system

The proposed transport system has been modelled according to an object-oriented logic, because it is the simplest and the most suitable to our modelization.

The steps followed in the development of the model have been:

1. Analysis of the problem and definition of the targets. The problem under study was the modelization of a car sharing system with specific strategies of management. The main targets of this work were to assess the performances of the proposed transport system, expressed in terms of level of service and of efficiency. For this purpose, and for the reasons exposed in chapter 3, it was necessary to know: the distribution of users' waiting times, the state of each vehicle during the simulation period.
2. Conceptualization of the model: A detailed description of the system which has been simulated is already provided in the chapter 3. A description on how the real system has been modelized will be provided in this section.
3. Development of the computer programme: which classes and methods have been used, which events have been chosen. A description on this point is also provided in this section.

Details on how the points 2 and 3 described above have been developed in the system under study, are provided in this section.

4.3.1. The main aspects taken into account

In order to develop a good model the following steps have been followed during the development of the code:

1. identification of classes and objects;
2. definition of the semantics of classes: at the end of this phase the structure of classes is clearly defined;
3. relationship among classes: it is necessary to define all the typologies of interrelations which exist among the various classes;
4. implementation of the code: during the implementation of the code some incoherencies can be shown and therefore it is necessary to briefly repeat all the previous steps and then continue coding.

The purposes of the modelization of the system under study, are to develop a model which is: simple, easy to understand, easy to modify, and versatile (i.e. applicable to several scenarios). In order to achieve all these issues, modularity and simplicity are key aspects. The object oriented approach allows to modify some parts of the code simply adding or removing methods, without having to rewrite completely the code, as it happens for non object-oriented programming.

The system under study has been modelized through event-based simulation. Despite the higher quantity of coding required than the simulation through processes, the event-based simulation is very easy and very versatile, because it easily allows to add or remove components of the system, and change the behavior of existing components, simply by introducing or removing classes or by introducing or removing events.

The simulator has also the target of being able to modelize any kind of scenarios. More in detail, the simulator must be able to be highly flexible in dealing with objects: more details about this last aspect will be provided in the following.

4.3.2. Object-oriented programming vs. simulation: objects and methods to modelize entities and events

In the development of the model, it has been decided to create the same classes in the code as the categories of entities involved in the system to be modelled. This choice has been performed in order to keep the shape of the code as close as possible to the reality to be modeled. The classes chosen to represent the system are:

- Picav
- User
- Parking, i.e. a station of the transport system
- Unit
- Supervisor, i.e. the system manager.

Each entity in the simulation has events associated; each class of objects has associated methods. In order to keep the code as simple and modular as possible, for each event of a category of entities, a module of the same class of objects has been developed. For example, besides the event "entrance user", of the entities "vehicles", i.e. the user enters a vehicle, a method "occupation" of the class of objects "vehicle" has been developed. Moreover, because in some events more tasks are performed, which often require to repeat some parts of code, in this case other methods are developed, all recalled by the main method which corresponds to the event. For example, to the event of the arrival of a vehicle at the destination station, corresponds a method "arrival vehicle": this method would have a high amount of tasks and repetitive instructions. Therefore, it recalls other methods to perform all the tasks. Another example of when an event is modelized through a main method which calls auxiliary ones, is the case in which distance needs to be calculated. In this case, some ad hoc methods to calculate the distance will be recalled by the main method.

Inheritance has not been implemented in the final simulator. In fact it has been developed in a first version of the simulator, i.e. the class "vehicles" inherited the class "users". However, this may constitute a problem because: firstly, as the user enters a vehicle, the vehicle inherits all the attributes and methods of the user, but this is not necessary, as only some of the user's attributes are necessary. Moreover, after the user enters the vehicle, all methods act directly on the vehicle but their results regard also the occupant, and therefore the occupant's attributes need to be modified coherently with the vehicle's ones. As a result, following this modelization it should be necessary that also the class user inherits some attributes of the class vehicle.

Besides the four classes exposed above, an auxiliary class has been implemented: the class “simulator”, which consists on the main code. This class has been developed in order to be able to recall the simulator from another code. This is important in particular in cases where the simulator is recalled to calculate the system’s performances. The main code also contains the time queue, from which some methods corresponding to events are recalled.

4.3.3. Permanent and temporary objects. Versatility of the output data provided by the code.

The objects are created outside the time queue, i.e. also the entities “users”, which are temporary, are implemented as permanent objects. Although this may seem poor programming, this choice has been made for several reasons. Firstly, the transport demand, provided in form of matrixes, must be assigned before the execution of the simulation: random components regard only the exact instant of arrive of the users; the time period in which the users arrive and the origin and destination of the users’ trips are fixed a priori. Moreover, the object user’s attributes must be kept also after the user exits the system in order to allow the simulator to provide any type of output data. In fact, according to the target of versatility that was required to the micro simulator, the programme is meant to be able to provide any typology of output data with minor modifications on the code. This is achieved because the simulator represents the second-by-second activity of each object of the system, and trace of this activity is kept in the object’s attributes.

4.3.4. Storage of objects in lists and dictionaries.

There is a problem which regards the creation of the objects in Pyhton and is typical of this language. In fact, there are two possibilities: or a fixed number of objects is created, for instance eight objects station, or each object is overwritten as a new object of the same class is created.

In the first case, no flexibility is given to the simulator, i.e. it is impossible to apply it to another scenario as the number of stations is kept the same. In this case, for example, an object is created as: p1 = station(), or p2 = station(). This avoids overwriting (the objects have different names) but only a predefined number of objects of a given class can be created.

In the second case, all the data about an object are lost as a new object is created. In order to avoid this drawback, objects are inserted in lists or dictionaries. In this case, for example, an object is created as: v = picav() inside a “for” cycle: as a new object is created, it has the same name and therefore is overwritten.

In order to resolve this drawback, all objects after creation are put into dictionaries, in particular a different dictionary for each class (e.g. we will have a dictionary “users”, another dictionary called “vehicles”, another called “stations”, etc.) and are referred through their name, which is actually their index, during the whole simulation. Lists instead are used to modelize queues at stations, or as auxiliary items for calculating output data. Details about lists and dictionaries are provided herewith.

The most versatile method to group data is the list, which can be written as a list of comma separated values (items) between square brackets. List items can be numbers, strings, and also objects. All operations can be done to the list, i.e. the elements of a list can be inserted and removed. Moreover, also the attributes of an object can be modified without the need of removing the object from the list. Lists are perfect to modelize queues, as their elements are not indexed. Lists can be also nested, therefore they can have other lists as elements. Dictionaries, instead, because their elements are indexed, cannot play the role of a queue.

Dictionaries are sometimes found in other languages as “associative memories” or “associative arrays”. Dictionaries are indexed by keys, which can be any immutable type; strings and numbers can always be keys. You can’t use lists as keys, since lists can be modified in place using index assignments, or methods like append() and pop(). This is a good feature of lists, which allows them, as stated before, to modelize efficiently queues. It is best to think of a dictionary as an unordered set of key: value pairs, with the requirement that the keys are unique (within one dictionary). The main operations on a dictionary are storing a value with some key and extracting the value given the key. It is also possible to delete a key - value pair with del. If you store using a key that is already in use, the old value associated with that key is forgotten. It is an error to extract a value using a non-existent key. The advantage of dictionaries is indeed in the usage of the key: in fact if an object is removed from the list, all the other objects change their index; if an object is removed from a dictionary, all the objects keep their index.

4.4. Inputs and outputs of the micro simulator

The simulator has been written to evaluate the system performance. The simulator receives in input: the simulation time period, the road network, the transport demand and the car-sharing system characteristics. The simulator allows to track the second-by-second activity of each user, as well as the second-by-second activity of each vehicle. The simulator gives in output the transport system performances, in terms of level of service (LOS) provided to users and in terms of efficiency from the management point of view. In fig. 4.1 inputs and outputs of the micro simulator are reported.

4.4.1. Input data

The input data described in the following are necessary to be able to run all the three system management strategies and to shift from a management scheme to the other, in order to compare their performances. In order to simulate only a given management scheme among the proposed ones, some data may result unnecessary.

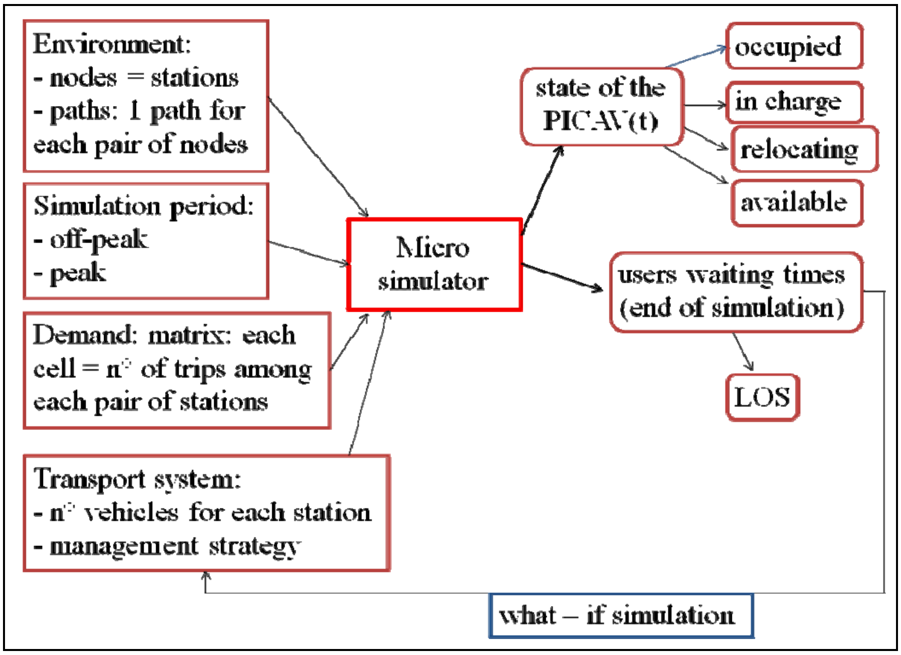


Fig 4.1. Inputs and outputs of the micro simulator

The simulation time period
The simulation time period usually starts when the transport system opens and ends when it closes. The simulation time period could be characterised by peak and off peak phases: for each phase an average pedestrian density *k* and a PICAV transport demand should be specified.

The road network
The road network includes stations, provided with charging stations, and the roads in which PICAV vehicles are allowed to travel. Stations have been represented through nodes. Each road is divided into sections and each section has been modelized again by a node. Between each pair of nodes, we take into account only one path, which could be the shortest one or the more attractive one since characterised by a high concentration of shops, museums and other activities. The characteristics of the path we are interested in are: its overall length and the average upslope. These data are necessary for the battery discharging law, in particular the quantity of discharge is assessed from the average upslope as it contributes heavily to the resistances to motion encountered by the PICAV vehicle. If the path is instead descending the recovery in battery charge is so slight that it is neglected therefore for each path the downslope parts are considered as flat in the calculation of the average upslope. The overall length of a path is required in order to determine the trip duration. The path lengths and the average upslope are assessed through Google Maps and Google Earth. These data are given as data

input in the simulator in the form of two matrixes. The matrixes are squared and the number of rows (or columns) equals the number of nodes in the network. In the first matrix, the cell *ij* represents the path length between the origin node *i* and the destination node *j*. In the second matrix, the cell *ij* represents the average upslope of the path between the origin node *i* and the destination node *j*.

The minimum charge level is calculated as a function of the average upslope and of the length of the more battery consuming path in the road network. It depends on the case of study and it has been calibrated for the Genoa scenario as equal to 10% of the total battery charge.

The transport system characteristics

The fleet dimension and the number of vehicles at each station at the beginning of the simulation time period should be specified.

The simulator allows a transport manager to choose the management strategy they want to simulate among the three previously described. According with his choice, the related transport system parameters have to be specified.

In case of the first system management strategy, the following parameters need to be specified:

- percentage of flexible users;
- choice sets of flexible users.

In case of the second and the third management strategies, the following parameters need to be specified:

- low critical, low buffer thresholds for each station in the form of two vectors;
- relocation technique: shortest time criterion or inventory balancing.

The transport demand

The transport demand is given to the simulator in the form of OD matrixes. OD matrixes are squared and rows/columns refers to nodes. Each cell gives the hourly number of trips by PICAV from the node (origin) the row refers to, to the node (destination) the column refers to.

In the second management strategy, OD matrixes have a number of rows and columns equal to the number of stations.

In the first management strategy, this is true for the trips of non-flexible users. For flexible users trips, the OD matrixes have a number of rows equal to the number of stations, and a number of columns equal to the number of choice sets.

In the third management strategy, OD matrixes have a number of rows and columns equal to the number of stations + the number of units. All the trips originated or attracted by a road section are considered concentrated in the node representing the section.

A trip by PICAV between two nodes could be a *direct trip* on board a vehicle, or a sequence of shorter trips (*multitask trip*) where one accomplishes a number of short tasks that require short term parking along the street, before finally returning the unit. In both cases what is of interest for the proposed study is the overall duration of the trip. Given an origin, a destination, the path between them, and an average pedestrian density, the trip duration changes according to the trip typology.

For each phase within the simulation time period the transport demand has been assumed constant. Therefore each OD matrix refers to a phase (peak - off peak) and to a trip typology (direct trip – multitask trip). In fig. 4.2 – 4.3 two demand OD matrixes are shown.

OD - type A off-peak period

	1	**2**	**3**	**4**	**5**	**6**	**7**
1	0	1	2	1	1	1	0
2	1	0	0	0	1	1	0
3	1	0	0	0	2	1	0
4	1	0	0	0	1	1	0
5	1	1	2	2	0	1	0
6	1	1	2	1	1	0	0
7	0	0	1	1	0	0	0

Fig. 4.2. The demand OD matrix for type A trip (trip chains with origin and destination on the border of the area) in an hour of the off-peak period. Origin and destination are both stations.

OD - type B off-peak period

	1	**2**	**3**	**4**	**5**	**6**	**7**	**10**	**11**
1	0	1	1	0	0	1	0	1	1
2	0	0	0	0	0	1	0	1	1
3	1	0	0	0	1	2	1	3	2
4	1	0	0	0	0	1	1	3	2
5	0	1	1	0	0	1	0	2	1
6	0	1	1	0	0	0	0	2	1
7	0	0	0	0	0	0	0	1	0

Fig. 4.3. The demand OD matrix for single trip in an hour of the off-peak period. Origin and destination are both stations.

In the simulation users are generated with the following characteristics: the origin of their trip by PICAV, the destination, the time at which they appear in the origin and the duration of the trip by PICAV. These data are assessed according to the OD matrixes. The time at which a user appears in their origin is randomly generated: if X users have to be generated between 8 and 9 a.m. in a given origin, X casual numbers are extracted within the given time interval and these casual numbers are the exact arrival instants of the X users in the origin.

4.4.2. Output data

Level of Service (LOS)

LOS measurement are assessed, based on the statistical distribution of waiting times. Castangia and Guala (2011) proposed a scale using as a reference the 50th , 90th and 95th percentile of users waiting times: the scale is provided in table 4.1. All conditions defining a specific LOS should be met.

LOS measurement could be assessed for each station or for the overall area, referring specifically to the waiting time of the users arriving in each station or to the waiting time of all the population of users.

Table 4.1. LOS measurement based on the statistical distribution of waiting times. (Castangia and Guala, 2011).

LOS	Waiting time (minutes) not greater than:		
	50th percentile	90th percentile	95th percentile
A	0.5	1	1.5
B	1	2	3
C	1.5	3	5
D	2.5	5	8
E	4	8	10
F	worse	worse	worse

Efficiency
From the management point of view, the transport system efficiency is inversely proportional to the fleet dimension and the number of required relocation trips. Another measure of the system efficiency is the ratio between the number of vehicles available or occupied by users at each time instant and the fleet dimension. Further details on the simulator are available in the section 4.5.

4.5. Description of the micro simulator

The simulator of the proposed transport system is: based on discrete events, and it follows an object-oriented logic. It modelizes in detail the instant by instant activity of each user and of each vehicle. In the figure 4.4 the general structure of the micro simulator is reported.

The simulator is therefore a valid instrument for a “what if” analysis of the transport system, given the main characteristics of the system, of the demand, and of the transport system offer.

The Python simulator code contains also some sub models, for example: the model for battery charge and discharge, a model which relates the PICAV vehicle speed with the pedestrian density.

The simulator of the transport system is characterized of the following classes:

1. user
2. PICAV vehicle
3. station
4. unit
5. supervisor

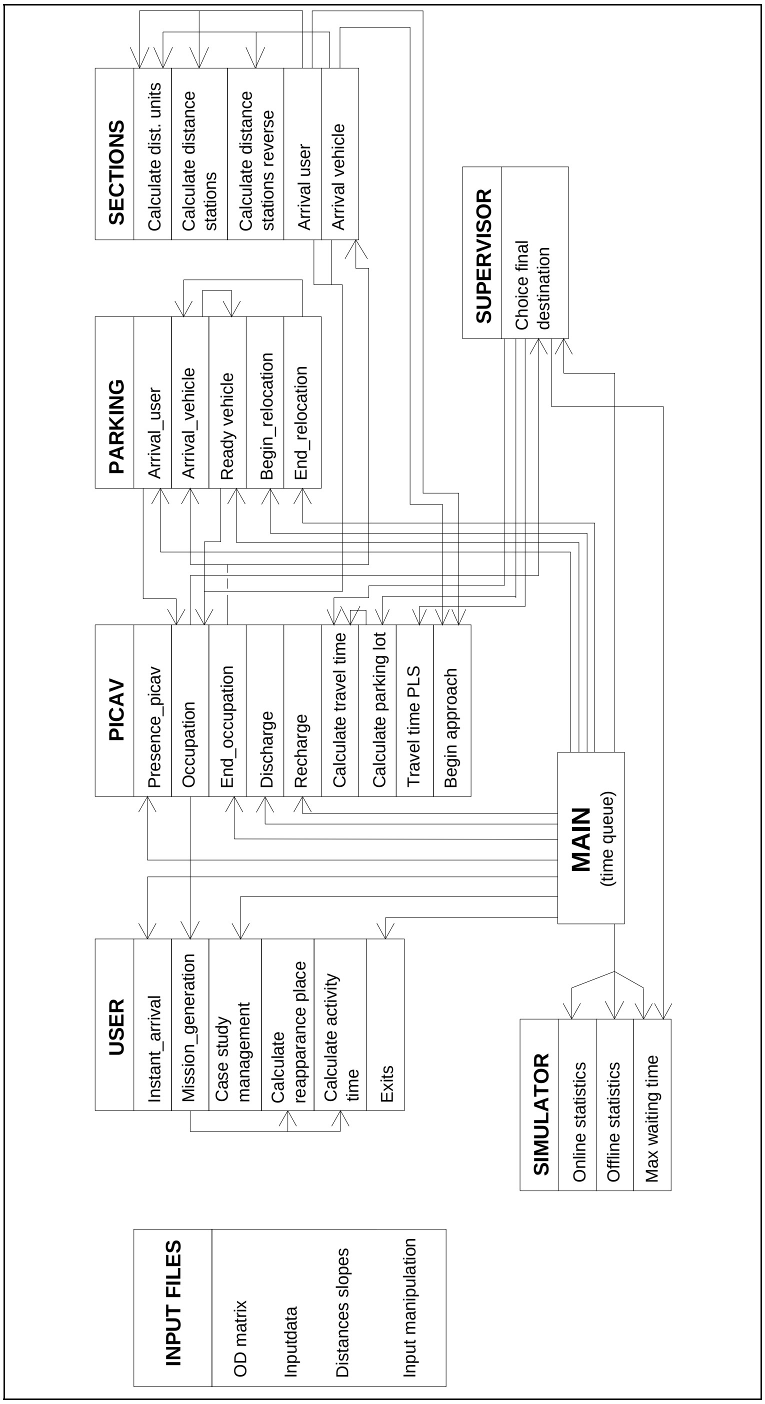


Fig. 4.4. Scheme of the micro simulator structure. In bold the name of classes is reported. Below each class name, the methods are written.

The simulator is composed of:

1. A main file called *simulatore*, which contains a main method called *simulazione*
2. several input files, in particular input files related to the transport demand and to the network. They are auxiliary files and their purpose is to make the simulator as easy as possible. In particular:
    a. inputdata.py: it contains all the general input files, e.g. the duration of the simulation period, organization of the simulation period, number of stations, number of units, thresholds, dimension of the fleet at the beginning of the simulation, etc.
    b. distances slopes.py: contains distances and slopes between each pair of stations, each pair of units, each pair of station and unit and vice versa.
    c. OD matrix.py: contains the OD matrixes of the transport demand, for each trip typology, and for each period of the day.
    d. Input manipulation.py: it fits the simulator inputs in order to make them easier readable.

In the following, all the parts constituting the simulator will be described in detail. In Attachment A of the thesis, a detailed scheme of all methods in the micro simulator is provided.

4.5.1. The file inputdata.py:

The file *inputdata.py* provides:

- Management strategy: it is a number which can be assume the following values:
    1 = vehicles fully automated, available only at stations.
    2 = flexible users, vehicles manually driven.
    3 = automated vehicles available also along the roads of the study area.
- Duration of the simulation and its subdivision into periods. In particular:
    1. Maximum duration of the simulation
    2. Beginning instant of the first period of the simulation
    3. Beginning instant of the second period of the simulation (it is assumed that the first period ends in the pervious instant)
    4. End instant of the second simulation period.

  Three periods are assumed because PICAV vehicles mainly used for leisure reasons, therefore to reach shops and museums etc.: we consider a first off-peak period, in which everyone is still at work, and a second peak period, in which the PICAV is much used. For example in the Genoa case study, it has been assumed that the simulation day begins at 8 am and ends at 0 am, and the following division in periods: a first off-peak period ranging from 8 am to 4 pm, and a second peak period ranging from 4 pm to 8 pm. In the period from 8 pm to 0 am, there is no longer demand but users already in the system terminate their trips and trip chains. In the Barreiro case of study, it has been assumed that the first period ranges from 7 am to 1 pm, and the second period from 1 pm to 7 pm. The third period, without demand, ranges from 7 pm to midnight.

- Pedestrian density:
    1. Pedestrian density in the first period (morning)
    2. Pedestrian density in the second period (afternoon)
    3. Pedestrian density in the third period (evening)
- Low critical thresholds
- Low buffer thresholds
- High critical thresholds
- High buffer thresholds
- Station capacities
- Number of vehicles at the beginning of the simulation period, and their distribution among the various stations. Thresholds, station capacities, number of vehicles at the beginning of the simulation period, are given in form of arrays, in which, the position in the array is related to the identifier of the station.
- Number of stations
- Number of units
- Duration of the activity time. The time that is necessary for the user to perform their activities can be determined in two alternative ways:
    1. The activity time is assumed proportional to the time necessary for a user to perform a direct trip from the user origin to their destination. For example, if the user has origin at the station 2 and destination at the station 7, and performs an activity chain, the time necessary for the user to perform all activities is assumed proportional to the travel time.
    2. The activity time is assumed independent from the origin and the destination of the user, because the time necessary to perform the various trips is considered negligible with respect to the time necessary to perform the activity themselves. Therefore input data are the average and the standard deviation of the activity time.
- Activity time. The activity time can be provided as proportional to the travel time through an activity time coefficient; or it can be provided as the average of the activity time plus a deviation:

  Activity time = k · travel time = E [activity time] ± σ.
    - Activity time coefficient. If the activity coefficient is different from 0, therefore the activity time is proportional to the travel time with a coefficient equal to 5 in Genoa case study.
    - Average duration of activity time. This input and the following (the deviation) are read by the simulator only if the activity time coefficient is greater than zero.
    - Deviation of activity time.

4.5.2. The file OD matrix.py:

The file OD-matrix.py contains all matrixes for each of the two time periods of the day considered, and for each trip. In particular, 4 typologies of trips are considered:

A. Activity chain from an origin to a destination, both on the border of the study area.
B. Single trip from the border to inside the intervention area, or vice versa
D. Single trip having both origin and destination inside the intervention area
E. Trip chain having both origin and destination inside the intervention area

These trip typologies are already schematized in the figure 3.1.
For example:

```
OD A morning = [[0,1,2,1,1,1,0],
                [1,0,0,0,1,1,0],
                [1,0,0,0,2,1,0],
                [1,0,0,0,1,1,0],
                [1,1,2,2,0,1,0],
                [1,1,2,1,1,0,0],
                [0,0,1,1,0,0,0]]
```

Each row of the matrix corresponds to a station of origin, each column to a destination station. For example, the number "2" in the 5th row and 3rd column, means that 2 users in an hour have origin in the station 5 and destination in the station 3.
Moreover, it contains:

- A matrix called *choice sets*, therefore: each row corresponds to a choice set, each element of the row is the identifier of a station belonging to the choice set. For example, the number 3 in the second row means that the station 3 belongs to the 2nd choice set.

  ```
  choice sets = [[1,2,3,4,6],
                 [1,2,3,4,5,6,7],
                 [1,2,3,4]]
  ```

- Because both type A and type B trips may correspond to flexible users, for the quota of demand related to flexible users, we have OD A choiceset and OD B choiceset for each considered time period.

```
OD A choiceset morning = [[1,1,1],
                          [1,1,1],
                          [3,2,1],
                          [3,2,1],
                          [2,1,1],
                          [2,1,1],
                          [1,1,1]]
```

Each row corresponds to a station of origin, each column to a destination choiceset. For example, the number 3 in row 4 and column 1 means that 3 users by hour have origin in the station 4 and destination in the station 1.

4.5.3. The file distances slopes.py:

The file *distances slopes.py* contains the distances between: each pair of stations, each pair station-unit and vice versa, and each pair of units. We have therefore the following matrixes:

- Distances stations
- Slopes stations
- Distances points points
- Slopes points points
- Distances points stations
- Slopes points stations
- Distances stations points
- Slopes stations points

```
distances stations = [[0,1060,976,1200,1400,604,1100,489,1060],
                 [1060,0,316,846,1180,732,1100,489,747],
                 [976,316,0,414,516,569,829,563,326],
                 [1200,846,414,0,481,848,1080,866,379],
                 [1400,1180,516,481,0,834,804,1050,372],
                 [604,732,569,848,834,0,678,458,464],
                 [1100,1100,829,1080,804,678,0,961,610],
                 [489,489,563,866,1050,458,961,0,670],
                 [1060,747,326,379,372,464,610,670,0]]
```

The matrix *distances stations* has therefore: each row corresponds to a station of origin, each column to a destination station, the number reported is the distance in metres from the origin to the destination.

The matrix *slopes stations* has rows and columns with the same meaning as distances stations. The slope calculated is the average slope, and the parts down slope are considered as flat. Indeed, on one hand the battery of the vehicle can be recharged in the downslope parts, and the electric motor is used as a generator; on the other hand, the energy cumulated in this way is negligible, and to neglect this energy is in favour of safety.

```
slopes stations = [[0,2.28,1.95,2.17,0.23,0.60,0.0,4.95,1.95],
                [0.0,0,0.0,0.81,0.0,0.0,0.0,1.36,0.0],
                [0.0,1.03,0,2.16,0.50,0.0,0.72,2.08,0.38],
                [0.0, 0.81,0.0,0,0.42,0.0,0.0,0.42,0.0],
                [0.23,0.05,0.50,0.42,0,0.0,0.0,0.35,0.42],
                [0.60,2.16,2.75,3.96,2.69,0,0.0,5.04,3.26],
                [0.17,0.69,0.72,2.49,2.97,2.68,0,2.65,3.21],
                [0.0,0.36,0.0,0.42,0.35,0.0,0.65,0.0,0.93],
                [0.95,0.28,0.38,1.35,0.42,0.0,0.0,0.93,0]]
```

4.5.4. The file attractivity units.py:

In the management strategy with flexible users, the call to the supervisor occurs in correspondence of the last stop of the trip chain. The unit in which the call to the supervisor takes place is determined through a random extraction, in which the probability of each unit to be extracted depends on its attractivity. The attractivity of each unit is given in the file *attractivity units.py*.

4.5.5. The file input manipulation.py:

The file *input manipulation* is composed of two parts: a first part in which the dictionary of the transport demand is created, and another in which the lists related to the distance and to slopes between each pair of stations or of units are created. The simulator's users, will inherit the data stored in the dictionary of the transport demand. To better modelize the transport demand, auxiliary objects "demand" are created. To each object is assigned:

- origin;
- destination;
- indicator, whether it performs a single trip or a trip chain;
- typology of user: 1 if the origin is on the border of the intervention area and 2 if it is inside;
- destination in case the user is flexible: if the user is not flexible, this attribute is set equal to 999999; if it is flexible, the attribute is set equal to the user choice set.

The demand is assigned in a deterministic manner: i.e. each user has an origin and a destination, or a choice set in case they are flexible. The stochastic part regards: if the user is flexible, the unit in which the user reappears in order to make a call to the supervisor; and, for all users, the exact instant inside the hour in which the user enters in the system. The stochastic components however are assigned in the simulator's main file.

For what regards distances and slopes, some auxiliary objects "distances" are created, with the following attributes:

- origin
- destination
- value.

Each object is actually a pair of distances, or of slopes. The objects are afterwards inserted in the lists:

- Distances stations
- Slopes stations
- Distances points stations
- Slopes points stations
- Distances stations points
- Slopes stations points
- Distances points points
- Slopes points points

4.5.6. The file Simulatore.py:

The file *Simulatore.py* is recalled by the file *chiamasimulatore.py*, it creates the object *simulator*, then recalls the method *simulation*. Below, all the classes of the file Simulatore will be described.

4.5.7. Class Simulator

The class Simulator is the main class, which contains all the main attributes and commands of the simulator. In particular it contains the queue time, in which all the different events, schematized through methods, will be recalled.

The first part of the class Simulator regards the declaration of the class attributes. Among them we have:

- The main dictionaries:
    - users = dictionary of users
    - vehicles = dictionary of vehicles
    - stations = dictionary of stations
    - units = dictionary of units.
- The lists which modelize the queues:
    - queue = queue of users waiting at the stations,
    - queue units = queue of users waiting at the units.

  They are both nested lists. For example, the list *queue*, in the Genoa case study, in which there are 9 stations, contains 9 sub-lists: each sub-list contains all the users currently in queue in a given station.
- Some counters and auxiliary lists necessary to calculate the statistics.

Method Simulation

In the first part of the method, the quantities previously declared are initialized. In particular, all lists, counters and dictionaries are set equal to zero. This aspect is important because when the simulator is recalled several times, e.g. in an iteration of the optimization algorithm described in chapter 5, without this command the risk is to have lists and dictionaries not empty, and therefore the simulator outputs report severe mistakes.

Then, in the method Simulation, all the general quantities declared in the input files are imported: for example, thresholds, or the number of available vehicles at each parking lot, the activity pattern coefficient, etc.

Afterwards, the PICAV vehicles speed is calculated. The vehicles velocity is calculated in a simplified way, and it is assumed the same in all roads of the intervention area, and the pedestrian density is also assumed the same in all roads. In the Genoa case study, the chosen density is the one registered in via San Luca, one of the most crowded roads of the historical city centre. The velocity of vehicles is calculated in the three periods of the day, i.e. morning, afternoon and evening (i.e. when there is no longer transport demand but users terminate their trips and activities). Two velocities are calculated in each period, i.e. when vehicles are user driven and when they move automatically.

Afterwards, the objects *stations* are created. At each station it is assigned: the number of vehicles defined in the input data, the station capacity, and the four thresholds. Actually, besides the number of vehicles, the stations have another attribute, i.e. *n_veh_relocation*, which is an integer that provides the number of vehicles currently relocating. This attribute is introduced because: when a station is in shortage of vehicles, it recalls a vehicle to colmate this deficit. Because the vehicle takes a certain time to perform its relocation trip, the receiver station will continue at each simulated time instant to recall vehicles until it is reached by the first vehicle. The result is that the station previously in shortage of vehicles will have a great number of available vehicles, while other stations will be in shortage in a following time. This leads to a severe increase in the total relocation time.

Then, the objects *units* are created. To the units the attractivity is assigned, and a number of available vehicles initially set equal to 0. It is assumed that at the beginning of the simulation all vehicles are at the stations after being in charge during all the night. Details on the law of charge and discharge of the battery will be provided in section 4.7.2.

The object *supervisor* is also created. It will be described in details later in this section.

Afterwards the objects *user* are created. The attributes of these objects, i.e. origin, destination, provenience, activity pattern indicator, typology, are imported from the class demands previously cited. Hour by hour, the objects user are created, and they import the proper attributes from the class demands. Finally, the method **instant arrival** of the class user is recalled. This method, given the hour in which the user will arrive at the station of origin, assigns to the user the exact minute of arrive.

Finally the objects PICAV are created. The method **presence picav** of the class picav is recalled: this method assigns to each vehicle the station in which the vehicle will be at the beginning of the simulation.

The time queue follows. All the following instructions are contained in the time queue, and therefore will be repeated at each simulation instant.

Firstly a check is performed, whether some vehicle has terminated its trip or trip chain in the current simulated time instant. If so, the method **end occupation** of the class picav is recalled.

Afterwards, a check is performed, if some vehicle has terminated its re direction trip in the current simulated time instant. In this case, the method **end redirection** is recalled.

The vehicles redirection occurs in these two cases:

- If the destination station is full, i.e. it has reached its capacity, the vehicle is redirected to the nearest available station
- If the vehicle has an unit as destination and its level of charge is below the minimum, then it is redirected to the nearest station to recharge. However, this type of situation never occurs in the practical cases because of a quite good battery management and of a high autonomy of the battery. This

situation has been artificially tested through introducing in the simulator a battery with very low capacity, in order to test the performance of the algorithm.

After, a check is performed whether some users waiting for a vehicle in the units have been reached by the vehicle, at the current time instant. In this case, the method **end occupation** of the class picav is recalled.

Afterwards, the code checks if some vehicle has finished to relocate. In this case, the method **end relocation** of the PICAV vehicle is recalled.

The code then controls whether the user has terminated the last activity in their trip chain. In this case, the method **choice final destination** of the class Supervisor is recalled.

The code then updates the state of charge of vehicles. More in details, if the vehicle is moving, the battery is discharged. If the vehicle is idle at an unit, the battery is neither discharged nor recharged. Otherwise, the battery is recharged. Two typology of charging law have been proposed, and more details are provided in section 4.7.2. If the battery is currently being discharged, the method **discharge** is recalled. Otherwise, the method **recharge** is recalled.

After, a check is performed whether some vehicles have enough power to relocate or be available to users. If the charging law is opportunity charging, the method **ready vehicle** is recalled, of the station in which the vehicle is available.

After, a check is performed whether some user enters in the system in the current time instant. In this case, the method **arrival user** of the origin station of the user is recalled

Finally, a check is performed whether the station in exam has shortage or excess of vehicles. When one of these two situation occurs, the method **begin relocation** of the station in exam is recalled. In the method, these two cases are treated differently.

Method online statistics

In this method, the statistics related to each simulated time step are calculated. This method is recalled by instructions placed inside the time queue. In each simulated time instant the following is calculated:

- Number of users in queue in each station
- Number of available vehicles in each station
- Number of vehicles in each state: available, occupied, in charge, relocating, redirected.
- Number of required relocations and number of required relocations that have taken place.

Method offline statistics

This method is related to statistics calculated at the end of the simulation time. In this method, the distribution of users waiting times is calculated, and therefore the average waiting time and the 50th, 90th and 95th percentiles of the distribution.

Method calc max waiting
This method is recalled in order to calculate the maximum waiting time. It calculates for each station the maximum waiting time at the current time instant.

4.5.8. Class User
Declaration of the attributes:
- Name
- Arrival time instant
- Waiting time
- Time of entrance in the queue
- Destination
- Activity pattern indicator
- State: it is equal to: 0 if the user still has to reach the origin station/unit; 1 if they occupy a vehicle; 2 if they are in queue at a station or unit; 3 if they have exited the system.
- Parking: identifier of the station in which the user is currently in queue.
- Unit: identifier of the unit in which the user currently is
- Pikav: identifier of the vehicle occupied by the user
- Typology of the user: 1 if the user has their origin on the border of the intervention area; 2 if their origin is inside the intervention area but the destination is on the border; 3 if both the user origin and destination are inside the intervention area.
- Identifier of the queue in which the user has waited.
- Identifier of the unit from which the user calls the supervisor
- End of travel time instant, end of activity pattern time instant, supervisor call time.

Method instant arrival
Firstly the arrival time is generated according to the user identifier. After, it extracts the arrival time instant within the hour in a random way inside an uniform distribution.

Method mission generation
If the user has an activity travel pattern, they recall the method **calculate activity time**, that will be described in the following, in order to calculate their activity time. If the user is flexible, the method **calculate reapparance place** is recalled, which determines the unit from which the user will call the supervisor.

If the user does not have an activity travel pattern, then only the travel time is calculated, from the distance between the origin and the destination and from the vehicle speed in a given time instant.

Method calculate reapparance place
It determines the unit from which the flexible user calls the supervisor. The identifier of this unit is extracted in a random way. The probability to perform a call from a given unit is proportional to the unit attracivity.

Method calculate activity time
It calculates firstly the travel time and after the activity time. The activity time is determined: or multiplying the travel time by the activity coefficient, or through extracting the activity time duration from a normal distribution with average and standard deviation defined in input.

4.5.9. Class Picav

Firstly the following attributes are declared:

- name
- parking: identifier of the station in which the vehicle currently is
- unit: identifier of the unit in which the vehicle currently is
- user: identifier of the user occupying the vehicle
- origin, destination, provenience
- travel time, activity time, supervisor call time
- state:
  - 0 if the vehicle is available, either at a station or at a unit. If the vehicle is in a station, it is put in charge,
  - 1 if the vehicle is occupied by a user,
  - 2 if the vehicle is in charge because the charge level is less than the minimum: therefore the vehicle is unavailable for user or relocation trips,
  - 3 if the vehicle is relocating between two stations or if it is reaching the user position
  - 4 if the vehicle is redirected: or to a station in order to recharge the battery, or to the nearest station because the destination is full.
- level charge: i.e. the battery charge level
- attributes necessary to calculate the discharge and recharge of the battery: i.e. the engine power, the battery capacity, the vehicle maximum velocity; other attributes, which are imported from other classes, and are: the average slope of the portion of the road, the initial and final battery charge level (i.e. at the beginning and at the end of the charging), are initialized.

Method presence picav:
It assigns to each vehicle the station in which it will be at the beginning of the simulation. It updates the number of available vehicles at each station.

Method occupation:
Occupation of the vehicle by the user. Firstly it recalls the method **mission generation** of the user, in order to calculate the user last attributes. Afterwards, the PICAV vehicle inherits all the attributes of the occupant. Moreover, the method updates the attributes of the vehicle and of the user, which are related to the identifier of the occupant. Moreover, it updates the attributes of the vehicle and of the user, related to the identifier of the occupant (attribute *user* of the vehicle) and of the occupied vehicle (attribute *pikav* of the user).

Method begin approach
Or the begin of the approach of the vehicle to the user: the travel time is assigned to the vehicle, more precisely the instant in which the vehicle will reach the unit occupied by the user.

Method end occupation
It recalls the method **arrival vehicle** of the vehicle destination station or unit.

Method end redirection
It also recalls the method **arrival vehicle** of the vehicle arrival station.

Method discharge
This method implements the battery discharging law. The quantity of battery discharged is determined according to the vehicle resistances to motion.

Method recharge
The battery is recharged. Two cases are distinguished, according to the initial and final level of charge of the vehicle.

Method travel time pls
Calculate the travel time between two stations
Method calculate travel time
It calculates the vehicle travel time according to the distance between the origin and the destination, and to the velocity. In this case, the calculated travel time is between an unit and a station.

Method calculate parking lot
Recalls the method **calculate travel time** or **travel time pls** to determine the destination station.

4.5.10. Class Parking

Method arrival user
The user arrives at the station. If there is an available vehicle, the user enters the vehicle. The method **occupation** of the class picav is recalled. Otherwise, the user enters in the queue.

Method arrival vehicle
The vehicle arrives at the station. If the station capacity is not reached, then the following happens. If the vehicle has a battery capacity less than the minimum, it is put in charge. Otherwise:

- if there is an user in queue, the user enters in the vehicle; the method **occupation** of the class picav is recalled
- if the management strategy is different from the 3rd, i.e. vehicles can be accessed only in correspondence of the stations, then the vehicle remains in the arrival station and the number of vehicles available in the station is updated.
- Otherwise, if the management strategy is the 3rd, the method checks if there is any user waiting at the current unit. In negative case, the method checks if there is an unit or a station lacking of vehicles. In this case, the vehicle is sent to this station or unit.

If the destination station has reached the capacity, the vehicle is sent to the available nearest station. It never happens that all stations in the transport system are at the capacity, therefore it is always possible to find an available station within the system.

Method ready vehicle
If the vehicle has enough charge for performing the longest trip chain and in the station there are no users in queue, then the number of available vehicles is updated. Otherwise, the first user is extracted from the queue and the method **occupation** is recalled.

Method begin relocation
Firstly in the method the two critical situations are distinguished, i.e. ZVT (= zero vehicle time, no vehicles are available) and FPT (= full port time, the station has reached the capacity). On the basis of the critical situation, it is evaluated whether there are stations able to receive or to provide vehicles. If this is the case, the supporting station is determined, and afterwards the counter of vehicles in the two stations is updated. Finally, the method determines which vehicle will be delivered. Of this vehicles, all attributes are updated, and in particular the travel time.
If no station is able to provide or receive vehicles, then: if the management strategy is different from the third one, i.e. vehicles are accessible at the stations, then: the

vehicle is sent to the nearest unit in case of FPT, and in the case of ZVT a request to the nearest unit is sent.

Method end relocation.
The vehicle attributes are updated, because the vehicle has reached its destination. The method **arrival vehicle** of the class picav is recalled.

4.5.11. Class Supervisor

Method choice final destination
This method implements the redirection of the flexible users performed by the supervisor. Firstly it recalls the method **calculate parking lot** of the vehicle which performs the supervisor call, in order to determine the distances, from the point in which the vehicle performs the call, to all the stations of the user choice set.
Afterwards, the method **calc max waiting** of the simulator is recalled, which calculates the maximum waiting time at the station. The destination station is afterwards determined, then the attributes related to the destination and to the travel time of the vehicle and of the occupying user are updated. The criteria used by the supervisor to allocate vehicles is provided in the chapter 3.

4.5.12. Class Sections
Sections is the name given to the units.
Firstly as usual the attributes of the class are declared: name, number of vehicles.
Method calculate distance unit. Calculates the distance between two units.
Method calculate distance stations. Calculates the distances from a station to an unit.
Method calculate distance stations reverse. Calculates the distances from an unit to a station.

Method arrival user.
Arrival of an user to an unit

a. If there are some vehicles available, the user occupies a vehicle and calls the method **occupation** of the vehicle itself.
b. Otherwise, the user enters in the queue and, if there are some vehicles available in the intervention area, he recalls the neaest available vehicle. He recalls the method **calculate distance unit** of the current unit in order to calculate the distance from the current unit to the others of the area; and the method **calculate distance stations** to calculate the distance from the current unit and the stations of the study area. After determined the miser station or unit, the number of available vehicles of the station or unit sender is updated, and it is determined which vehicle will reach the user position. The method **begin approach**, of this vehicle is recalled.

c. If instead there are no available vehicles anywhere, the user is set into the queue and no other instruction is executed.

Method arrival vehicle.
It determines the arrival of a vehicle at an unit.

a. If the charge level of the vehicle is below the minimum, then it is redirected to the nearest station. Afterwards, the method **calculate distance station reverse** of the unit under study is calculated in order to determine the nearest free station, and then the travel time from the current unit to that station. Finally, the travel time and the redirection time of the PICAV vehicle are updated.
b. If the charge level is above the minimum:
    a. If there are some users in queue at the current unit, the first user in queue enters in the vehicle. The method **occupation** of the vehicle in exam is recalled.
    b. If there are no users waiting at the current unit, a request is addressed to the surrounding units and stations. Firstly the units are checked, then the stations, because obviously there will be many more available vehicles at the station rather than at units.
        i. If there are users waiting at an unit, the vehicle is sent to the user who is waiting for longer. For all users waiting at an unit, the waiting time is calculated. As the receiving unit is calculated, the distance and the travel time are calculated and the method **calculate distance unit** of the current unit is recalled. Finally, the method **begin approach** of the current vehicle is recalled.
        ii. If there are no users waiting at an unit, the code checks if there are any waiting at some station. If so, the station with the highest waiting time is determined and the method **calculate distance stations reverse** of the surrent unit is recalled. Finally, the method **begin approach** of the current vehicle is recalled
        iii. If there are no users waiting anywhere, the vehicle remains in the current unit and the number of vehicles available at that unit is updated.

4.5.13. The file chiamasimulatore.py:

This file recalls the simulator, creates an object *simulator* and recalls the method *simulation* of the object itself. Then it provides the outputs that we desire.

The simulator in its current version is developed in order to interact with the GUI developed by the University College of London (UCL). This GUI (Graphical user interface) recalls in input from the simulator some text files, in format csv, which are produced from the simulator. A brief scheme of this is shown in the figure 4.5. In particular, *chiamasimulatore* produces in output the following files:

- PP.csv. This file provides, for each station and for each simulated time instant, the state of the station in terms of: number of available vehicles, length of the queue and maximum waiting time. It is composed of 5 columns, which are respectively:
  - Identifier of the station
  - Time instant
  - Number of available vehicles
  - Number of users in queue
  - Maximum waiting time
- Users.csv. It provides the distribution of users waiting times, i.e., the waiting time for each user and for each station. It is composed of three columns:
  - Identifier of the user
  - Waiting time
  - Identifier of ths station in which the user has been in queue (9999999 if the user has waited at an unit, i.e. along the road)

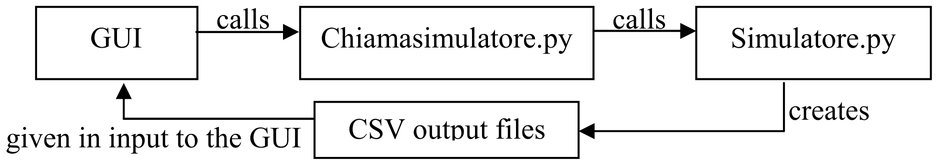


Fig. 4.5. Scheme of the interaction between the simulator and the GUI.

- Picav.csv. Provides the state of vehicles at each simulated time instant. It is composed of the following columns:
  - Time instant
  - Number of available vehicles
  - Number of vehicles occupied
  - Number of vehicles under charge
  - Number of vehicles relocating
  - Number of vehicles redirected

4.6. The simulator debugging.

The simulator has been completely verified in order to avoid typing errors or more conceptual ones. Below, a detailed description on the tests performed on the simulator.

4.6.1. First simulator debugging.

Firstly, the simulator code has been modified in order to print on the computer screen all the second-by-second activity of all objects and entities involved in the simulator study. This modification makes the simulator work very slowly as a big amount of pages are written on the screen. Each object has associated an identifier (a "name") in order to allow follow it along the simulation. Randomly, some users and some vehicles have been followed during their lifetime in order to assess that they behave properly. For example, to check if any vehicle was lost, or that any user kept on with their trip or trip chain for an anomalous amount of time. All the checks have been performed on both Genoa and Barreiro scenarios. The checks that have been performed on the whole were the following.

Check on the vehicles.

Some vehicles have been followed during their entire "lifetime" in a simulation day, i.e. from the beginning of the day, in which they were idle at stations, to the end of the day, in which they were supposed to be again idle at stations. At a first stage, all the simulator parameters have been set equal to their ordinary values, regarding e.g.: the demand, the battery lifetime, the network characteristics, etc.

- All the events associated to each test vehicle have been checked if they happened in the proper time. For example, if e.g. at time 8:55 the vehicle began occupation and was supposed to end its trip at 9:01, it was checked if at time 9:01 that vehicle really finished its trip. The same check is performed also for relocation trips.
- The duration of trips and trip chains was also checked: all trips and trip chains durations had to be reasonable, e.g. it couldn't take a vehicle more than 1 hour to reach Piazza Caricamento from Piazza de Ferrari.
- The indicator of the vehicle state had to change properly: i.e. it had to become 0 (available) or 2 (in charge) as it become empty, and be equal to 1 (occupied) when the vehicle was occupied by the user.
- When the vehicle becomes empty and it is not in charge, it must be available to users. If in the station there are some available vehicles, users cannot stay in queue.
- When a vehicle is occupied, it cannot be available and it cannot be accessed by an user. Moreover, the vehicle cannot be at the same time available at a station and occupied by an user.
- If the vehicle is occupied, its attribute "user" contains the identifier of the occupying user. At the same time, the user attribute "picav" must contain the identifier of the right vehicle.

Check on the supervisor's assignment.

If the management strategy is the flexible users one, a check is performed if the supervisor really assigns the vehicles to the station according to the proper criteria:

- if there are some users in queue, the best station is the one for which the ratio between maximum waiting time and distance from the user position is maximum
- if there are no users in queue, the best station is the one for which the product between number of vehicles available and distance from the user position is minimum.

Check on the stations.
A check is performed on the stations, whether all the indicators are in the proper way and coherent. This check is performed in all the stations but in a random time instant for each station. In this case, as the "problems" usually arise as the simulation goes on (they rarely arise at the first simulation instants), the majority of checks are performed between 4 p.m. and midnight.

- Firstly, if a vehicle is available at a station, it must be counted within the station vehicle. If a vehicle arrives at a station, the total number of vehicles available must be increased by 1. If a vehicle leaves a station, this number must be decreased by 1.
- If an user arrives at a station and is in queue, the list "queue" related to the station under study must be updated. The same is when a vehicle becomes available and an user exits the queue.
- A check is also performed for relocations: that really the station which is in most defect of vehicles requires the relocation, and that really the nearest valid supporting station provides the required vehicle. This is for the ZVT (zero vehicle time) occurrence: the FPT never happens in normal situations and therefore it is not checked at this stage
- If there is a user waiting at an unit, and the current station is the nearest one able to send vehicles, it is checked that this effectively happens.

Check on the units.
A check is performed on the units, whether all the indicators are in the proper way and coherent. This check is performed in all the units but in a random time instant for each unit. In this case, as the "problems" usually arise as the simulation goes on (they rarely arise at the first simulation instants), the majority of checks are performed between 4 p.m. and midnight.

- Firstly, if a vehicle is available at a unit, it must be counted within the unit vehicle. If a vehicle arrives at a unit, the total number of vehicles available must be increased by 1. If a vehicle leaves a unit, this number must be decreased by 1.
- If there is a user waiting at an unit, and the current unit is the nearest one able to send vehicles, it is checked that this effectively happens.

Check on the users
Some checks are performed randomly on users.

- Firstly, a coherence check is performed: if the vehicle is occupied, its attribute "user" contains the identifier of the occupying user. At the same time, the user attribute "picav" must contain the identifier of the right vehicle.
- All the events associated to each test user have been checked if they happened in the proper time. For example, if e.g. at time 8:55 the user began occupation and was supposed to end its trip at 9:01, it was checked if at time 9:01 that user really finished its trip.
- The duration of trips and trip chains was also checked: all trips and trip chains durations had to be reasonable, e.g. it couldn't take a user more than 1 hour to reach Piazza Caricamento from Piazza de Ferrari.
- The indicator of the user state had to change properly: i.e. it must be equal to 1 (occupied) when the user occupies a vehicle, 2 (in queue) if the user is in queue at a station, 3 (out of system) if the user has finished their trips and has exited the system.

Check on the general attributes
The general attributes, which are used to calculate the system statistics, both on-line (during the simulator time)and off line (at the end of the simulation) are also checked. In particular:

- The maximum waiting time at each station
- The time of entrance in the queue and the time of exit from the queue, for each user
- The number of relocations
- Etc.

Check on the final output data
A check on the final output data has been performed, if the values obtained were expected. Some indicators of probable error may be:

- a vehicle keeps occupied also at the end of the simulation: the period without demand is so long that all trips are supposed to end.
- a user keeps inside a vehicle also at the end of the simulation: the same as before.
- a vehicle keeps relocating or redirected at the end of the simulation
- at a given time instant, in a station there are in contemporary vehicles available and users in queue
- anomalous value of users' waiting times at a station in a given instant.
- negative number of users or vehicles in queue
- anomalous distribution of users waiting times: e.g. all users have waited for 0 minutes, or some users have a negative waiting time (if some counters of a previous simulator iteration haven't been reset)

- anomalous occupation period of vehicles, e.g. a vehicle keeps occupied for 4-5 hours; the same is valid for relocations and redirections

4.6.2. Debug performed by checking some specific simulator parameters.

In this case, some system parameters are modified.

Battery characteristics

The battery capacity has been reduced. In particular, still a Li-ion battery is considered, but with only one module and a few cells, therefore the battery capacity is reduced to 1% of the originary one. This analysis has been performed because, thanks to the battery charging management, vehicles charge level is always much above the minimum. In this case, it has been checked if the vehicles status changes correctly to 2 (i.e. unavailable because in charge) and as the minimum level of charge is overcome, that the vehicle becomes available to user and relocation trips in the proper way.

4.6.2.2. Stations capacity

As stated above, the FPT (i.e. full port time) situation is never reached in the normal ongoing of the system and the stations are far beyond capacity. But in order to check the code, it is necessary to create also this extreme situation. Therefore, the station capacity is heavily decreased, for example it is set equal to 7 instead of 15, the high buffer thresholds are set equal to 5 and the high critical thresholds set equal to 6. In this case, it is evaluated:

- if also the relocations in case of FPT work properly, i.e. if the right supporting station is chosen.
- if the station is completely full, it is checked whether the vehicles are really redirected to the nearest station able to accept them.
- when the redirection of the vehicles is completed, if the vehicle in question becomes available, and if the number of available vehicles of the receiving station is properly updated with this new vehicle

4.6.2.3. Pedestrian density

The pedestrian density is heavily changed, it is firstly set equal to nearly 0 pedestrians/m^2 and then set equal to 1 ped/m^2.

4.6.2.4. Duration of periods

The duration of the simulation periods, and also the duration of the simulation day, have been modified, and several durations have been assumed (also a day made of only four hours, in which the first one was the peak one and the second was the off-peak).

4.6.2.5. Transport demand

The transport demand has been modified, specially the coefficient of the trip chain: in particular, this coefficient is set equal to 1 firstly, and secondly equal to a great quantity (like 100). Also user choice sets have been modified, in order to check the correctness of the supervisor choice also for completely different scenarios.
Modifications on the number of units and of stations have not been performed, as these parameters are fixed for the scenario under study.

4.6.3. Modifications to the simulator after debugging.

After debugging, all the bugs related to the simulator have been fixed. This procedure does not need any specific explanation. Instead, an other problem about some specific aspects of the management strategies have come out.

The problem is related to relocations among stations. Indeed, the request for a relocation is dominated by the counter of the vehicles available at the station in the critical situation and at the supporting station. Well, for the case of FPT (full port time) no problems arise. Instead, for the case of ZVT (zero vehicle time), when a station is in critical situation, it requires a vehicle and a relocation begins. However, the vehicle takes some time to reach the station and therefore the stations remains in a critical situation and it continues requiring vehicles. What happens is that some of the providing stations can be in defect of vehicles in a future time, while the receiving station is in excess of vehicles and therefore this multiplies the number of required relocations. Therefore, a new counter has been created in order to avoid this fact, which is the number of vehicles assigned to each station.

4.7. The sub-models contained in the simulator.

4.7.1. The model between vehicle speed – pedestrian density

One of the simulation inputs is the relationship between the speed of the PICAV vehicle and the pedestrian density along the roads. The chosen model of interaction between vehicles and pedestrians is a result of much research performed in the field.

Firstly, a microscopic model has been developed, which is meant to modelize the adaptive behaviour of pedestrians when they meet an obstacle, in corridors, squares, roads, etc. The model is discrete both in space and time and aims to estimate the pedestrian's next position, given his current position and: the positions of other pedestrians and obstacles in the area around the pedestrian in study, and the relative velocity of the obstacles with respect to the pedestrian under study. The pedestrian's behavior has been assumed of "stimulus – response". Much bibliography has been collected regarding this topic. The basis of our work is in Cepolina et al. (2008). In this work the concept of awareness area is taken into account. The awareness area (AwA) is the region in which an obstacle can influence the subject's behaviour, in terms of slowdown, acceleration and modification of the trajectory. The model has been calibrated through experimental data collected at the PAMELA laboratory, in the University College of London, and validated through a series of videos taken in the city centre of Genoa. The results are not much satisfactory, because the R-square index value is 0.23, whilst the T – Student statistics underline that only some cells are significant.

Moreover, the level of information provided by this model is also too much detailed. A high level of detail in the model results in high computational load; and the benefit of the high detail in this model is vanished in the uncertainties related to transport demand. Furthermore, the representation of the network in the simulator involves only the distances and the travel time between each couple of stations and of units. The distances are known; the input data that we need from the model to provide to the simulator is therefore only the travel time between each couple of stations and units, given the length, the slope and the pedestrian density on the path. A higher level of detail of the model can also result in the necessity to modelize the network in a more complex manner.

For all these reasons, a simple model of interaction PICAV vehicle - pedestrians has been developed. This model analyzes the interaction between the PICAV and pedestrians from a macroscopic point of view, and provides the velocity of the vehicle given the value of pedestrian density in a given section of a road. And it does not take into account the microscopic interrelation between the vehicle and pedestrians as it is not necessary as input data for the simulator.

Before developing this model, some literature has been studied about macroscopic models of pedestrians. In the Highway Capacity Manual (HCM) are presented some constitutive relations between speed and density, for what regards both vehicles and pedestrians. In particular, there is a wide literature for what regards vehicle traffic. For what regards pedestrian flows, the topic is still under study, but in the HCM are reported some relations, exposed in the figure 4.6. Several curves are reported in the figure 4.6 according to different desired speeds of pedestrians. It should be noted that for values of density low enough (i.e. the space available is greater than 3 m^2/pedestrian) the pedestrian speed is no more sensitive to density and instead remains constant and equal to the maximum.

Speed – density diagrams for the pedestrian case are much similar to the vehicle ones: the speed diagram is horizontal for reduced vehicular density, and it drastically diminishes for values of density close to capacity.

The density – velocity model for PICAV vehicle and pedestrians has been therefore developed. The model has been developed in an experimental way: an electric scooter for mobility impaired people, which has been assumed having the same performances as the PICAV vehicle, has been driven in pedestrian only environments and a database of couple of values vehicle speed – pedestrian density has been assessed by video records.

It has been assumed that PICAV speed is a linear function of pedestrian density. The function has been determined through linear regression. Several shapes of the functional relationships have been explored, but the linear one has been selected. The model development, calibration and validation is described in detail in the section 6.1.3.

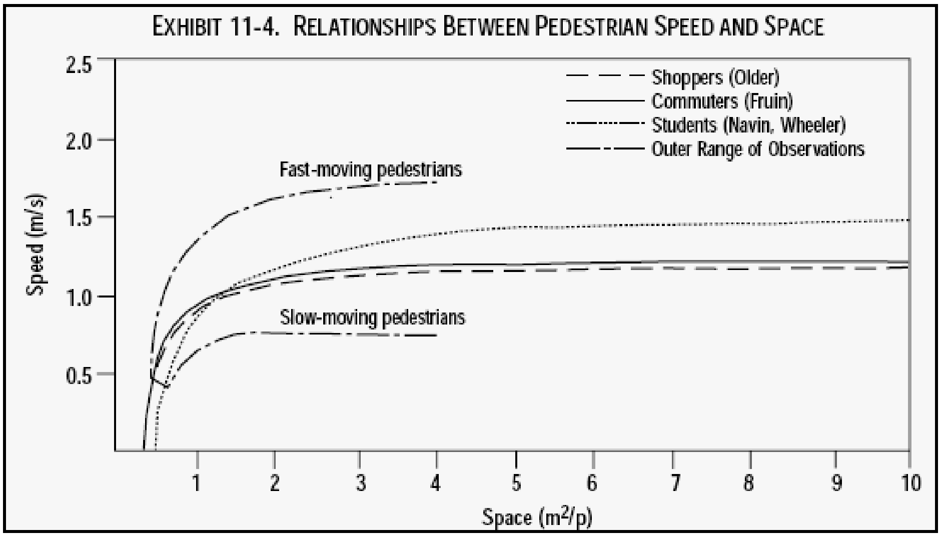


Fig. 4.6. The fundamental diagram for pedestrians as reported in the HCM (HCM 2004). In the y axis the pedestrian speed is reported; in the x axis the free space, expressed in m^2 per pedestrian, is reported.

4.7.2. Model of the vehicle battery

The battery of the PICAV vehicle has been designed by Mazel Ingenerios, a company from Abrera, Barcelona, Spain. The battery is Li-ion and it is composed of 15 blocks in series which together provide 48 V DC. Each block is composed of 27 cells in parallel, which on the whole provide 204 Ah.

The formula to calculate the resistances to motion is the following:

$$R = \left(Pa + Pgc_r + \rho_a C_x S v^2 + Pgi \right) \qquad 4.1.$$

Where:

- a = acceleration of the vehicle;
- P = weight of the vehicle and of the occupant in kg, assumed equal to 500 kg;
- g = gravitation constant, equal to 9,81 m/s^2;
- c_r = rolling coefficient: i.e. specific resistance to rolling, assumed equal to 0,010 N/kN;
- i = average slope expressed in ‰;
- ρ_a = air density, equal to 1.239 kg/m^3;
- C_x = aerodynamic coefficient, equal to 0.5;
- S = PICAV vehicle section, equal to 1.36 m^2;

- The total resistance to motion *R* is expressed in N.

The state of charge (SOC) of the battery is equal to:

$$SOC = 1 - \frac{Q_e}{C} \cong 1 - \frac{E_e / U_m}{C} \qquad 4.2.$$

- Q_e (Ah) = quantity of electricity supplied;
- C (Ah) = battery capacity, equal to 204 Ah;
- U_m (V) = average electric potential of the battery, equal to 48 V;
- E_e (kWh) = energy supplied by the battery.
- The hypothesis of the electric current to be constant in time is made.

The energy supplied by the battery E_e is equal to:

$$E_e = \int_T P_{DC}(t)dt \qquad 4.3.$$

Being the power *P* constant in time, the energy supplied by the battery is equal to: $E_e = P_{DC}\, T$. Where *T* is the considered time period
Moreover, P_{DC}, i.e. the power in entrance to the inverter:

$$P_{DC}\ (\text{Wh}) = \frac{1}{\eta_c} \cdot \frac{1}{\eta_m} P_r = \frac{1}{\eta_c} \cdot \frac{1}{\eta_m} \left[F_r v \right] \qquad 4.4.$$

Where:
- F_r is the force provided at the wheels; it is considered equal to the resistances to motion *R* (i.e. the force needed to move the vehicle);
- η_c ed η_m are respectively the efficiency of the inverter and of the engine, equal to 0.92 e 0.93 respectively (Mazel, 2011);
- v = vehicle's velocity in m/s. It is an input of the model.

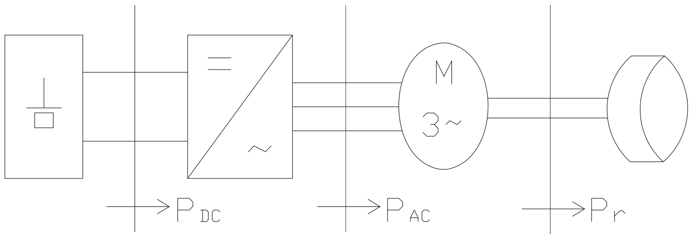


Fig. 4.7. General scheme of the components: battery, inverter, engine, wheels (from left to right). P_{DC} is the power in output of the battery; P_{AC} is the power in output of the inverter, P_r is the power to the wheels.

The battery is a Li-ion, therefore it has no memory effect, and it is possible to recharge it also if its charge level is high. Two charging procedures have been assumed:

1. The first charging procedure is called *opportunity charging*. According to this schema, as the vehicle is idle at some station, it is put under charge. Moreover, it can in any moment be requested by an user if its charging level is greater than the minimum. In each station, however, the vehicle given to the user is always the one with the highest level of charge. The minimum level of charge has been taken, for security reasons, equal to at least three times the quantity of charge requested to perform the most battery consuming trip. an idea of this level has been taken from the questionnaires results, but the precise value has been obtained through simulation. For Genoa case study, the quantity of charge consumed to perform the worst trip is equal to 3%, therefore as minimum level of charge has been assumed the 10%. If the vehicle has a charge level less or equal to 10%, it is not occupied by an user, and it is parked along the road, it is put in charge in the nearest available station.
2. The second charging procedure considers the possibility to exploit the range of charging values in which the battery shows the highest performance. This happens for the level of charge between 40 and 80%. In any case, if the vehicle has a charging level of the battery less than 40% and it is not occupied by an user, it is put in charge in the station. As it is under charge, however, the vehicle becomes unavailable until the battery has recharged up to 80%, independently to the possibility that the vehicle is required or not. The choice of not to overcome the 80% of charge level in case of a recharging process, is due to the fact that above this charge level, the time requested to recharge increases relevantly.

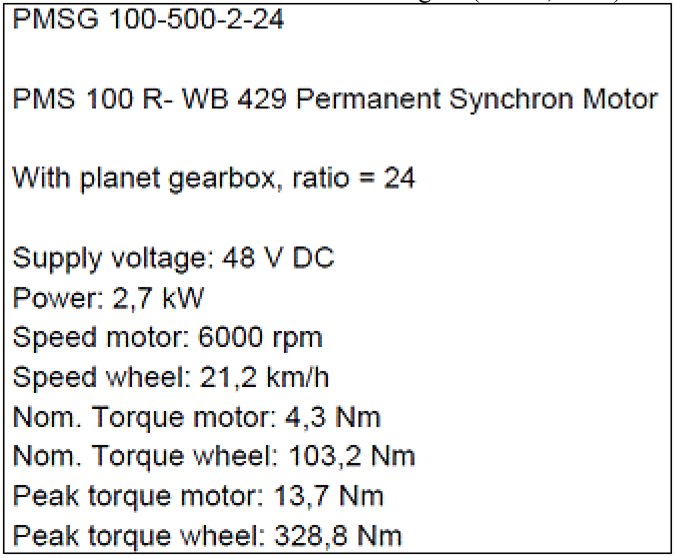
Both charging strategies are performing, but both report some drawbacks. The opportunity charging performs very well for what regards the transportistic point of view. Indeed in Genoa case study, the transport demand registers a first off-peak period and a second peak period. Through the opportunity charging system, in the peak period vehicles still have a very high level of charge and therefore during the four peak hours, the vehicles are always available to users. But however the opportunity charging multiplies the charging cycles therefore batteries diminish their capacity more quickly. The second strategy instead reduces the number of charging cycles but, because the vehicle is never put in charge if its charging level is greater than 40%, it may happen that some vehicles are put in charge exactly in the moment in which they are more necessary. Moreover, as the battery is recharged until 80%, the vehicle keeps unavailable for a long time.

The characteristics of the battery are provided in table 4.2. The characteristics of the engine are provided in table 4.3.

Table 4.2. Characteristics of the battery (Mazel, 2011).

CHARACTERISTICS OF CELLS	
V_{max} cell	3.8 V
$V_{nominal}$ cell	3.20 V
V_{min} cell	2.50 V
$Q_{nominal}$ cell	7.50 Ah
$P_{nominal}$ cell	24.00 Wh
Height	6.7 cm
Diameter	3.2 cm
Weight	135 g
Volume	0.0539 l
Energy density	445.40 Wh/l
Specific Energy	177.78 Wh/kg

Table 4.3. Characteristics of the engine (Mazel, 2011).



PMSG 100-500-2-24

PMS 100 R- WB 429 Permanent Synchron Motor

With planet gearbox, ratio = 24

Supply voltage: 48 V DC
Power: 2,7 kW
Speed motor: 6000 rpm
Speed wheel: 21,2 km/h
Nom. Torque motor: 4,3 Nm
Nom. Torque wheel: 103,2 Nm
Peak torque motor: 13,7 Nm
Peak torque wheel: 328,8 Nm

Chapter 5. The optimization procedure

Introduction

Some of the simulator's input data cannot be directly quantifiable by observations on the field. These data are: the thresholds, and the fleet dimension and its distribution among stations at the beginning of the time period.

Moreover, each of these variables can assume values on a quite wide range; and the problem is combinatorial.

As a result, it is impossible to determine these input data through "what if" simulations. Therefore, the need for optimization procedures arises.

The chapter is organized as follows. Firstly, a general introduction on the optimization problem is provided. Afterwards, the main characteristics of heuristic algorithms are described. Furthermore, the chosen cost function and the chosen optimization procedure are described. Finally, a description on the optimization code is provided.

5.1. Optimization

As reported in Goldberg (1989), the first important aspect is to be clear about our goals when we are going to optimize a function or a process. The optimization seeks to improve performance toward some optimal point or points. Note that this definition has two parts: (1) we seek improvement to approach some (2) optimal point. There is a clear distinction between the process of improvement and the destination of the optimum itself. In judging optimization procedures we commonly focus solely upon convergence (i.e. whether the method reaches the optimum) and forget entirely about interim performance. This emphasis stems from the origins of optimization in the calculus. It is not, however, a natural emphasis.

Consider a human decision maker, for example a businessman. How do we judge his decisions? Usually we say he has done well when he makes adequate selections within the time and resources. We conclude that convergence to the best is not an issue in business or in most walks of life: we are only concerned in doing better relative to others. Therefore, the most important goal of optimization is improvement. Can we get some good, satisficing level of performance quickly? Attainment of the optimum is much less important for complex systems. It would be nice to be perfect: meanwhile, we can only strive to improve.

It is also important to question whether conventional search methods meet our robustness requirements. Robustness is defined as the balance between efficiency and efficacy necessary for survival in many different environments (Goldberg, 1989). The implications of robustness for artificial systems are manifold. If artificial systems can be made more robust, costly redesigns can be reduced or eliminated.

The current literature identifies three main types of search methods: calculus-based, enumerative and random.

Calculus based methods have been studied heavily. They are subdivided into two main classes: indirect and direct. Indirect methods seek local extrema by solving the usually nonlinear set of equations resulting from setting the gradient of the objective function equal to zero. This is the multidimensional generalization of the elementary calculus notion of extremal points. Given a smooth, unconstrained function, finding a possible peak starts by restricting search to those points with slopes of zero in all directions. On the other hand, direct search methods seek local optima by hopping on the function and moving in a direction related to the local gradient. This is simply the notion of "hill-climbing": to find the local best, climb the function in the steepest permissible direction. While both of these calculus-based methods have been improved, extended, hashed, and rehashed, some simple reasoning shows their lack of robustness.

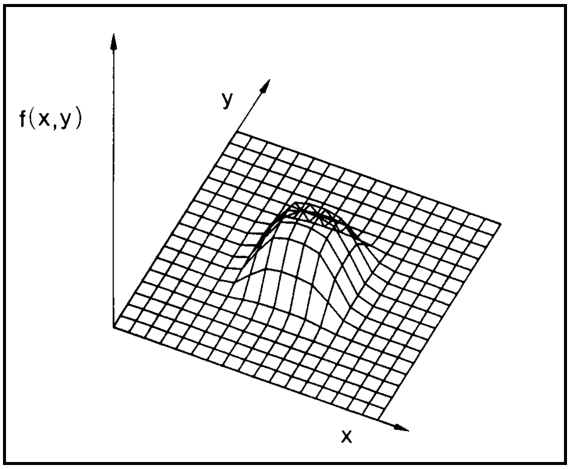


Fig. 5.1. The single-peak function is easy for calculus-based methods (Goldberg, 1989).

First, both methods are local in scope; the optima they seek are the best in a neighbourhood of the current point. For example, suppose that figure 5.1 shows a portion of the complete domain of interest; a more complete picture is shown in figure 5.2. Clearly, starting the search of zero-finding procedures in the neighbourhood of the lower peak will cause us to miss the main event (the higher peak). Furthermore, once the lower peak is reached, further improvement must be sought through random restart of our trickery. Second, calculus-based methods depend upon the existence of derivatives (well-defined slope values). Even if we allow numerical approximation of derivatives, this is a severe shortcoming. Many practical parameter spaces have little respect for the notion of a derivative and the smoothness this implies. Theorists interested in optimization have been too willing to accept the legacy of the great eighteenth and nineteenth-century mathematicians who painted a clean world of quadratic objective functions, ideal constraints, and ever present derivatives. The real world of search is fraught with discontinuities and vast multimodal, noisy search spaces as depicted in a less calculus-friendly function in figure 5.3. It comes as no surprise that methods depending upon the restrictive requirements of continuity and derivative existence are unsuitable for all but a very limited problem domain. For this reason and because of their inherently local scope of search, we must reject calculus-based methods. They are insufficiently robust in unintended domains.

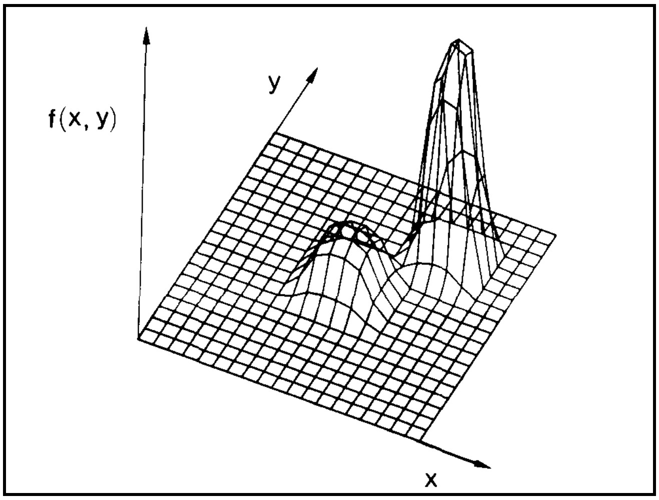


Fig. 5.2. A multi peak function causes a problem: which hill to climb (Goldberg, 1989).

Enumerative schemes have been considered in many shapes and sizes. The idea is fairly straightforward: within a finite search space, or a discretized infinite search space, the search algorithm starts looking at objective function values at every point of the space, one at a time. Although the simplicity of this type of algorithm is attractive, and enumeration is a very human kind of search (when the number of possibilities is small), such schemes must ultimately be discounted in the robustness race for one simple reason: lack of efficiency. Many practical spaces are simply too large to search one at a time and still have a chance of using the information to some practical end. Even the highly touted enumerative scheme dynamic programming breaks down on problems of moderate size and complexity, suffering from a malady melodramatically labelled the “curse of dimensionality” by its creator (Bellman, 1961). We must conclude that less clever enumerative schemes are similarly, and more abundantly, cursed for real problems.

Random search algorithms have achieved increasing popularity as researchers have recognized the shortcomings of calculus-based and enumerative schemes. Yet, random walks and random schemes that search and save the best must also be discounted because of the efficiency requirement. Random searches, in the long run, can be expected to do no better than enumerative schemes. In our haste to discount strictly random search methods, we must be careful to separate them from randomized techniques. Using random choice as a tool in a directed search process seems strange at first, but nature contains many examples. The important thing to recognize at this juncture is that randomized search does not necessarily imply directionless search. While our discussion has not been an exhaustive examination of the myriad of methods of traditional optimization, we are left with a somewhat unsettling conclusion: conventional search methods are not robust. As more complex problems are attacked, other methods will be necessary. To put the point in better perspective, inspect the problem spectrum of fig. 5.4. In the figure a mythical effectiveness index is plotted across a problem continuum for a specialized scheme, an enumerative scheme and an idealized robust scheme. The gradient technique performs well in its narrow problem class, as we expect, but it becomes highly inefficient elsewhere. On the other hand, the enumerative scheme performs with egalitarian inefficiency across the spectrum of problems, as shown by the lower performance curve. Far more desirable would be a performance curve like the one labelled Robust Scheme. It would be worthwhile sacrificing peak performance on a particular problem to achieve a relatively high level of performance across the spectrum of problems.

Among random search algorithms, the Simulated Annealing (SA), which has been implemented in the thesis, and the Genetic Algorithms, should be mentioned.

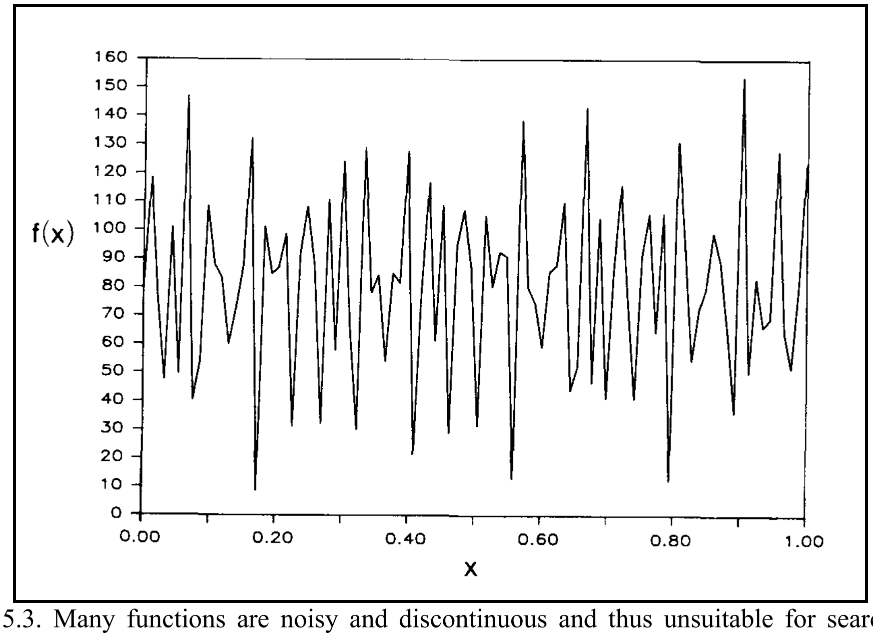


Fig. 5.3. Many functions are noisy and discontinuous and thus unsuitable for search by traditional methods (Goldberg, 1989).

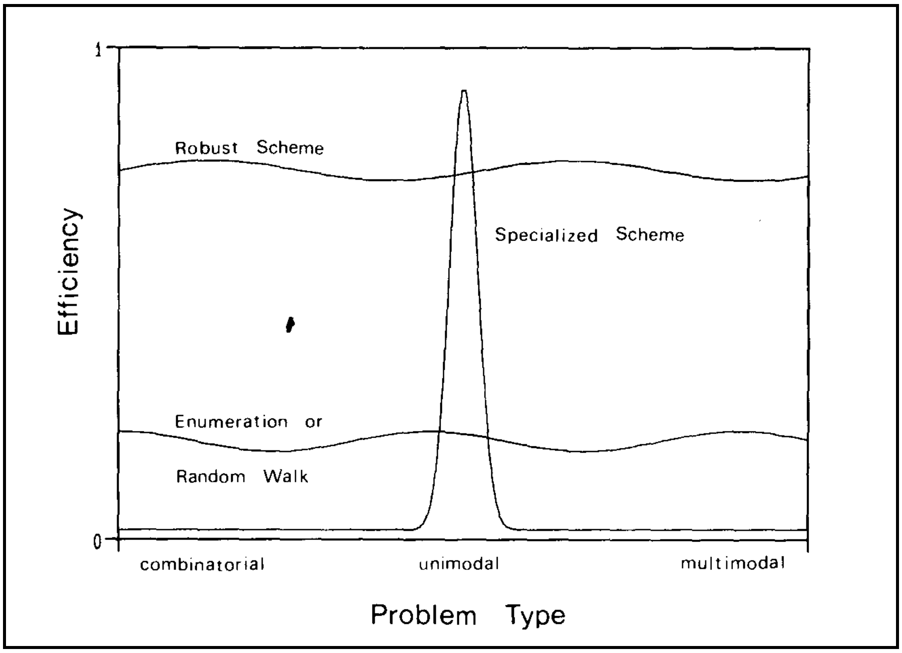


Fig. 5.4. Many traditional schemes work well in a narrow problem domain. Enumerative schemes and random walks work equally inefficiently across a broad spectrum. A robust method works well across a broad spectrum of problems (Goldberg, 1989).

5.2. Introduction on heuristic algorithms

The determination of the solution of a difficult problem may be too long or impossible if we use the so-called exact methods, which also sometimes cannot be applied. In most cases, the practical aspect is preponderant, and therefore the purpose is not to find the exact solution but a good solution. Several techniques have been developed to determine a good solution, called heuristic algorithms.

As stated in Monaci (2012), an heuristic algorithm is a technique which is able to provide a solution to a problem. The ideal heuristic algorithm should be always able to determine the optimum solution of a problem. This can happen for a specific typology of problems which have a specific structure, and are called matroides: e.g. the algorithm to determine the tree with minimum cost. For more difficult problems it is envisaged that the algorithm provides a good solution of the problem, i.e. whose value is close enough to the optimum solution. For some difficult problems, such as the travelling salesman problem on a non complete graph, it is also possible that the algorithm is unable to determine a feasible solution to the problem.

Given an instance *I* of a minimization problem, a heuristic algorithm *A* provides a solution with value $z^A(I)$:

$$z^A(I) \geq z^*(I) \qquad 5.1.$$

where $z^*(I)$ is the optimum value of the instance. We can say that the algorithm provides an upper bound of the value of the optimum solution. Obviously in the case of a maximization problem is valid the opposite inequality of 5.1. and the algorithm in this case provides a lower bound of the solution. An euristic algorithm may be able to determine an optimum solution of the problem $z^*(I)$ but this optimality cannot be verified.
Heuristic algorithms can be classified as follows: constructive algorithms, local search algorithms, meta-heuristic algorithms:
The time of calculation, and the quality of solutions produced, increases as we pass from constructive algorithms to meta-heuristics. According to the application and to the quality of solutions desired, it will be more convenient to choose an algorithm instead of another. The important thing is that the time to calculate required is much less than the time required by an exact algorithm to resolve the same problem.

5.2.1. Constructive algorithms

They determinate an admissible solution starting only from the input data of the problem in exam. For some problems it may be difficult to determine an admissible solution: in this case we look for a solution which is as much admissible as possible.

Among constructive algorithms, *greedy* algorithms start from an empty solution and construct iteratively the solution following an expansion criterium, which conisits on making, at each iteration, the most convenient choice compatible with the problem constraints. Any choice, as it is made, is never put under discussion. The conceptual choice of any greedy algorithm is as follows:

- Initialization of the solution *S*
- For any choice to make, take the most convenient decision, according to the problem constraints

The great diffusion of greedy algorithms is due to the fact that they simulate the manual behaviour in the determination of the problem solution, that the implementation of the algorithm is much easy, and that the computational time is much low.

In some cases, greedy algorithms are based on pre-sorting of the elements to be considered for the solution, called dispatching rule: the elements that define the solution are considered in this order and possibly inserted in solution. Generally, the sorting criteria used include associate to each choice a "score" that indicates the goodness of the move, trying to reward at each iteration a move that seems to be the most promising. The information about the score or can be calculated at the beginning of the execution, basing on the input data. Often the same heuristic algorithm provides best results if the sort order of the elements is dynamically updated in order to take account of the choices made earlier; obviously the continuous updating of the score of the elements leads to an increase in the computation time required by the algorithm itself.

Generally, greedy algorithms are of primal, i.e. they make choices that meet always all constraints. There are also dual versions of such algorithms: these start from non admissible solutions and make choices aimed at achieving the admissibility, trying not deteriorate too much the value of the solution.

The *algorithms based on optimization techniques* are iterative heuristic algorithms that, at each iteration, perform one or more certain choices solving an optimization problem which is simpler than the original problem. Typically they provide good results based on the heuristic relaxations: they are greedy algorithms in which the scores are defined using some (optimal solution, reduced costs, lagrangian costs) obtained by solving a relaxed problem. Generally, the calculation time required for execution of these algorithms is greater than that required by greedy algorithms, but the solutions tend to be produced in much higher quality.

The fact of having available a relaxation of the problem also allows the reductions in the size of the problem, and these reductions can have a dramatic effect in reduction in computation time for large instances. These reduction techniques are aimed at fixing some of the variables to their optimal value. The techniques used can be exact (definition of a core problem based on, for example, reduced costs or lagrangian costs) or heuristics (fixing of variables according to their value in the relaxed solution).

The *algorithms based on implicit enumeration* of the solutions are algorithms based on partial implementation of an enumerated algorithm. The most common variant is that obtained by doing a complete branch-and-bound algorithm after a certain time limit or after a predetermined number of backtracking. Another possible solution is to modify the branch-and-bound algorithm in order to generate, starting from each node, at most k nodes "children" (where k is a parameter to be calibrated according to the computing time available). The choice of *k* sub-problems to generate is usually carried out by assessing, for each potential node "son", a bound on the value of the best solution in the corresponding sub-tree and taking the *k* most promising children nodes.

5.2.2. Local search algorithms

Given an optimization problem *P* defined by an objective function *f* and from one region admissible *F*, a neighbourhood is an application that at each point *s* of the feasible region associates a subset of *N(s)* of the feasible region *F*:

$$N: s \rightarrow N(s) \tag{5.2.}$$

Sometimes it can be helpful to define the neighbourhood, as the set of solutions reachable, from the current solution through moves of the chosen type.

Transition rule is the methodology utilized to explore the neighbourhood: it defines the criteria for searching a new solution departing from the current one.

The basic idea of the local search algorithm is to define an initial solution (current solution) and try to improve it by exploring a neighbourhood (appropriately defined) of this solution.

If the optimization in the neighbourhood of the current solution produces an improving solution the procedure is repeated starting, as the current solution, from the solution just determined.

The algorithm ends when: the exploration of the neighbourhood is completed or after a predetermined number of iterations.

With regard to the definition of the initial solution, it is likely to think that it is better to start from a "good" solution rather than a "poor" solution: actually no theoretical result exist that confirms this. Generally what is done is to run multiple times the local search algorithm starting from different solutions (randomly generated) in order to explore different areas of the feasible region.

As regards the dimension of the neighbourhood, this depends on the type of moves that are wanted. It is obvious that if, for any feasible solution, the neighborhood coincides with the entire feasible region, the procedure returns the optimal solution of the problem. In general, the larger the size of the neighbourhood and the higher the probability of escape from local optima; for this reason one would like, if possible, to have neighbourhoods that contain a lot of solutions. Obviously there is also the need to be able to explore the neighbourhoods efficiently, i.e. the algorithm must find the solution with a low number of iterations; often, however, these two requirements are conflicting, so it is necessary to find a compromise between the dimension of the neighbourhood and the "goodness" of local optima.

Another feature that is desired for the search space is the fact of the neighbourhood being connected; this means that regardless of the starting solution, it is always possible to reach any feasible solution (including the optimal one) through an appropriate sequence of moves, so if the neighbourhood is connected, a "lucky" algorithm would always find the optimal solution of the problem.

5.2.3. Meta-heuristic algorithms

The solution to which converges a local search algorithm is a local optimum, given a chosen neighbourhood. In general, the solution found is not a global optimum, unless the problem that want to solve is not convex programming (as is the case for linear programming). The local optima determined by local search procedures can be of poor quality. For improving the solution found by local search techniques it is possible to run the algorithm more times, starting with different initial solutions and / or defining the around (or the exploration strategy of the surroundings) in a different way in each iteration. Another way to try to improve the resulting solutions is the use of meta-heuristic algorithms: these are essentially extensions of local search algorithms, which are introduced in appropriate techniques aimed to avoid to terminate in a local optimum. The basic idea of a meta-heuristic algorithms is the possibility of accepting also worse moves, in order to "escape" from the local optima. Once that a worse move has been made, we must avoid to return to the starting move (although this is improving), as this would create an infinite loop. It is therefore essential to establish techniques that will avoid incurring cycles.

The meta-heuristic techniques are of general algorithmic paradigms that must be particularized for individual problems. Often the corresponding algorithms are not very competitive compared to algorithms specifically developed for the problem, in the sense that the solutions produced are comparable in terms of the value of the solution but the computation time required can be much higher. On the other hand, the timing of development and of implementation of a basis meta-heuristic algorithm are much lower compared to a technique ad hoc. The critical phase in the development of these algorithms often ends up being the calibration of a series of parameters that appear in the algorithm itself.

The most important meta-heuristic algorithms, that will be described in detail in the following, are:

1. Simulated Annealing
2. Genetic algorithms
3. Tabu Search

5.2.4. Simulated Annealing

The basic idea of these algorithms, born in the early 80's, is to simulate the behavior of a thermodynamic process of annealing (annealing) of solid materials (glass, metal, ...). If a solid material is heated to above its melting point and then is cooled (cooling) in so as to return it to its solid state, its structural properties depend strongly on the process cooling (cooling schedule). Essentially, a simulated annealing algorithm simulates a change of energy of a system (considered as a set of particles) subjected to cooling, until it converges to a solid state; this allows to look for eligible solutions of optimization problems, trying to converge towards the optimal solutions.

In a thermodynamic process, the probability that a system involves a change of state which corresponds to an increase of energy δE is equal to:

$$\mathrm{p}(\delta E) = \exp\,(-\,\delta E / \,k\, T\,) \qquad 5.3.$$

where k is the constant of Boltzmann and T is the temperature of the system.

The simulated annealing algorithms use local search techniques to define and explore the neighborhood of a current solution. If the solution contains an improvable neighbourhood, then this becomes the new current solution and the process is iterated. Otherwise, the worsening of the solution value is evaluated, that would move from the current solution to the best solution of the neighbourhood, and this displacement is carried out in accordance with a certain probability. The algorithm ends when it reaches a predetermined number of iterations or a predetermined time of calculation, or if the optimality of the current solution is proved.

The deterioration of the value of the solution is seen as an increase in the energy of the associated thermodynamic system, for which the probability of accepting a move worsening is given by the formula of Metropolis: $p = \exp(-\delta/T)$; where δ is the extent of deterioration (difference between the value of the new solution and the value of the current solution) and T is the *process temperature*. If the move is accepted, then the algorithm continues starting with the new current solution, otherwise, the second-best solution of the neighbourhood is evaluated, and so on until any acceptable solution is found.

The formula of Metropolis states that: the probability of accepting the worse move decreases with the increase of the worsening of the move; and increases with the increase of the process temperature. The temperature is a control parameter of the algorithm, which generally is initialized to a value $t_0 > 0$ and updated afterwards, with a certain frequency, according to a function of reduction, or when the number of moves accepted was above a certain threshold for a certain number of iterations. This is due to the fact that the probability of accepting worsening moves must be relatively high at the beginning of the algorithm (when the sequence of solutions considered strongly depends on the initial solution), then becomes lower as the solution improves (when you are exploring promising areas of the feasible region).

At the basis of the approach are some theoretical results, since the behaviour of a Simulated Annealing algorithm can be modelled using Markov chains. Fixed the temperature, the probability of moving from the solution i to the solution j is a quantity $p_{ij} > 0$ that depends only on i and j. The corresponding transition matrix corresponds to what is known as a homogeneous Markov chain (at least if the number of iterations performed at a constant temperature is sufficiently large): if you can find a sequence of exchanges that transform a solution in another solution with non-zero probability, then the process corresponds to a stationary distribution, regardless of the starting solution. Since the temperature tends to zero, this stationary distribution tends to be a uniform distribution in the set of optimum solutions, for which the probability that the algorithm converges to an optimal solution tends to 1. However, given that the solution space has generally exponential size, the number of iterations (and thus also the calculation time) required to ensure the convergence of the algorithm to an optimal solution is also exponential.

5.2.5. Genetic Algorithms

Genetic algorithms are search algorithms based on the mechanics of natural selection and natural genetics. They combine survival of the fittest among string structures with a structured yet randomized information exchange to form a search algorithm with some of the innovative flair of human search. In every generation, a new set of artificial creatures (strings) is created using bits and pieces of the fittest of the old; an occasional new part is tried for good measure. While randomized, genetic algorithms are no simple random walk. They efficiently exploit historical information to speculate on new search points with expected improved performance.

Genetic algorithms have been developed by John Holland, his colleagues, and his students at the University of Michigan. The goals of their research have been twofold: (1) to abstract and rigorously explain the adaptive processes of natural systems, and (2) to design artificial systems software that retains the important mechanisms of natural systems. This approach has led to important discoveries in both natural and artificial systems science.

Genetic algorithms are based on the observation of the evolution of a population of individuals: individuals combine themselves to generate new individuals in successive generations and the evolutionary process tends to select (to make survive) only the most suitable. By matching each individual with a solution and by associating to each solution a fitness, these algorithms seek to simulate the process of evolution, trying to keep alive the only solutions of high quality.

The starting point for these algorithms is the genetic representation of the solution; generally each solution is defined by the value of *n* integer variables, and each variable corresponds to a chromosome. The representation of continuous variables turns out to be more difficult, so it generally uses an encoding that represents the approximate value of the variable.

The algorithm starts with an initial population of cromosomes (distinct solutions generated in some way) and tries to optimize the fitness of the population by recombination (cross-over) and mutation of genes in the course of several generations. The process is repeated until a given number of populations is generated or the optimality of the best solution found is demonstrated.

5.2.6. Tabu Search

Local search algorithms, i.e. the simulated annealing and genetic algorithms, in their evolution, do not take track of all past history, but only some basic information (the current solution, the best solution found, the number of iterations which do not improve the solution, etc.). The systematic use of memory is instead one of the essential characteristics of the algorithms like Tabu Search, i.e. local search algorithms in which at each iteration we also agree to move towards worsening solutions, provided eligible and *legal* ones (in meta-heuristics was only eligible ones). What are legal and illegal solutions, will be described herewith.

These algorithms generally provide solutions better than other meta-heuristic methods, although there is no formal proof of convergence. The fact of accepting moves that worsen the current solution is to avoid ending up in a local optimum. In order to avoid loops, a structure is defined (called Tabu List) in which the last *t* moves are stored. The solutions corresponding to these moves are forbidden (*not* *legal*, i.e. tabu) for subsequent iterations, i.e. they cannot be accepted any more during the execution of the algorithm. Then, with x^k indicating the solution considered to the *k*-th iteration, at the generic iteration *k* the tabu list is a structure of the type $T = \{x^{k-1}, x^{k-2}, \ldots, x^{k-t}\}$ and prevents the occurrence of cycles of length less or equal to t; the length *t* of the tabu list is called *tabu tenure*.

So, in a tabu search algorithm type, the definition of the neighborhood *N (x)* associated to the current solution *x* depends not only on *x* but also on the previous history (set of solutions visited so far), this did not happen neither in the simulated annealing algorithm nor in the genetic algorithms.

As usual, the stop condition is linked to the fact that a predetermined number of iterations has been carried out or a predetermined calculation time has elapsed, or that it has been proven optimality for the solution x^*. There are also other criteria e.g. related to the fact that the best solution has not been updated by too many iterations.

The storage of complete solutions in the tabu list may not be very efficient from the point of view of calculation time and memory occupation. So it is preferred to store in the tabu list only some attributes of the solutions; alternatively, for each solution $\underline{x}$ obtainable from the solution *x* by a move *m*, we can store only the move *m*. For example in the KP-01 (knapsack-01), if it is chosen to insert the object *j*, it is prohibited to remove the object *j* in the subsequent iterations. If there is an exchange between objects *i* and *j* , exchanges involving again *i* and *j* are prohibited.

The performance of a Tabu Search algorithm depends on several factors:

1. the choice of the neighbourhood, because the algorithm turns out to be efficient only if it is possible to explore efficiently the neighborhood. In order to increase the probability to explore different areas of the feasible region, different neighborhoods are often used, within the same Tabu Search algorithm, and different priority criteria for exploring these neighborhoods. Each neighborhood is associated with a tabu list, the management of which is completely independent from the other tabu lists.
2. The constraints of the problem: if the problem is much constrained the cardinality of the neighborhood can be very small, it is possible to get quickly to a situation where $N(x) \setminus T = 0$, where *N(x)* is the neighbourhood and *T* is the set of tabu solutions. In this case it is convenient to relax some of the constraints and to add to the objective function a penalty proportional to the violation of the constraint by the solution in exam.
3. Management policy of the tabu list. In order to ensure the efficiency of the algorithm it is necessary to be able to control efficiently if a solution is tabu or not. This fact involves also the definition of tabu tenure *t* of the list, in the sense that an increase of *t* increases the system memory (and increases the length of the cycles forbidden) but also increases the time of calculation required to verify the legality of a solution. The easiest way to

manage the length of the tabu list is to define a constant length *t*, although this may result limitative.

4. Intensification and diversification. The algorithm must be able to fully explore areas of the feasible region that seem most promising, and quickly escape from the areas that seem uninteresting. This can be carried out with a dynamic management of the tabu tenure, in which the value of *t* increases as the best solution x^* is updated (intensification phase), while *t* decreases if x^* has not been updated during the previous α iterations (diversification phase). Another way to get the intensification and the diversification is to modify the objective function, penalizing or rewarding solutions based on how much they deviate from the current solution.

5.3. The optimization algorithm

An optimization algorithm has been developed in order to assess the best fleet dimension, the low critical and low buffer thresholds, and the number of vehicles to assign to each station at the beginning of the time period. The problem of optimizing the quantities exposed above is formulated as a constrained minimisation problem. The function *z* to be minimised is a linear combination of the transport system cost $C_d^{\,s}$ and of the user's costs $C_d^{\,u}$, subject to the constraint that the level of service results not lower than E. Both costs refer to the daily reference time period and their units are €/day.

5.3.1. The cost function

The cost function assumes the form:

$$\min z = \min\left(C_d^s + C_d^u\right) \qquad 5.4a$$

subject to:

$$\mathbf{g}(\mathbf{s}) = \begin{cases} t_w^{50} < 4 \\ t_w^{90} < 8 \\ t_w^{95} < 10 \end{cases} \qquad 5.4b$$

Where:

- C_d^s = daily cost supported by the transport system
- C_d^u = daily cost supported by users
- $t_w^{50}, t_w^{90}, t_w^{95}$ = 50th, 90th and 95th percentiles of user waiting time, expressed in minutes.

According to table 4.1, if one of the 3 constraints in 5.4b on the 50th, 90th and 95th percentiles of the user waiting time is not satisfied, the LOS of the system is F.

The daily cost of the transport system, C_d^s, is equal to:

$$C_d^s = C_d^f + C_d^{run} + C_d^{su} + C_d^{man} - C_d^{ts} + C_d^r \qquad 5.5.$$

The transport system cost therefore includes the cost C_d^f of purchasing the fleet, the running cost C_d^{run} of the PICAV units, the cost C_d^{su} of the parking lots setup and the management costs C_d^{man}. The revenue C_d^{ts} derived from the tickets is subtracted from C_d^s.

For C_d^f, i.e. the daily cost of the fleet, we assume that the purchase price for each vehicle is 9000 €. This cost includes the vehicle with a lifetime of 8 years and two lithium-ion battery packs with a lifetime of 4 years each. The cost of adding vehicles to the system increases linearly with each vehicle. The linear increase in vehicle costs is a rather simplistic assumption and does not take into account any economy of scale, but it is a quite frequent assumption (Barth and Todd, 1999). The daily cost of amortization of the fleet has been calculated with a discount rate of 8%, according with the following formula:

$$C_d^f = n_v c_v \left[\frac{r(1+r)^{lt}}{(1+r)^{lt} - 1} \right] \frac{1}{365} \qquad 5.6.$$

Where:

- n_v is the number of PICAV vehicles within the fleet;
- c_v is each PICAV vehicle purchase cost (it includes two battery packs);
- r is the discount rate.
- lt (number of years) is the PICAV vehicle lifetime.

We choose a discount rate of 8% as it is an average rate of return for investments and therefore reasonably represents the opportunity cost of the purchase. There is disagreement on what should be the appropriate discount rate but it is a parameter in the proposed methodology and a sensitivity analysis can be performed.

Again, C_d^{su} and C_d^{man}, which includes the salaries of the system supervisor and of the maintenance, are other flat costs. C_d^{ts} is the revenue derived from the tickets. Multiplying the number of PICAV users by ticket price provides approximate ticket revenues. Since the number of PICAV users in a time period has been assumed constant, C_d^{ts} is again a flat cost.

C_d^{run}, i.e. the daily cost of running the system, includes the maintenance costs and the electricity cost to run the PICAV fleet. Regarding the maintenance costs, and excluding the cost of batteries that we already included in the vehicle purchase cost, electric cars incur in very low costs and we neglect them. As it concerns the electricity needed to charge all the PICAV vehicles during an operative day, it is proportional to the average daily kilometres travelled by the PICAV users and to the length of the relocation trips. Since we assume a constant PICAV transport demand, the average daily kilometres travelled by the PICAV users do not depend on the transport system parameters (i.e. the fleet dimension and the thresholds). Conversely , the length of the relocation trips depends on the transport system parameters. The electricity cost related to relocation trips is taken into account in eq. 5.7 and it is proportional, through the vehicles velocity, to the total amount of time spent in relocation.

C_d^{su} , i.e. the daily cost of system setup, includes infrastructure cost which is related to the number of parking spaces in each station. This cost depends on the fleet dimension. But due to the small dimension of PICAV vehicles (1.3 m long and 1.1 m wide) (Masood et al., 2011) we consider it as a flat cost.

C_d^{man} , i.e. the daily cost of system management are again flat costs.

C_d^{ts} is the daily revenue derived from the tickets. The ticket price is the same for all the PICAV users. If we multiply the number of PICAV users by the ticket price, this provides approximate ticket revenues. Since the number of users in a time period has been assumed constant, C_d^{ts} is again a flat cost.

C_d^r , i.e. the daily cost of relocation, is very relevant in the determination of the objective function. It is calculated from the total amount of time that PICAV vehicles have spent relocating:

$$C_d^r = c_r \sum_{j=1}^{n} t_{rj} \qquad 5.7$$

Where:

- c_r is the cost of each minute of relocation, assumed equal to 0.01 €/min;
- t_{rj} is the total amount of time spent by the vehicle *j* in relocation. It is an output of the micro simulator;
- *j* is the *j*th vehicle and *n* is the number of PICAV vehicles.

The total relocation time takes into account the time spent by all PICAV vehicles relocating among the stations.

The value of the cost of relocation, equal to 0.01 €/minute, has been assessed in order to make the daily relocation cost and the daily user cost comparable with each other.

With regards to the C_d^u in 5.4a, this has been measured in terms of the total customer waiting time and the cost the users have to pay for the service. The total customer waiting time is the overall length of time, in minutes, customers have to wait for a vehicle when it is not immediately available during the reference time period. C_d^u has the following expression:

$$C_d^u = c_w \sum_{i=1}^{m} t_{wi} + C_d^{ts} \tag{5.8}$$

Where:

- c_w is the cost of a unit of waiting time, namely 0.10 €/ minute (Cherchi, 2003);
- m is the total number of users who have been in a queue during the reference time period;
- t_{wi} is the waiting time of the i^{th} user in minutes;
- $\sum_{i=1}^{m} t_{wi}$ is the total customer waiting time in the simulation day, expressed in minutes. It is a function of the number of PICAV units assigned to each parking lot at the beginning of the time period, or of the fleet dimension and the threshold values. It is determined through the simulator and it is an output of the simulator itself;
- C_d^{ts} is the overall cost paid by the PICAV users and it is given multiplying the number of PICAV users by ticket price. Since the PICAV transport demand has been assumed constant, C_d^{ts} is again a flat cost. Moreover this term equals the revenue derived from the tickets that appears in the transport system cost, therefore both terms appear in the cost function with opposite signs, and therefore delete each other.

5.3.1.1. The independent variables for the management strategy involving flexible users

In the current management strategy, thresholds are not parameters of the system therefore they do not have to be optimized.

If we call **s** a vector whose dimension is equal to the number of parking lots and the value of each vector component equals the number of PICAV units in the related parking lot at the beginning of the reference time period, we have that $\sum_{i=1}^{m} t_{wi}$ is a function of **s**: $\sum_{i=1}^{m} t_{wi}(\mathbf{s})$.

Instead of minimizing the z function in order to assess the best number of PICAV units to assign to each parking lot at the beginning of the time period, we minimise another function f which coincides with z but it doesn't include the flat terms since they do not affect the minimization problem. The f function has the following expression:

$$f(\mathbf{s}) = n_v c_v \left[\frac{r(1+r)^{lt}}{(1+r)^{lt} - 1} \right] \frac{1}{365} + c_w \sum_{i=1}^{m} t_{wi}(\mathbf{s}) \tag{5.9.}$$

In the faced problem, the independent variable is **s** since n_v is the sum of all the components of **s**. The research space is given by $\boldsymbol{N}^a$ if a is the number of parking lots in the study area: therefore the research space could be extremely large.
We introduce the constraints on $f(\mathbf{s})$ minimisation problem: the three constraints shown in formula 5.4b. and named as

$$\mathbf{g}(\mathbf{s}) = \begin{cases} t_w^{50} - 4 \leq 0 \\ t_w^{90} - 8 \leq 0 \\ t_w^{95} - 10 \leq 0 \end{cases} \qquad 5.10.$$

We transform the constrained problem:

$$\begin{gathered} \text{Minimize } f(\mathbf{s}) \\ \text{Subject to: } \mathbf{g}(\mathbf{s}) \leq 0 \end{gathered} \qquad 5.11.$$

into a single unconstrained problem using penalty functions. The constraint is placed into the objective function $h(\mathbf{s})$ via a penalty parameter $\hat{\mu}>0$ in a way that penalizes any violation of the constraint:

$$\min h(\mathbf{s}) = f(\mathbf{s}) + \hat{\mu}\left[\sum_{i=1}^{3} \max\{0, g_i(\mathbf{s})\}\right]^2 \qquad 5.12.$$

where i is the ith constraint.
If $g_i(\mathbf{s}) \leq 0$, then the $\left[\max\{0, g_i(\mathbf{s})\}\right]^2 = 0$, and no penalty is incurred.
On the other hand, if $g_i(\mathbf{s}) > 0$, then $\left[\max\{0, g_i(\mathbf{s})\}\right]^2 > 0$ and the penalty term $\hat{\mu}\left[\max\{0, g_i(\mathbf{s})\}\right]^2$ is realized.
The weight of the penalty function $\hat{\mu}$ is unknown and we determine it with an iterative process, according with Bazaraa (Bazaraa et alii, 1993), starting from a small value and increasing it progressively. In fact, as the value of β increases, the optimal solution of $h(\mathbf{s})$ get arbitrarily close to the optimal solution of the original constrained problem:

$$\begin{gathered} \min\ f(\mathbf{s}) \\ \text{subject to } \mathbf{g}(\mathbf{s}) \leq 0 \end{gathered} \qquad 5.13.$$

It is possible also to take different weights for each constraint. In this case 5.12 becomes:

$$\min h(\mathbf{s}) = f(\mathbf{s}) + \sum_{i=1}^{3} \hat{\mu}_i \left[\max\left\{0, g_i(\mathbf{s})\right\} \right]^2 \qquad 5.14.$$

The initial value of the penalty parameter is taken as $\mu_1 = 0.1$, and its updating happens in this way: $\mu_{k+1} = \beta\mu_k$, where the scalar μ is taken as 10.0 and k and $k+1$ are two successive iterations.

For a given μ_k, we have a function $h_k(\mathbf{s})$ to minimise. If the vector $\mathbf{s}_*$ that minimises $h_k(\mathbf{s})$ is the same vector for which $g(\mathbf{s}_*) \cong 0$ we stop the iterative process and $\hat{\mu} = \mu_k$.

For a given value for the penalty parameter μ_k and a given point in the research space (a given $\mathbf{s}_*$), we calculate the value of the $h_k(\mathbf{s}_*)$ function simulating the transport system with the microscopic simulator. Given an area of study and the number of PICAV users, the simulation input is the PICAV fleet and its distribution among the various stations, represented by a given vector $\mathbf{s}_*$, and the simulation outputs are the total customer waiting time $\sum_{i=1}^{m} t_{wi}(\mathbf{s}_*)$ and the percentiles of user waiting times $\mathrm{t}_{\mathrm{w}}^{95}(\mathbf{s}_*)$, $\mathrm{t}_{\mathrm{w}}^{50}(\mathbf{s}_*)$, $\mathrm{t}_{\mathrm{w}}^{90}(\mathbf{s}_*)$.

Since there is not an analytical expression for $h_k(\mathbf{s})$, we cannot exclude the need to deal with a multi-peak function and the risk of reaching a local minimum is high. Moreover, since the research space is extremely large, Simulated Annealing (SA) has been chosen for solving the minimization problem.

5.3.1.2. The independent variables for the two management strategies involving automated vehicles

We call **s** a vector whose dimension is equal to 3 and whose components are:

- the first component is equal to the *fleet dimension* n_V;
- the second component is equal to the *low critical threshold*. Low critical threshold values are taken constant for all the simulation day and also are assumed to be the same in all stations. The value reported in the second component of the vector **s** is the low critical threshold in one of the stations;
- the third component is equal to the *low buffer threshold*. Low buffer threshold values are taken constant for all the simulation day and also are assumed to be the same in all stations. The value reported in the third component of the vector **s** is the low buffer threshold in one of the stations.

Both $\sum_{i=1}^{m} t_{wi}$ and $\sum_{j=1}^{n} t_{rj}$ are a function of **s**: $\sum_{i=1}^{m} t_{wi}(\mathbf{s})$, $\sum_{j=1}^{n} t_{rj}(\mathbf{s})$.

As stated before, high critical thresholds have not been optimised since they correspond to the station capacities. The high buffer thresholds are not optimized, since a relocation strategy based on preventing ZVT occurrences, guarantees that FPT occurrences do not take place .

Instead of minimizing the z function in eq. 3.1a in order to assess the best transport system parameters, we minimize another function f which is simply the function z but without the flat cost terms since they do not affect the minimization problem. The f function has the following expression:

$$f(\mathbf{s}) = n_v c_v \left[\frac{r(1+r)^{lt}}{(1+r)^{lt}-1} \right] \frac{1}{365} + c_w \sum_{i=1}^{m} t_{wi}(\mathbf{s}) + c_r \sum_{j=1}^{n} t_{ri}(\mathbf{s}) \qquad 5.15.$$

In the formulated problem, the independent variable is **s**.

The search space has three dimensions, each one is determined by a component of the vector **s**. the search space is limited as follows:

- the low buffer threshold of each station must be greater or at least equal than the low critical threshold,
- the fleet dimension must be greater than the sum of the low buffer thresholds.

The penalty function is determined in the same way as in the previous section and the same procedure is followed.

Also in this case, a Simulated Annealing algorithm has been chosen to solve the minimization problem.

5.3.2. The Simulated Annealing procedure

Below, a deeper description on the Simulated Annealing procedure, and of the parameters of the algorithm chosen for the optimization procedure.

The method is an iterative process that searches from a single point moving in its neighbourhood and allows worse moves to be taken some of the time, that is, it could allow, for example, some uphill steps so that a local minimum can be escaped. Worse solutions are accepted according to a probability. This probability depends on a parameter, i.e. the temperature, which decreases with the number of steps.

The algorithm evolves by means of an iterative cycle, in which the search space exploration is performed. This search depends on a control parameter called temperature T which decreases as the number of the iteration of the cycle increases. In each iteration, a new point $\mathbf{s_n}$ is reached from **s**, according to the transition rule. At the new point, the value of the cost function f is checked.

Since the cost function does not have an explicit formula, *at each step of the Simulated Annealing algorithm*, *the simulator is recalled* in order to calculate the users' waiting times and the relocation times, input data of the cost function. A diagram showing this key aspect, for a bi-dimensional search space, is provided in figure 5.5.

At each iteration of the Simulated Annealing, the following two variables are taken into account and updated:

- the generated new solution $\mathbf{s_n}$;
- the current solution $\mathbf{s}$

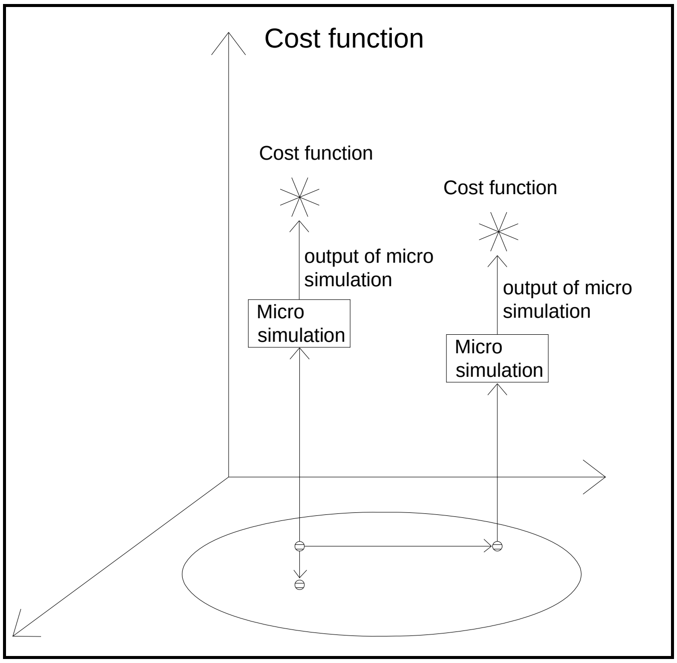


Fig. 5.5. General scheme on the evaluation of the objective function. At each iteration of the optimization algorithm, the optimization code recalls the micro simulation. The simulation's outputs are used to calculate the cost function.

The updating happens in this way:

a) if $h(\mathbf{s_n}) \le h(\mathbf{s})$ → $\mathbf{s_n}$ substitutes $\mathbf{s}$, i.e. $\mathbf{s} := \mathbf{s_n}$

b) if $h(\mathbf{s_n}) > h(\mathbf{s})$ → $\mathbf{s_n}$ will become the current solution $\mathbf{s}$ with a probability given by:

$$p = \exp\left(- \frac{h(\mathbf{s_n}) - h(\mathbf{s})}{T} \right) \qquad 5.16.$$

This is the core of Simulated Annealing and is known as the Metropolis algorithm. T is the value of the temperature for the current inner cycle (Laarhoven, Aarts, 1987).

Given that $r \in [0,1]$ is a pseudo random number, updating happens according to the following:

if $r \leq p \rightarrow$ the new solution $\mathbf{s_n}$ substitutes **s**,
if $r > p \rightarrow$ the new solution $\mathbf{s_n}$ is rejected and therefore **s** will not be updated.

Therefore the algorithm needs the definition of the cooling schedule, the local search and the starting and stopping conditions.

5.3.2.1. The cooling schedule.

The cooling schedule is defined by: the initial temperature, the law of its decrease and the final temperature. We fixed the cooling schedule in such a way to guarantee a good exploration of the research space.

The starting temperature has been determined according to Laarhoven and Aarts (1987). An initial acceptance ratio p_0 of the worse solution, is fixed at the first step of the algorithm. Thence, the initial temperature T_0 is determined from the acceptance ratio p_0 in this way, according to Laarhoven and Aarts (1987):

$$p_0 = \exp\left(- \frac{h(\mathbf{s_n}) - h(\mathbf{s})}{T_0} \right) \qquad 5.17.$$

An initial acceptance ratio p_0 of the worse solution, e.g. 0.5, is fixed at the first step of the algorithm. From this point, the initial temperature T_0 is determined from the acceptance ratio p_0 in this way, according to Laarhoven and Aarts (1987): the choice of the initial acceptance ratio has the purpose of performing a quite good exploration of the search space without slowing down too much the algorithm. In the management strategy involving flexible users, p_0 has been assumed equal to 0.9. In the other two management strategies, p_0 has been assumed equal to 0.5.

The most commonly used temperature reduction function is geometric: $T_{k+1} = \alpha \cdot T_k$ where T_k and T_{k+1} are the temperatures in two consecutive iterations of the algorithm (Laarhoven and Aarts, 1987). Typically, $0.7 \leq \alpha \leq 0.95$. It has been adopted:

- in the management strategy of the flexible users, a very slow cooling schedule in such a way to guarantee a good exploration of the research space: $\alpha = 0.99$.
- in the other two strategies, because the search space is smaller, in order to have a good exploration of the search space but not a too slow algorithm, α has been assumed equal to 0.9.

The final temperature scheme of the cooling schedule is instead replaced by a stopping condition. The algorithm is stopped when 100 iterations without accepting any more new solutions is reached, according to the stopping criteria given in (Laarhoven and Aarts, 1987). If this does not happen, the algorithm is stopped in any case when a high number of iteration, i.e. 5000, is reached.

5.3.2.2. The transition rule

The transition rule regards the exploration of the search space: from a given vector **s**, a new vector $\mathbf{s_n}$ is selected in the neighbourhood of **s**. The cost function value related to $h(\mathbf{s_n})$ is assessed through the micro simulator described in section 4.5.

The transition rule is probabilistic and 'blind'. It passes from **s** to $\mathbf{s}_n$ changing only one component of the vector **s**.

In the case of the *flexible users' management strategy*, the algorithm randomly determines the component of the vector to modify. It also determines whether to increase or decrease the chosen component: it is increased with a probability of 50% and it is decreased with the same probability.

In the *other two management strategies*, the vector has only three components and each component has the same probability to be selected. The first component of **s**, i.e. the fleet dimension, if selected, is increased or decreased by m, where m is the number of stations in the intervention area. The second and the third component of **s**, the low critical and low buffer thresholds, if selected, are increased or decreased by 1. The chosen component is increased with a probability of 50% and it is decreased with the same probability.

Moreover, the algorithm avoids that, in a given iteration, the vector component to change is the same as the one that has been changed in the previous iteration.

In this way, it is guaranteed that the new vector $\mathbf{s_n}$ is taken in the neighbourhood of the previous vector **s**. Keeping the neighbourhood that small allows to reach faster the optimum solution but, on the other hand, it cuts down the possibility of great improvements.

5.3.3. Parallel optimization

In the case of the *flexible users' management strategy*, no problem occurs and the optimization algorithm works properly.

In the *other two management strategies*, the overall optimization procedure has shown a serious problem: the objective function is heavily dependent on the fleet dimension (first component of vector **s**) and slightly dependent on threshold values (second and third components of vector **s**). This causes some difficulties in the calculation of the optimal solution for what regards the threshold values, which result in a the slowdown of the optimization algorithm.

Therefore, it has been decided to split the search space into two components: on a processor the objective function is kept dependent only on the fleet dimension and all threshold values are fixed, while on the other processor the objective function depends only on the low critical and buffer thresholds while the fleet dimension is kept constant. This technique is in accordance to the search space decomposition methodology (Crainic et al., 2010).

Search space decomposition refers to the case where the problem domain, or the associated search space, is decomposed and a particular solution methodology is used to address the problem on each of the resulting components of the search space. The chosen solution methodology is the Simulated Annealing for both components. The two SA processes are not fully independent and data exchange occurs at the end of each run of the algorithm. This is a simple parallel optimization technique. In fact parallel / distributed computing means that several processes work simultaneously on several processors solving a given problem instance (Crainic et al., 2010).

Therefore the transition rule described changes since on the first processor only the first component of **s** is modified while on the second processor only the last two components of **s** are modified.

The parallel optimization steps are the following:

1. The first processor receives in input the thresholds and it optimizes the fleet. The optimal fleet, given the threshold values, is calculated through Simulated Annealing.
2. When the first processor ends its optimization algorithm, the second processor receives in input the optimal fleet dimension from the first processor, and optimizes the threshold values through Simulated Annealing.
3. Then the two processors work together; they both receive in input the optimal fleet dimension calculated in step 1 and the optimal threshold values calculated in step 2. The first processor optimizes the fleet through keeping the thresholds constant and the second processor optimizes the thresholds keeping the fleet dimension constant.
4. When both the processors end their optimizations, the values of the objective function in the two minimum points are compared. The departure point for the following iteration of the parallel optimization is the point **s*** which gave the minimum value of the objective function
5. Now the first processor optimizes the fleet dimension, given the thresholds values in **s*** while the second processor optimizes the threshold values given the fleet dimension in **s***.
6. The step 4 is repeated until stopping conditions occur.

The parallel optimization is stopped if one of the following conditions occur:

1. if in a given step the values of the objective function in the two minimum points differ less than 5%,
2. or if the two processors do not find any better solution than the initial one
3. or if the current best solution (step 4) of the parallel optimization is close enough to the best solution obtained at the previous iteration of the parallel optimization

A scheme of the overall proposed methodology is shown in figure 5.6.

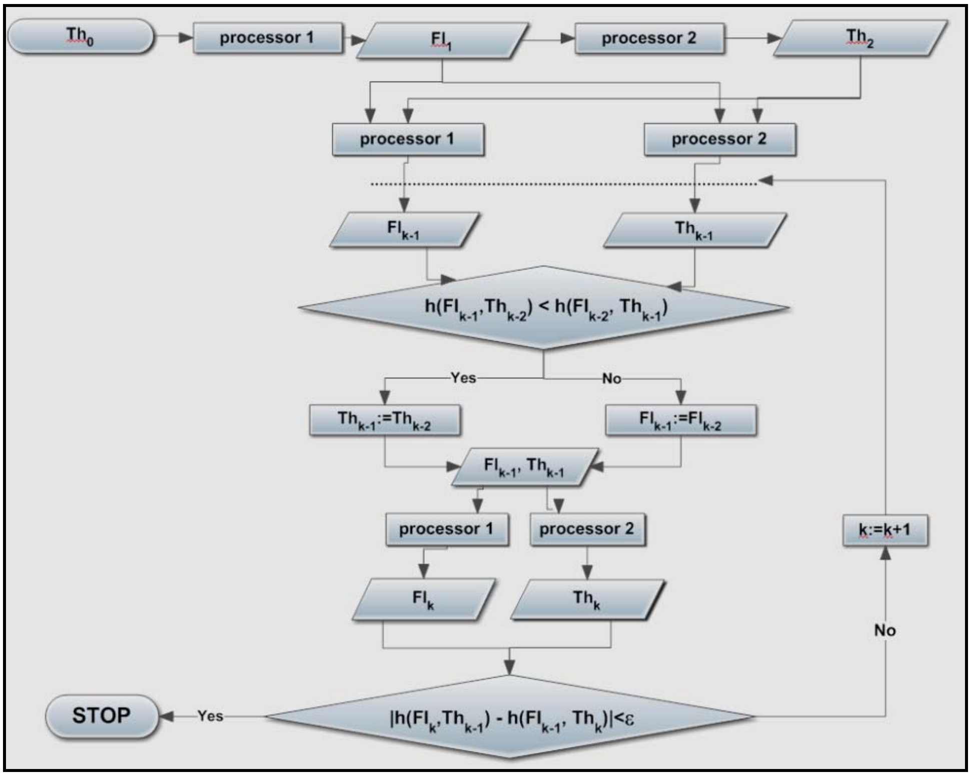


Fig. 5.6. The parallel methodology implemented in the optimization procedure. In the figure, the fleet dimension is referred as Fl, and the low critical and low buffer threshold values are referred as **Th**.

5.4. Description of the optimization code

5.4.1. Description of the code

5.4.1.1. Initialization – first iteration:

First of all, the file provides:

- Number of vehicles at each station at the beginning of the simulation time
- Low critical thresholds
- Low buffer thresholds
- Management strategy
- An indicator called flag: if the management strategy is the flexible users one, then its value is not relevant. Otherwise, in the other two management strategies, if flag is equal to 1 then the algorithm modifies the fleet dimension; otherwise if the flag is equal to 2 the algorithm modifies the thresholds.

These parameters are the input of the optimization algorithm, and they represent the quantities to optimize. They replace the same input data reported in the file inputdata.py.
Afterwards, the quantities fixed by the cost function are declared, i.e.:

- The cost of waiting, equal to 0.10 €/minute,
- Daily amortization cost of each vehicle, equal to 4.117 €/(vehicle · day). The lifetime of the vehicle is assumed equal to 8 years, and the lifetime of the battery is assumed equal to 4 years.
- Relocation cost, assumed arbitrarily equal to 0.01 €/minute. This choice is made in order to avoid to make it predominant respect to the cost of waiting.
- Constant cost, equal to 20.82 €/day, cost of the energy consumed by the vehicle and transport system management.
- μ = penalty coefficient

The following quantities are setup:

- a list which contains the waiting times of all users,
- the objective function,
- some components of the objective function, i.e. the total cost of waiting and the total cost of relocation
- 50th, 90th and 95th percentiles.

The objective function has the following shape:

$$f_{ob} = c_r \sum_j t_{r,j} + c_w \sum_i t_{w,i} + n_v c_v \left[\frac{r(1+r)^{lt}}{(1+r)^{lt}-1} \right] \frac{1}{365} + \mu \sum_i \left[\max\{0, g_i\} \right]^2 \qquad 5.18.$$

- c_r = relocation cost
- c_w = cost of waiting
- i, j = ith and jth user
- $t_{r,j}$ = relocation time of the jth user
- $t_{w,j}$ = waiting time of the jth user
- n_v = number of vehicles
- c_v = daily amortization cost of each vehicle
- r = discount rate, equal to 8%
- lt = lifetime, equal to 4 years for the battery and 8 years for the vehicle
- μ = penalty coefficient
- g_i = i-th constraint (i = 1, 2 or 3)
- moreover $\mu \cdot \sum_i \left[\max\{0, g_i\} \right]^2$ is the penalty function.

Constraints are on the percentiles of user waiting times, in order to avoid to have a LOS F, in particular:
$t_{90} < 8$, $t_{95} < 10$, $t_{50} < 4$; t_{90} , t_{95} e t_{50} are the percentiles of user waiting times. The waiting time is expressed in minutes. The function **g** is therefore:

$$\mathbf{g} = \begin{pmatrix} t_{95} \\ t_{90} \\ t_{50} \end{pmatrix} - \begin{pmatrix} 10 \\ 8 \\ 4 \end{pmatrix} \qquad 5.19.$$

The code executes for 30 times the following instructions and averages the obtained results:

- recalls the method “simulation” of the simulator’s code.
- updates the list of users’ waiting times
- calculates the objective function without the penality function
- calculates the user waiting times
- calculates the relocation costs
- calculates the 50th, 90th and 95th percentile.

30 iterations are performed in order to reduce the stochastic effects in the simulator’s outputs.
At the end of the iterations, the average value of the objective function and of percentiles is calculated. Afterwards, if the constraints are not satisfied, the objective function is updated with the penalty parameter.

5.4.1.2. Second iteration:
The values of: fleet dimension, low critical thresholds, low buffer thresholds, related to the previous step, are inserted in lists but named with the same name but added “old”: for example: *fleet dimension old* instead of *fleet dimension*, *low critical thresholds old* instead of *low critical thresholds*.

If the management strategy is different from that of flexible users, and if the flag indicator is equal to 2, then the simulator will modify the thresholds. In order to do so, a casual number i comprised between 0 and 1 is generated. If $i < 0.5$, then the algorithm will act on the buffer thresholds; otherwise it will act on critical thresholds.

Afterwards, in both cases, a new random number j comprised between 0 and 1 is also generated. If $j < 0.5$, then all thresholds are decreased by an unit; otherwise they are increased by an unit. However, this happens under these limits: the fleet dimension must be greater than the sum of buffer thresholds, and the sum of buffer thresholds must be greater than the sum of critical thresholds. Moreover, the thresholds value is assumed the same in all stations.

If instead the flag indicator is equal to 1, then the algorithm acts on the fleet. Again, a random number i is generated, and it is comprised between 0 and 1. If $i<0.5$, the number of available vehicles in all stations at the beginning of the simulation time is increased by an unit. Otherwise, this number is decreased by an unit.

If instead the management strategy is that of flexible users, firstly a number i comprised between 0.000001 and the number of stations is generated. This number is rounded to the following integer and provides the identifier of the station in which the number of available vehicles at the beginning of the simulator will be modified. Another random number j is generated comprised between 0 and 1, therefore: if $j<0.5$, then in the chosen station the number of vehicles will be decreased by 1. Otherwise, it will be increased by 1.

Afterwards, in all examined cases, with the new values of thresholds and fleet dimension, the code recalls the simulator 30 times as exposed in the initialization. The objective function will be calculated and also the percentiles, in the same manner as in the initialization.

At the end of the second iteration, the Simulated Annealing parameters will be calculated. A value of acceptance ratio, i.e. probability to accept worse solutions, is considered, initially equal to a given value, e.g. 0.8, which will be decreased with the execution of the algorithm. The initial temperature T_0 is calculated as in the following:

$$T_0 = \frac{f_{ob}^{new} - f_{ob}^{old}}{\log(1/a_r)} \qquad 5.20.$$

Where f_{ob}^{new} is the objective function calculated in the current iteration, f_{ob}^{old} is the objective function calculated at the initial step. Instead a_r is the initial acceptance ratio.

After determining the initial temperature, the algorithm decides whether to accept or reject the new solution.

- If the new solution is better than the old one, then the new solution will be the departure point for the following iteration
- If the old solution is better than the new one, then:
    - A random number is generated, extracted from an uniform distribution and comprised between 0 and 1
    - If this number is smaller than the initial acceptance ratio, then I accept the new solution
    - Otherwise I reject the new solution and the initial one will be the departure point for the following iteration.

5.4.1.3. Body of the algorithm

The algorithm performs 500 iterations, because it is assumed that at the end of the 500 iterations it may converge to a local optimum. Two counters are introduced:

- Local optimum, which takes into account the number of performed iteration without that a new solution is selected
- Previous solution, which avoids that the algorithm jumps between two near solutions without converging.

Firstly it is decided what to variate, i.e. which of the two thresholds, or the fleet, and of which quantity, exactly as previously described for the second iteration.

Then the new values of the fleet and of the thresholds are provided in input to the simulator. Then the objective function and the percentiles are calculated. Finally the temperature is decreased: $T_{k+1} = \alpha\ T_k$: k is the step of the algorithm, α is a coefficient comprised between 0 and 1, which has been assumed equal to 0.9. it is necessary to be extremely careful both regarding this coefficient and the acceptance ratio, because they provide the velocity of convergence of the algorithm. If indeed the values of the acceptance ratio a_r end of the α coefficient are too low, the risk to find a bad solution, i.e. a bad local optimum, is very high. If these values are too high, instead, the risk is to be far from the termination of the algorithm still after 500 iterations.

Finally, the objective function is evaluated in the same manner as after the second iteration, in order to determine whether to accept the new solution or to reject it. If the new solution is accepted, the vectors fleet dimension and thresholds will be namedwith “old”, and also the objective function will be named as “old”, i.e. *objective_function_old.* Moreover the previous solution is saved in the list *previous solution*. In this way, if in the following step the simulator determines a new solution equal to this one, automatically it will be rejected.

Moreover, at each iteration where the new solution is not chosen, the counter *local optimum* will be increased by 1. When the counter *local optimum* reaches the value 100, i.e. in all the 100 previous solutions the algorithm couldn’t find any better solution, the algorithm is terminated, and the last accepted solution is chosen as the optimum solution.

5.4.2. Debug of the algorithm

The optimization algorithm has been verified for correctness.

The optimization algorithm takes in input the output quantities calculated from the simulator. However, due to the high stochastic effects coming from the simulation, the simulator is run several times, in order to cope with these effects and the outputs of the simulator are averaged. At each simulator’s run, the distribution of users waiting times and the total relocation time are calculated. These quantities are stored in lists which are updated and averaged iteration by iteration with the new quantities.

The number of required iterations to cope with stochastic effects has been determined in the following way. The average waiting time, and the 50th, 90th and 95th percentiles of the distribution of users’ waiting times, the total users’ waiting time and the total time spent in relocation have been registered in each iteration of the simulator. All these quantities have been averaged iteration by iteration. At each iteration, the variations between the values obtained from the simulator at the current iteration and the averages calculated so far, are determined. If the greatest of these variations is less than 5%, then the number of iterations of the simulator performed is enough. Several replications of this procedure have been performed and the results show that with 23-27 iterations of the simulator it is possible to cope with all the stochastic effects.

The stochastic effects in the simulator come from the simulator input data. The transport demand in the simulator is provided in the form of OD matrixes, i.e. the number of users that desire to travel from a given station or unit to a given station or unit during an hour in a given period of the day, is provided. But the exact time instant in which the user arrives in the origin station or unit is extracted from an uniform distribution. This choice is made because sometimes the transport demand is extremely low, specially in the case of the third management strategy, in which there is a great number of units. For example the number of users which desire to travel from e.g. station 7 to station 9 is 5 users by hour, or the number of users which desire to travel from unit 5 to station 3 is e.g. 2 users per hour.

The key aspects tested regard that the optimizer correctly recalls the simulator, and that provides the right inputs to the simulator and deals with the correct outputs. This control is performed also in the simulator code, that all lists, libraries and quantities are reset at the beginning of each simulator run. Indeed, because the objective function is calculated as an average of several runs of the simulator, in order to reduce stochastic effects, if all these quantities are not reset, they take memory of the previous iterations and therefore the statistics' values may be strongly affected.

The last key aspect in the optimization code's debug regards that the Simulated Annealing scheme is correctly implemented. In particular, it has been checked if the rule for the choice of the new solution has been correctly implemented, i. e. that the code correctly decides whether to accept or reject the new solution, according to the simulated annealing scheme. Moreover, it has been verified that the algorithm really follows the correct cooling schedule. And finally, in order to avoid that the algorithm "jumps" between the same two or three solutions, without determining the optimum one, a memory of the solutions previously determined is taken, and these solutions cannot be accepted again in following steps of the algorithm.

The calibration of the Simulated Annealing parameters, such as the cooling schedule, is instead described in the chapter 6 of this thesis.

Chapter 6. Calibration, validation and coherence analysis of the proposed transport system

Introduction

The main phases of the development of a model are:

1. Specification of the model: i.e. definition of the main characteristics of the model;
2. Calibration of the model's parameters;
3. Validation of the model.

This chapter will deal with the second and the third phases exposed above.

Firstly, the calibration will be described. It is necessary to calibrate two models: the micro simulator and the optimization algorithm.

For what regards the simulator, the calibration of the simulator's parameters, as well as the calibration of the sub-models contained in the simulator, i.e. the PICAV velocity model and the model of the battery, will be described.

For the model under study, a similar system still does not exist and therefore it is impossible nowadays to validate the proposed model. Therefore, an analysis of coherence of the model's assumptions results necessary.

6.1. Calibration of the optimization algorithm and of the micro simulator.

6.1.1. Introduction on the problem

As stated above, the models to calibrate are: the optimization algorithm, and the micro simulator. The micro simulator is developed as a decision support tool for a political stakeholder. The micro simulator can be used:

- stand-alone, to evaluate the possible different scenarios;
- as an inner procedure the optimization algorithm. The optimizer in fact at each iteration recalls the simulator to calculate through it the cost function.

It is necessary firstly to clearly distinguish between the input data and the parameters.

Parameters of the optimization algorithm.

The parameters of the optimization algorithm are the parameters of the Simulated Annealing, i.e. the cooling schedule and the transition rule.

Parameters of the micro simulator.

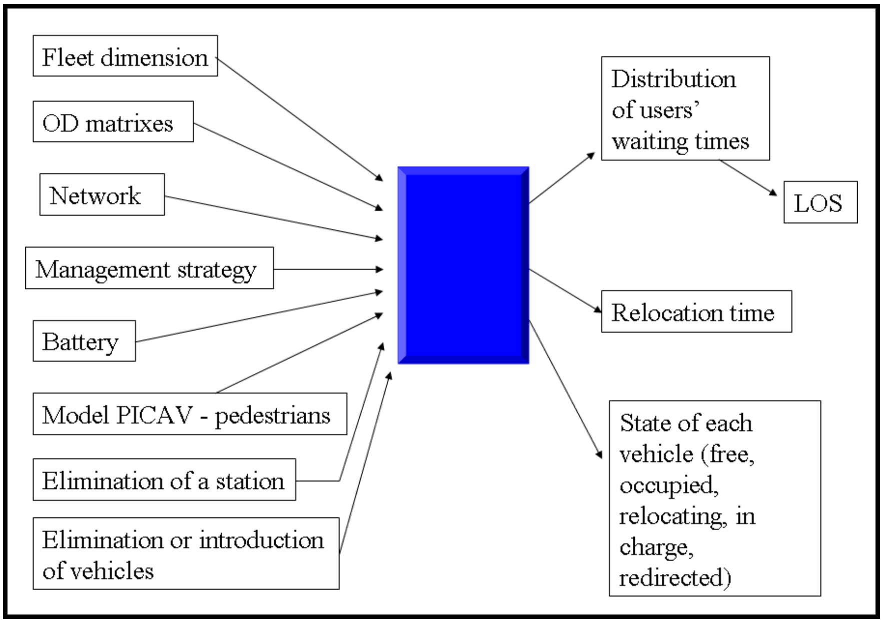
The micro simulator receives in input:

- fleet dimension;
- transport demand, for what regards the demand OD matrixes;
- major modifications, such as elimination of a station;

- battery capacity;
- model PICAV vehicle – pedestrian density;
- characteristics of the network: stations, units and length and slope of the paths among stations and units.

Output of the model are:
- waiting time, from which the Level of Service depends;
- relocation time.

A general scheme of this is provided in fig. 6.1.


Fleet dimension
OD matrixes
Network
Management strategy
Battery
Model PICAV - pedestrians
Elimination of a station
Elimination or introduction of vehicles
Distribution of users' waiting times
LOS
Relocation time
State of each vehicle (free, occupied, relocating, in charge, redirected)
Fig. 6.1. General scheme of inputs and outputs of the simulator.

The parameters of the micro simulator are therefore related to:
- values of the thresholds;
- the law for the users' arrival times;
- the battery management chosen, the battery charge and discharge laws;
- the parameters of the model of interaction vehicle – pedestrians.

6.1.2. Calibration of the micro simulation model

6.1.2.1. Calibration of the threshold values

The threshold values, are calibrated for each scenario by the optimization procedure.

The *data collection* for the calibration of these parameters coincides with the assessment of the transport demand and of the network for the two scenarios of

Genoa and Barreiro. Details on the data collection for what regards Genoa and Barreiro scenarios are provided in the chapters 7 and 8.

6.1.2.2. Calibration of the law of users' arrivals

As stated above, the demand OD matrix provides the number of users having a given origin and a given destination in a predefined time interval. Therefore, for each user the origin and the destination are provided, and the time period in which he arrives at the origin station.

The exact simulated time instant within the interval when the user arrives at the origin station is assessed through a probability distribution. The most popular distribution to determine random arrival times is the Poisson. However, in the situation under study, there are some couples of stations or units which have a much low demand, such as 3 – 4 users by hour, and therefore the important aspect is to distribute their arrivals efficiently within the hour. With a uniform distribution, all the instant of the hour have the same probability of being chosen. Instead, using a Poisson distribution, the arrivals tend to be mostly concentrated around the average. Therefore, if we have for example a demand of 3 users by hour from the station 7 to the station 8, if we choose a uniform distribution we can hope to have their arrival times distant about 15 – 25 minutes. If instead we choose a Poisson distribution, we risk of having all users to arrive around the same moment.

6.1.2.3. Calibration of the battery management

The battery charging technique chosen is the opportunity charging. The term opportunity charging refers to the charging of the batteries wherever and whenever power is available. Simply put, rather than waiting for the battery to be completely discharged, or for the duty cycle or work shift to be over, opportunity charging is the "power as you go" opportunity to extend the capabilities of your equipment during every stop in a station. Several run of the micro simulation have been performed and the results have shown that, thanks to this technique, the vehicles have always a charge level which is greater than the minimum and therefore they can all be available to users when peaks of demand occur.

Fort what regards the minimum battery level for performing the trips, it strictly depends on the case study, in particular on the demand and on the network. Therefore, this is an input datum and not a parameter and it does not have to be calibrated.

6.1.2.4. Calibration of the models contained in the simulator: the battery charging and discharging law and the model of interaction PICAV - pedestrians

The model of interaction vehicles – pedestrians will be described in detail in the next section.

The battery charging and discharging laws have been already validated and they belong to the literature in the field.

6.1.3. Calibration of the model of interaction vehicles/pedestrians

The model for the interrelation vehicle velocity – pedestrian density has been determined in an experimental way. Because the vehicle still has to be constructed in the moment in which this model has been realized, it has been assumed that it has the same performances as an electric scooter for mobility impaired people. It has also been assumed that pedestrians respond in the same way to the electric scooter and to the PICAV vehicle, as the space occupied by the two vehicles is more or less the same.

6.1.3.1. Data collection

The scooter has been driven in a crowded environment, in particular in a section of via San Luca and of via di Canneto in Genoa, two of the most crowded roads in the historical city centre of Genoa. Via San Luca and via di Canneto are typical pedestrian-only roads. Data collected in via San Luca have been used for the calibration of the model, data collected in via di Canneto have been used for the model's validation. A screenshot of these videos is reported in fig. 6.2 for via San Luca and 6.3 for Via Canneto.

A series of video clips have been collected of the electric scooter travelling in a straight line. In that part, via San Luca is wide just over 2.5m. The surface of the street is also typical: cobbled stone. These videos were taken from a birds-eye point of view and took in an area of 18.75m^2, which was divided into 12 squares (6 squares long and 2 squares wide).

The videos have been performed: in the morning from 10:30 to 11:30, which has resulted to be an off-peak period; in the afternoon from 15:30 to 17, which has resulted to be a peak period, with much higher values of pedestrian density. The videos allowed us to determine the average pedestrian density to consider in the two periods, i.e. the morning period (off-peak) and the afternoon period (peak). This distribution of pedestrian flows is due to the high number of commercial activities presents along via San Luca, which becomes therefore very attractive in the afternoon, after people have come back from work.

The analysis was performed by watching the video clips and each run as timed from the moment the scooter entered the 18.75 m^2 area of interest until the moment it left. During this time the average pedestrian density was also recorded. Values of density are expressed as the number of pedestrians per square metre. Rides were completed in both upslope and downslope directions. Eight rides were filmed in the morning, and 36 in the afternoon. Usually, morning rides refer to low values of pedestrian density, while afternoon rides were characterised by high values of pedestrian density. The rides are the following. The entrance film time and the exit film time refer to the time reported in the video tape for when the scooter entered and exited the area of interest. It is expressed in this way: mm:ss:d. where mm = minutes ; ss = seconds ; d = decimal, i.e. 01:10:9 means: 01 minutes, 10.9 seconds.

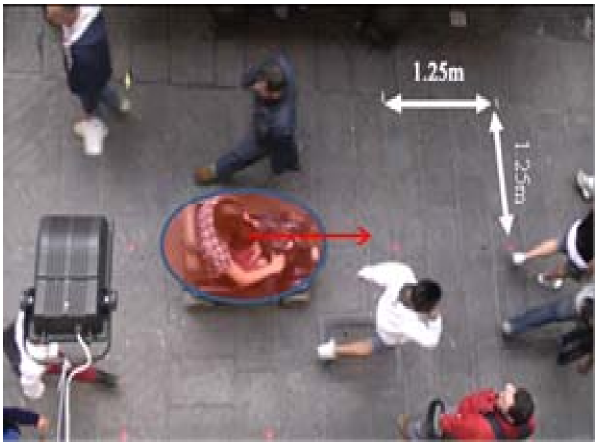


Figure 6.2. The electric scooter in the pedestrian area of Via San Luca

Figure 6.3. The area filmed in via di Canneto

6.1.3.3. The density-velocity model calibration

The empirical data have been filtered to exclude three runs in the morning and other three in the afternoon. These rides were different from all others as some disturbances occurred. The remaining rides have been used to create the relationship shown in figure 6.5, which shows the average pedestrian density on the x-axis and scooter velocity on the y-axis. The recorded data are displayed in the figure 6.4.

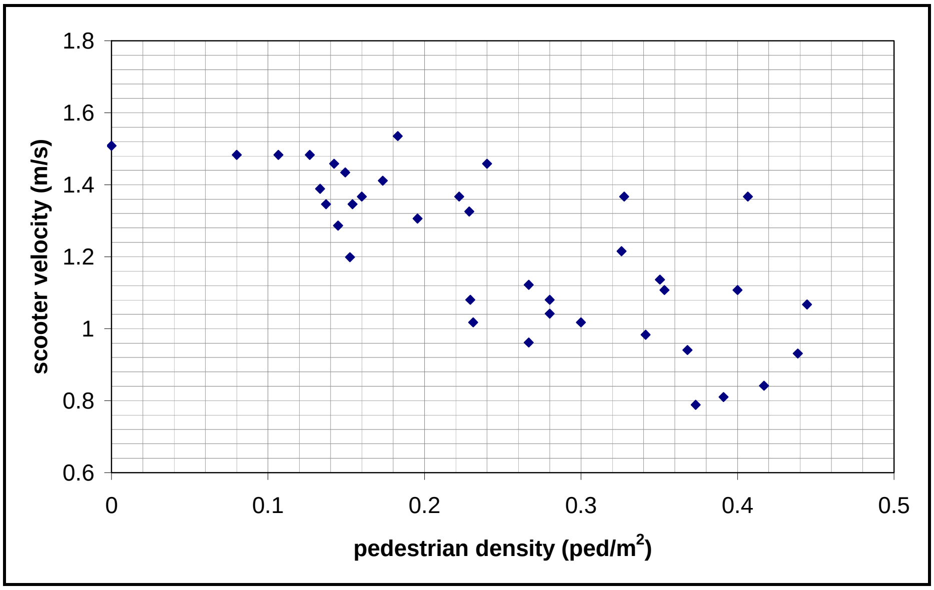


Fig. 6.4. The recorded data vehicle speed – pedestrian density in via San Luca.

Several relationships between speed and density were explored using the statistical package R. In fact the linear model between scooter speed and pedestrian density provides an R^2 equal to 0.55, which is not satisfactory. The models explored were:

1. Linear model between: scooter speed on one side; and on the other side pedestrian density plus an index being 1 if the scooter is going upslope and 0 if it is going downslope:

   $v = a\,k + g$, where g = 0 or 1. 6.1

2. Polynomial models between scooter speed and pedestrians density, where v = speed, k = density. One such model could be e.g.:

   $v = a\,k^4 + b\,k^3 + c\,k^2 + d\,k + e$ 6.2

   where a, b, c, d, e are some coefficients.

3. Logarithmic and exponential model: e.g.

   $v = a\,exp(k) + \mathrm{b}$ or $v = a\,ln(k) + b$ 6.3

4. Discontinuous linear model: e.g. $v = a\,k + b$ where a and b have two values, e.g. there is a threshold in the density, called k_0; therefore:

   $a = a_0$ and if $k < k_0$ and $a = a_1$ if $k \geq k_0$
   $b = b_0$ and if $k < k_0$ and $b = b_1$ if $k \geq k_0$

Several values of k_0 have been explored.

The number of available observations to calibrate the model was 44.

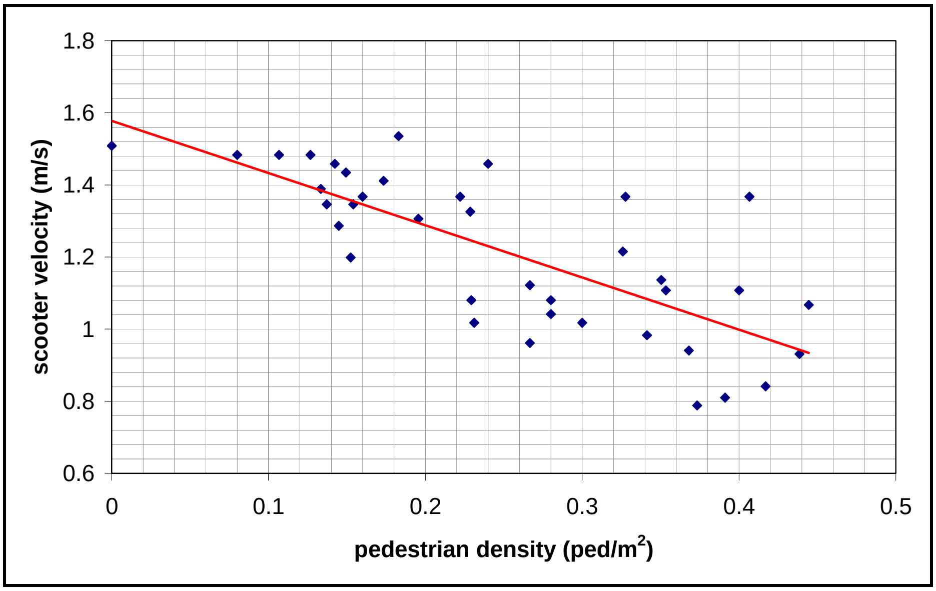


Figure 6.5. Model of pedestrian density with reference to scooter velocity

At the end the linear model $v = a\ k + b$ has been chosen, although it provides a quite low adjusted R-squared value, equal to about 0.55, but it provides a high level of significance for both the constant and the variable (i.e. the density). In fact, when the further models were investigated it was found that the increase in complexity of the model did not result in large improvements of the adjusted R-square value. Increased values for the adjusted R^2 were found (0.58 – 0.59), but the increase in the R-squared value was so small when compared to the increase in complexity, that it was decided to use the linear model. The final model is defined by the following equation:

$$v = 1.58 - 1.45\ k \tag{6.4.}$$

where v is the velocity of the scooter (m/s) and k is the density of pedestrians (number of pedestrians /m^2).

This model should however be improved: data available were extremely reduced, as they count only 44 drives, and it would have been necessary instead to perform various campaigns of videos in different days.

Another problem of the model calibration is that the range of pedestrian density in which the various experiments have been developed was quite little, and moreover, because of the small number of observation, it has been impossible to take into account the desired speed, i.e. the speed that the scooter's driver would have kept in case of zero density.

This model is applicable to a PICAV vehicle driven by a person. Indeed the scooter maximum speed, which is about 1.6 m/s, therefore equal to 5.76 km/h, is highly less than 10 km/h, which is the average operative speed of the PICAV vehicle. Instead, the PICAV vehicle automatically driven can reach at maximum a speed of 4 km/h, as at a greater speed the sensors and laser scanners which allow it to avoid collisions cannot work. Therefore, a new relationship has been considered for the PICAV automatically driven:

$$\begin{aligned} &v = 1.38 - 1.45\, k \quad \text{if } k > 1 \text{ pedestrian/m}^2 \\ &v = \text{cost.} = 1{,}11 \quad \text{if } k \le 1 \text{ ped/m}^2. \end{aligned} \qquad 6.5$$

The videos allowed us also to assess the values of pedestrian density for the Genoa case study. Several videos have been taken in various roads of the city. The highest pedestrian density among the videos collected has been registered in via San Luca, and it is equal to: 0.15 ped/m^2 in the off-peak period, from 8 am to 4 pm; and 0.35 ped/m^2 in the peak period, from 4 pm to 8 pm.

6.1.4. Calibration of the optimization parameters

The parameters of the optimization algorithm, which have been introduced in section 5.2, have been calibrated. The parameters to calibrate are:

- Those which regard the cooling schedule:
  - the initial temperature T_0,
  - the law to assess the decrease of the temperature,
  - the final temperature, i.e. the stopping condition.
- The transition rule.

All these parameters have been calibrated through several initial runs of the algorithm. The purpose was to reach a good solution without spending too much time; also because at each iteration the algorithm recalled the simulator code and this made the overall optimization quite slow.

6.1.4.1. Calibration of the cooling schedule

The starting temperature T_0 has been assessed from p_0 according to the following formula:

$$p_0 = \exp\left(- \frac{h(\mathbf{s_n}) - h(\mathbf{s})}{T_0} \right) \qquad 6.6.$$

The probability to accept the new solution p_0 has been set equal to:

- 0.9 for the first management strategy, i.e. flexible users
- 0.5 for the other two strategies.

Conversely, the decrease of the temperature is assessed according to the following law: $T_{k+1} = \alpha \cdot T_k$ where T_k and T_{k+1} are the temperatures in two consecutive iterations of the algorithm. α has been taken equal to:

- 0.99 for the first management strategy,
- 0.9 for the other two strategies.

These values have been chosen in order to reach a good solution without spending too much computational time. In the first case, the search space is very large and therefore α must be closer to 1 to perform a better exploration. In the other two cases, the search space is much smaller therefore it is not necessary to keep α so large. Moreover, in the last two cases, the algorithm is run a few times to perform the parallel optimization, described in section 5.2.3, therefore it is necessary to keep quite low the computational time of the algorithm. These values have been assessed after about 30 runs of the optimization code for each management strategy.

The final temperature, has been assessed in this way:

- For the first management strategy, the algorithm is stopped when 5000 iterations are performed, or after 200 runs in which the algorithm never chooses a new solution
- For the other two management strategies (automatic relocation and capillarity), the algorithm is stopped after 500 iterations, or after 50 runs in which the algorithm never chooses a new solution.

Taking a higher quantity of runs of the algorithm does not improve the quality of the solution. These choices for the final temperature have been made after about 30 runs of the optimization algorithm for each management strategy.

The different cooling schedule in the management strategies is performed for these reasons:

1. In the first management strategy, the search space has several dimension (in the case study of Genoa it has 9 components); therefore it is first of all important to achieve a good exploration of it. In the other two management strategies, the search space has 1 or 2 dimensions and therefore it is not necessary a too slow cooling to reach a good solution
2. In the other two management strategies, a parallel implementation is performed; therefore, it is important to have a quick enough algorithm as it must be run a few times before reaching the correct solution.

6.1.4.2. Calibration of the transition rule

Regarding the transition rule, the calibration has been decided not after algorithm's runs but the choices made are the best from the coherence point of view. The chosen transition rule has been already explained in detail in chapter 5.

6.2. Validation of the model under study

6.2.1. Theoretical introduction on the validation problem

Each simulation model must be validated. It is therefore necessary, after its development, to determine whether the proposed model is a valid representation of the reality under study. Validation is not the debugging of the simulator, indeed this is called verification, but the control if the conceptual model is an accurate representation of the system under study. If a model is validated, it is also credible: it is difficult to give credibility to a non validated model.

According to Law and Kelton (1987), in order to develop a valid and credible model, the following issues should be addressed.

Develop a model with high face validity:

A model with high face validity on the surface seems reasonable to the people who have knowledge about the system under study. In order to achieve this, the following must be performed:

- conversation with system experts: a simulation model is not an abstraction developed by an isolated analyst: the point of view of people who have experience or data about the topic modelled must be taken into account;
- observations on the system: if a system similar to the one of interest exists, then data should be obtained from this system to validate the model. Care must be taken to ensure that the data taken from the "similar" system can be valid also for the model under study;
- existing theory: it is necessary to refer to the data and to the knowledge coming from the literature;
- relevant results from similar simulation models should be taken into account;
- experience on how some components of the model work;
- be attentive to model the right problem: the most effective model for the wrong problem is invalid;
- perform a structured walk-through of the conceptual model prior to beginning of coding, to ensure that the model assumptions are correct, complete and consistent. This is extremely important in order to avoid making significant changes to the code.

Test the assumptions of the model empirically:

The assumptions made on the model should be tested quantitatively.

If a probability distribution for the input data is available, the adequacy of fit can be assessed by the graphical plots. For example, if several sets of data have been observed for the same random phenomenon, then the correctness of merging these data can be determined by the Kruskal – Wallis test of homogeneity.

An useful tools to test the assumptions of the model is the *sensitivity analysis*. This can be used to check whether the simulation output changes significantly if the value of an input parameter is changed, or if the level of detail of a subsystem is changed. If the output is sensitive to some aspects of the model, then these aspects should be modelled carefully.

Determine how representative the simulation output data are:
The most definitive test in establishing the model validity is to check if the output data resemble the expected output data from the system. If a real system exists, then a simulation model of the system is developed and its output data are compared to those from the existing system itself. If the two sets of data are comparable, then the model can be considered valid.

There is not any definitive approach for validating the model: if there were, there might be no need for a simulation model.

In order to compare the data coming from the simulator with those from the existing system, it is possible to use a Turing test. People knowledgeable of the system are asked to examine one or more sets of data without knowing which sets are which. If the experts can distinguish between model data and system data, then their explanation can be used to improve the model.

If a similar system does not exist, then it is necessary again to have a revision from experts to check the reasonableness of data.

6.2.2. Validation of the simulator

The systems to validate are the three car sharing systems exposed in the previous sections, in particular:

1. Flexible users, redirected to the best station from both the user's and the system's perspective, user trips have origin and destination at stations.
2. Automated vehicles, which relocate automatically among stations; user trips have origin and destination at stations.
3. Automated vehicles, which automatically reach the user position; user trips have origin and destination also along the roads.

Each simulation model must be validated. However, the car sharing system proposed does not exist in reality because:

- no such car sharing systems have been settled yet in Genoa or in Barreiro;
- similar car sharing systems which implement the same management strategy still have to be developed;
- as the technology of vehicle automation is still under development, no systems involving automated vehicles exist.

Therefore, there are no data available to validate the simulator so far. However, as the proposed system will be implemented, a validation of the model will be possible.

For what regards validation of a model without availability of data, there are two main points of view.

The first point of view considers that a model is a “black box”: the model has some precise inputs, some precise outputs, the internal structure of the model is not important: also if the hypotheses that it makes are not correct, it works if its outputs are always close to the expected values. For example, the models for the light. The light is assumed to have two natures: corpuscular and undulatory. Clearly, the light has none of these two natures, but these models, although based on wrong assumptions, represent very well the behaviour of the light. Moreover, if a model is composed of some sub-models and all of them work, it may be possible that a model on the whole does not work.

The second point of view considers that: if the model cannot be entirely validated, then let’s validate its components. If all the assumptions made on the model are correct, if all the model’s components are validated, and if all components are put together to constitute the model in the proper way, then the model with high probability is valid.

I think that the second approach is better. It is true that some models based on wrong assumptions may work, but these are just limited cases. On the other hand, it is also quite rare that if all components of a model are valid, the overall model does not work.

As a result, I have chosen to validate the two sub models contained in the simulator, and to test for coherence also the hypotheses made. However, in order to be more secure on the results, a conversation with experts in the field is foreseen.

The sub models contained in the simulator are the model of the battery and the vehicle – pedestrian model.

The model of the battery has been already validated and applied several times. For what concerns the relationship vehicle speed – pedestrian density implemented in the simulator, it will be validated herewith because there are data available.

An analysis of coherence of the model’s assumptions has been performed and will be reported in the next section.

6.2.3. Validation of the speed-density relationship.

The relationship vehicle’s speed – pedestrian density has been validated by taking into account data collected in via di Canneto.

Several video tapes have been recorded. The area was equal to 49.95 m^2, and in particular it is 11.05 m long and 4.5 m wide. The road videoed is via di Canneto, besides Piazza Banchi. The road is wider in this part than via San Luca, but the pedestrian density is still high, though the density values registered in via San Luca has not been reached.

The available rides of the scooter for validation are 14. Other two rides have been deleted because of disturbances, one due to human factors of the driver and the other to an external disturbance. Differently from Via San Luca, the videoed area is flat.

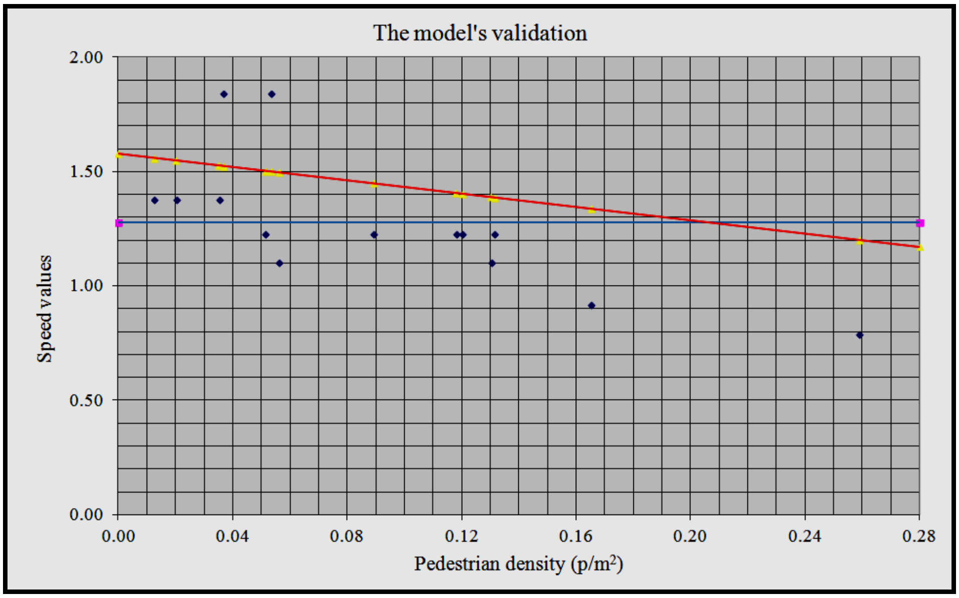


Fig. 6.6. The pedestrian model’s validation. Data coming from the observations in via Canneto are the blue squares. The expected values of velocity (from the model) for the given value of density is represented by the yellow triangle. The red line represents the model, the blue line is the average velocity recorded in via Canneto.

The couples density – speed have been graphed and they correspond to the blue points in the fig. 6.6. The points expected from the model are the yellow ones. The line of the model is the red line. The average of the recorded speed values is the blue line, which is equal to 1.28 m/s.

From the validation, an R^2 value has resulted equal to 0.47. which is slightly less than the model R^2, i.e. 0.55. The model has resulted validate, thought the R^2 of the model is low (0.55 is not much). As a result, the model must be improved; however, human factors can strongly affect data and therefore a “perfect” model, i.e. with an R^2 about 0.8, is much difficult to achieve.

6.3. Analysis of coherence of the model’s hypothesis

An analysis of coherence of the hypotheses of the model has been performed. The hypotheses taken into account are the following.

6.3.1. Relocation strategy

6.3.1.1. Necessity to relocate
In order to determine the necessity of some relocation schemes, some runs of the algorithm have been performed without any relocation scheme implemented. The average waiting time resulted equal to about 40 minutes, the 90th percentile about 75 minutes and the 95th about 135. Moreover, also several vehicles were redirected because the destination station was full.

6.3.1.2. Flexibility of users
An analysis of coherence has been performed, whether the users can be considered flexible. Ten people have been interviewed in the historical city centre of Genoa, the project has been explained in detail, and it has been asked them whether they accept to be redirected to another station according to the system's needs. All people answered that they accept to be redirected because this reduces the waiting time: for them it is a greater disutility to wait longer than to be told the place in which to return the vehicle. What came out from these interviews is the extreme importance of clearly defining the users' choice set. For example, with reference to Genoa scenario, all people answered that for them Piazza Fontane Marose, Piazza De Ferrari and Piazza Dante, i.e. stations 1, 2 and 3, were indifferent to return the PICAV vehicle, but for Darsena not everyone agreed, and only people having destination the west neighbourhood of Genoa (not also people having destination the north of Genoa) accepted to be redirected also to Piazza Caricamento.

As a result, to consider users flexible is acceptable, but the users' choice set must be defined carefully. Moreover, not in all scenarios users can have a choice set wide enough, and for example in Barreiro several stations were not interchangeable.

6.3.1.3. Automation of vehicles
Actually the technology to allow vehicles on the road without being driven is still under development. If the road is much congested, the laser scanners do not work properly and therefore vehicles cannot move. To resolve this problem, it is necessary to design some paths in which pedestrian flows are low and therefore vehicles are able to move. However, the technology is improving a lot in this field and it is possible in a near future to have automated vehicles moving on the roads.

The aspect which instead must be resolved regards the legal problems to determine the responsibility in case of an accident with an automated vehicle.

6.3.1.4.. The opportunity charging technique
The opportunity charging technique has resulted to perform very well as it exploits all the times in which a vehicle is idle at a station. In the two scenarios of Genoa and Barreiro, the vehicles' battery level is always above the minimum and therefore, in correspondence of the peaks of demand, all vehicles are always

available for user trips and relocation trips. However, the choice of the battery must be very careful, as if batteries have a memory effect, the opportunity charging technique may heavily affect the battery duration.

6.3.1.5. The network representation

The road network in transport sciences is usually represented through nodes and links, with centroids playing the role of origin and destination of the demand. In this case, we have no links and several centroids, each corresponding to a road unit or to a station. The links are not represented in detail and instead only some informations about the distances and the slopes between each pair of units or stations (i.e. about the path) is provided. This representation is cost effective only if a little information on the path is needed. If for example we want to add to each path several informations it is necessary each time to take into account all the road parts for each path and therefore this is not cost effective at all. On the other hand, the modelization of the roads through nodes is necessary in order to assess precisely where the PICAV vehicle is returned or parked. Therefore, some more cost effective representations of the network, in order to cope with a high number of origins and destinations on one hand, and with the possibility to add much information to each path on the other, must be investigated.

6.3.1.6. The speed density relationship and its features

The relationship between vehicle speed and pedestrians density has been validated quantitatively by considering some other data sets taken in via di San Luca in Genoa city centre. The model has shown to represent quite well the reality. The transferability of the model to other roads of the same area has been evaluated, and indeed the same model works well also in Via di Canneto, another road of Genoa centre.

The problem which regards the speed–density relationship is the impossibility to consider the same pedestrian density for all the roads of the intervention area, without making any difference among the roads. This is a strong assumption and therefore the simulator must be modified in order to allow to define different density values to be assessed in the various roads of the area.

Chapter 7. Genoa case study

Introduction.

The car sharing system has been planned and simulated for two different scenarios: the historical city centre of Genoa, Italy, and the old part of Barreiro, Portugal, that will be presented in the chapter 8. The application of the proposed car sharing system to Genoa has been described in several papers: Cepolina et al. (2010), Cepolina and Farina (2012a), Cepolina and Farina (2011a), Cepolina and Farina (submitted to TRB).

Genoa is a city of 650,000 inhabitants over 240 squared km, of which only 75 inhabited. It lies mainly on a 30 km long and narrow coastal plain extending through few narrow valleys (along rivers of torrential character) into the western steep slopes (reaching 1200 m high) of the Apennine Mountains. The transport system is strongly influenced by: the complex orography, the overcrowded and congested structure, the presence of an heavy industrial area within the city limits and an extended historical centre (considered to be the widest in Europe) that practically separates the town into two parts. Access to traffic is practically impossible in the historical centre, thereby strongly affecting the town transportation system. The traffic is forced to flow through limited routes across densely populated areas. An image of Genoa and of the PICAV intervention area is provided in figure 7.1.

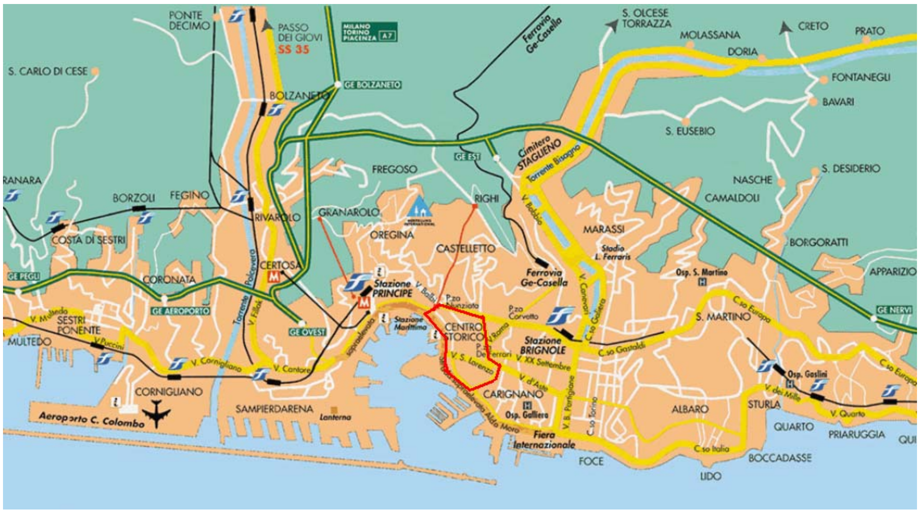


Fig. 7.1. Genoa and the intervention area. The intervention area is circled in red.

The historical city centre of Genoa has an area of about 1.13 km^2 and it is a pedestrian area. This area is quite small but it is much populated. Its density is indeed about 11,000 inhabitants per square kilometre. Genoa historical city centre is the inner part of the city centre itself. It is completely restricted to private cars – apart for the residents – and also public transport cannot operate because of the narrowness and the slope of the roads.

7.1. The scenario under study

7.1.1. Transport modes in the urban area of Genoa.

7.1.1.1. Conventional public transport

The public transport coverage is 785 km of bus lines (over 600 vehicles circulating during daytime/working day), 56 km of urban railway (16 existing stations and 4 in project), 3 km of metro (4.3 under construction), 3 km of funiculars and 10 lifts. A new public transport on sea (particularly addressed to tourists) links Pegli with the Ancient Harbour. An on-demand shuttle service connects Park and Ride zones and the historical centre by electric buses. A total of 150,000,000 passengers/year travel by buses and 4,300,000 passengers/year by trains. The cost of the integrated rail-bus ticket is 1.50€/90 minutes (Cepolina, 2009). A schema of the bus lines around the historical city centre is reported in figure 7.2. Bus lines travel around the historical city centre. The orange dashed line is the underground, whose path is also around the historical city centre. In figure 7.3, a scheme of the path of the underground is reported.

7.1.1.2 Car-sharing and bike-sharing systems.

Genoa is also provided with car-sharing and bike-sharing systems.

As it concerns the *car-sharing* system, the car needs to be booked in advance and the user, at the booking time, has to specify where he is going to pick up the car and when he is going to return it. The user has to return it in the parking lot where he picked it up. The fleet is composed by 360 cars that are kept in 49 terminals throughout the city. The parking lots are 94. Subscribers pay a 180 € annual registration fee, then the cost of the car rental is proportional to the travelled time and the travelled distance. The time cost ranges from 2 to 3,4 €/1hour depending on the car typology; the distance cost ranges from 0,24 to 0,78 €/Km depending on the car typology and the trip length. The system is attractive to customers who make only occasional use of a vehicle, as well as others who would like occasional access to a vehicle of a different type than they use day-to-day. At the end of 2007 the car sharing users were 11300.

As it concerns the *bike sharing* system, 55 bicycles are kept at 6 self-service terminals throughout the city. Individuals registered with the program (subscribers pay an 35 € annual registration fee) identify themselves with their membership card (or a smart card, via cell phone, etc) at any of the hubs to check out a bicycle. The individual is responsible for the bike until it is returned to another hub. The first half hour is free, each additional half-hour costs € 0.50.
The localization of car sharing and bike sharing stations is shown in fig. 7.4.

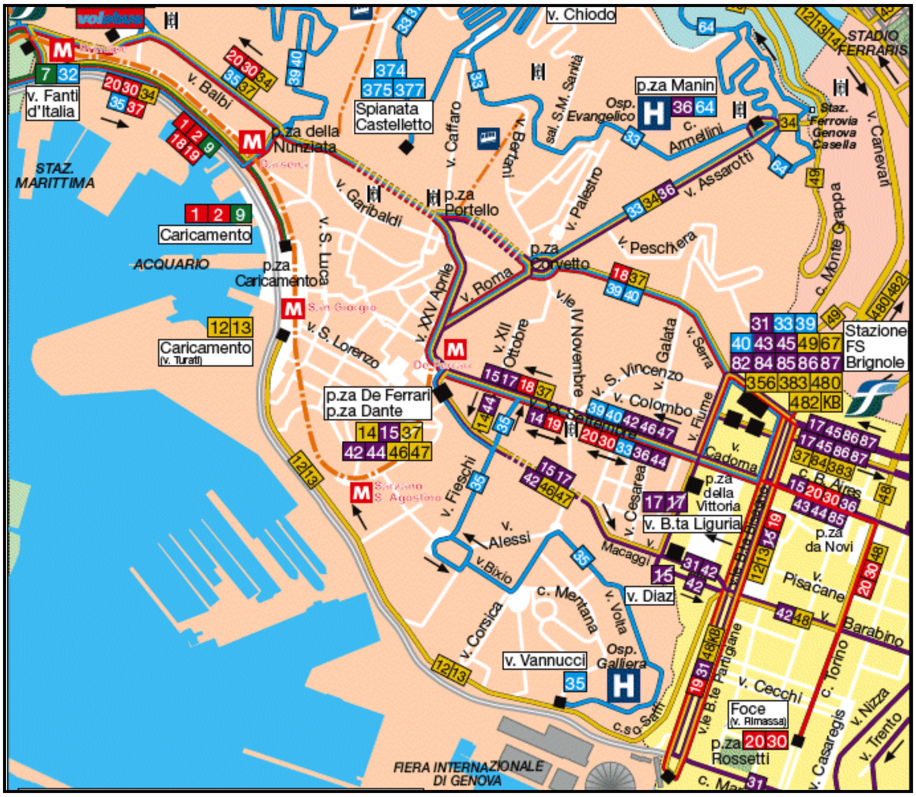


Fig. 7.2. The bus lines around Genoa historical city centre. The underground, reported with a dashed orange line, has been prolonged to the station "Brignole" (Cepolina, 2009).

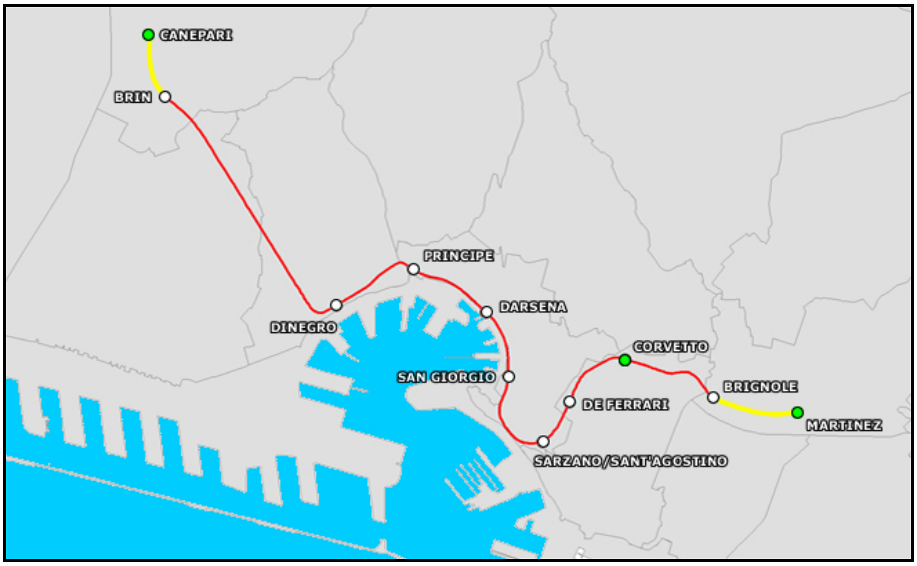


Fig. 7.3. The underground. In red, the underground line currently in exercise. In yellow, the underground line under construction. The stations under construction are drawn in green (www.amtgenova.it).

7.1.1.3. Private transportation

Private transportation consisting of 392,000 cars - 32,000 small trucks - 67,000 motor bikes is characterized by one of the slowest average speed in Italy (12.6 km/h). The main urban roads show fluxes above 30,000 vehicles in the interval 7 a.m. - 8 p.m.

In Genoa people are encouraged to use public transport through the implementation of traffic limitations and urban pricing and the creation of "Park and Ride" (10 car parks / 1000 vehicles). As it concerns urban pricing, it is based either on parking (Blue Area) and on cordon crossing (Road Pricing). A wide part of the city, marked with light blue pattern (Blu Area) in the following figure, is divided in small areas: residents in each zone pay an annual fee and are allowed to park freely in their zone, while the other cars are subject to an ordinary parking tariff. The red line (virtual cordon) in the following figure delimits the part of the city subject to *Road Pricing*. The entrances to this area are provided with electronic gates that allow to control the cars that cross the cordon and enter the area. The access right is not only for the categories of exempted, but is also simply acquired through the payment of a sum. The payment is due every time a non-exempted car enters the restricted area (in the restricted hours). The localization of blue area, ZTL and road pricing in Genoa are shown in fig. 7.5.

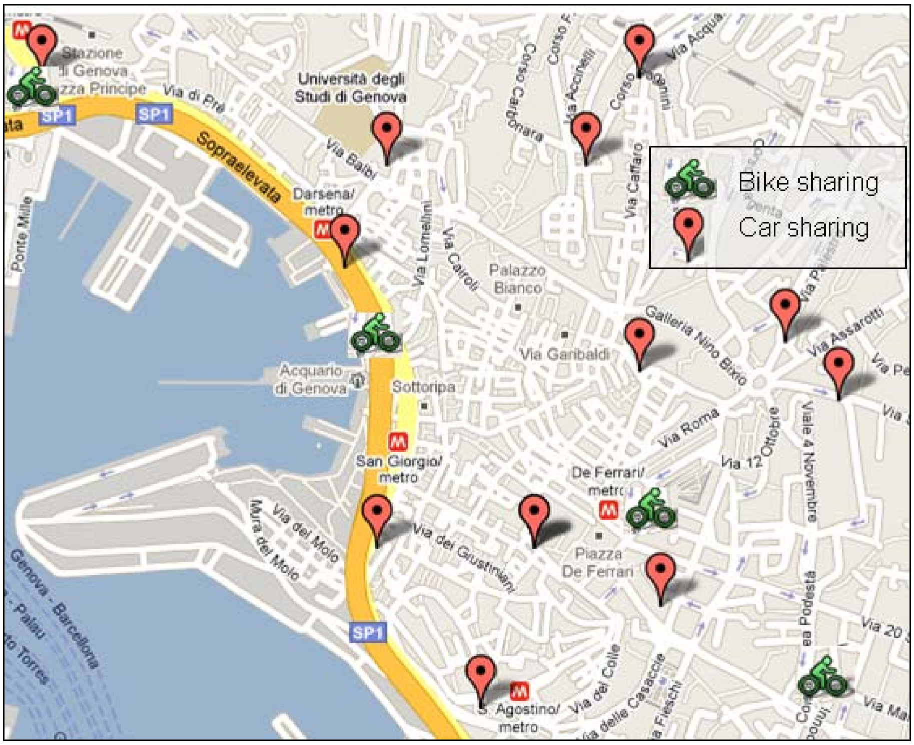


Fig. 7.4. Localization of car sharing and bike sharing systems (Cepolina, 2009).

Traffic limitations: ZTL (i.e. restricted traffic zone). As for the Road Pricing, part of the city (marked with a green pattern in the following figure) is delimitated by a virtual cordon. The entrances to this area are provided with electronic gates that allow to control the cars that cross the cordon and enter the area. Its access is defined by citizen's categories. This strategy aims simply at reducing motorized displacement within a certain area through prohibition. We would like to spend some words also to a recent and innovative schema related to the freight transport in the historical city centre, that begun this year. It is called Mercurio and it is based on Mobility credits. Mercurio is active in the historical city centre and it applys only to the freight transport. Each commercial activity, office and handicraft shop in the historical city centre receives a certain number of credits and can spend them for the freight tranport making different transport choices; every trip has a differnet cost in terms of "mobility credits" depending on the transport mode, origin and estination. The credits are transferred from the commercial activity, office and handicraftsman to the freight traspoters. Once spent all the assigned credits, the commercial activities, offices and handicraftsmen can buy new credits from the public administration.

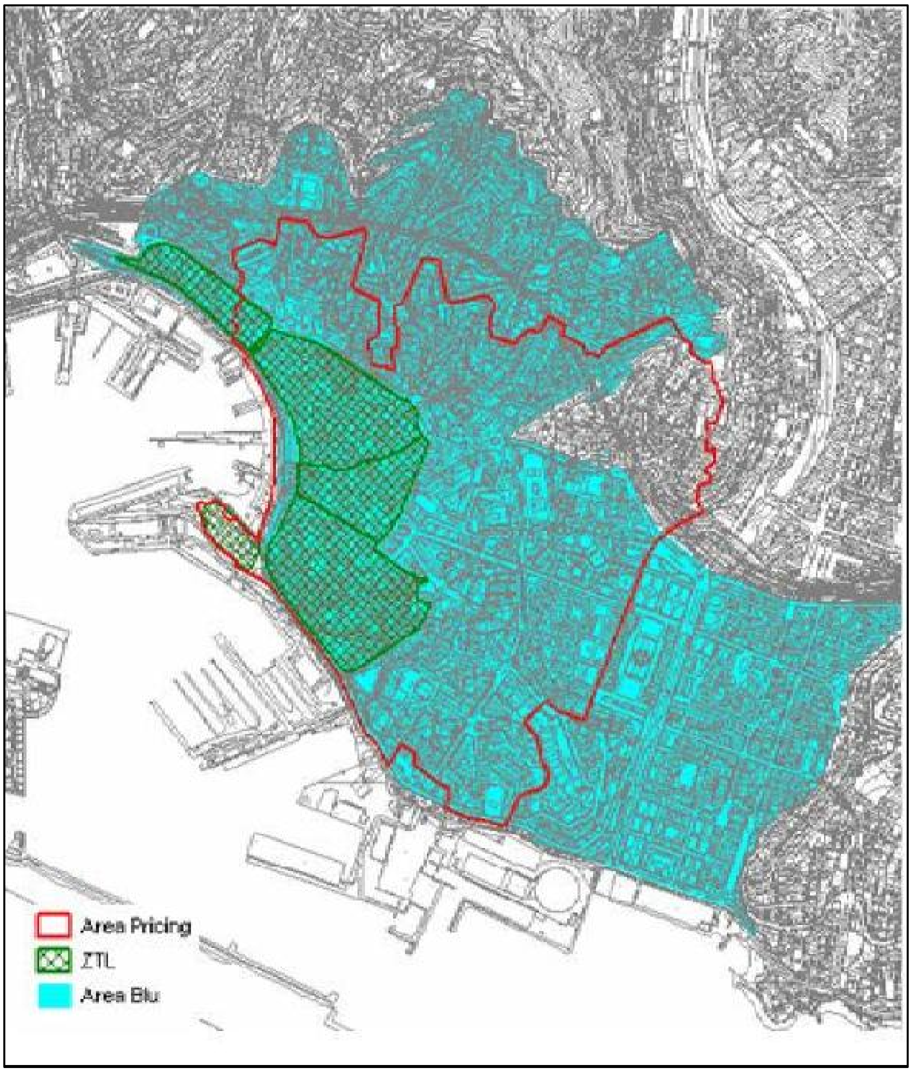


Fig. 7.5. Blue Area, ZTL and Road Pricing in Genoa (Cepolina, 2009).

7.1.2. The historical city centre

The historical city centre of Genoa is the most inner part of the old city and it is organized in several small roads of middle-aged origin, called “carrugi”. The historical city centre is comprised between the hill of Carignano (in the east) and the railway station “Genova Piazza Principe” (in the west), close to the former “Palazzo del Principe” and residence of the admiral Andrea Doria.

The area is part of the municipality 1 Center – East.

Because of the extension of the original nucleus, i.e. 1.13 km^2, it is considered the most extended historical city centre in Europe. In reality, even the historical city centre of Pisa is greater. However, the population density is one of the highest in Europe. In the area live about 20000 inhabitants, subdivided in about 5000 buildings.

7.1.2.1. Tourist attractions.

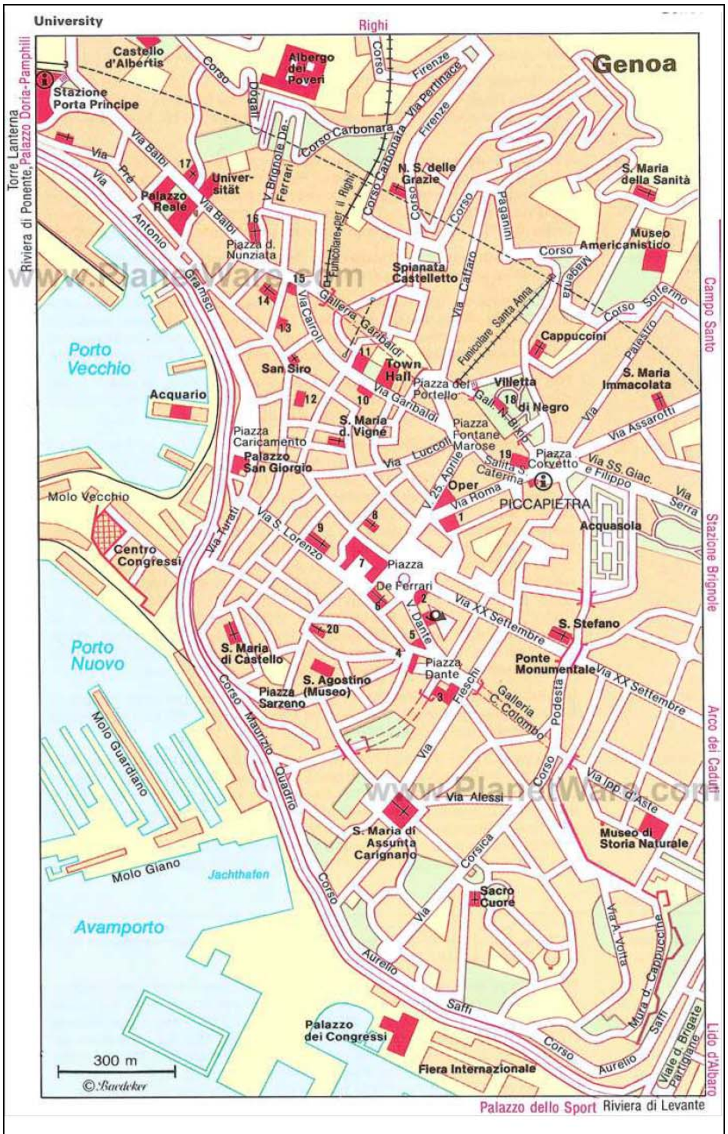
Genoa historical city centre has plenty of tourist attractions. There are various museums, art galleries and exhibitions in town along with civic museums (Comune di Genova museums). There are also two national museums and many private ones. Within the historic centre, there is a unique system of aristocratic residences obliged to host state visits that was included in UNESCO’s World Heritage list. The area inscribed on UNESCO’s list extends from Via Garibaldi and Via Balbi to a section of the historic centre which runs through Via Lomellini, Piazza Fossatello and Via San Luca up to Piazza Banchi, the merchant heart of the old town. Along this stretch are located some of the most significant examples of the Palazzi dei Rolli built in the Modern Age within the medieval fabric. In fig. 7.6, the main attractions in Genoa historical city centre are shown.

7.1.2.2. Commercial activities.

Within the CIVIS project a census of commercial activities in the historical city centre was performed. The commercial activities were classified in 7 categories:

- grocery store;
- bar and restaurant;
- pharmacy;
- clothing store;
- furniture store;
- craft shop / antiques;
- varied.

The location of the commercial activities is shown in figure 7.7.



Some attractions within Map of Genoa Attractions Map:

- (3) Grattacielo
- (5) Casa di Colombo
- (6) Sant'Ambrogio
- (10) Palazzo Rosso
- (11) Palazzo Bianco
- (12) Palazzo Spinola
- (19) Prefecture
- (20) San Donato
- (14) San Filippo Neri
- (17) San Carlo

Fig. 7.6. Attractions in Genoa historical city centre.

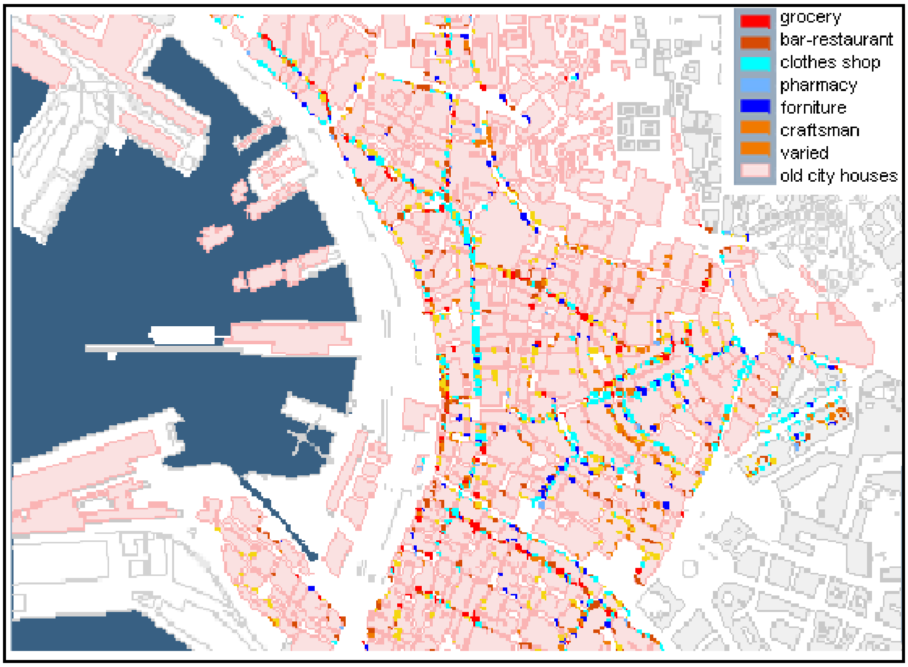


Fig. 7.7. Location of the commercial activities (Cepolina, 2009).

7.1.3. The representation of the scenario under study

We consider as simulation time period the PICAV service during a reference working day: the service starts at 8 a.m. and ends at midnight.

The localizations of bus stops and underground stations have been identified, as well as the localization of hotels, museum, offices, schools and commercial activities (food shops, clothes shops, handicraft shops and other shops). The roads where shops, museums and other attractions are mostly present have been determined. The PICAV vehicle is designed to operate in pedestrian-only environment. Some of roads in the study area have high pedestrian flows and their dimension is extremely reduced. Not all the roads can be travelled by a PICAV: they must be wider than 2 metres.

Each road has been divided into 50 m long sections and therefore 120 units resulted. The roads of the network and the localization of stations is shown in the figure 7.8.

Stations are located on the border of the pedestrian area or inside it, in the proximity of:

1. Interchange points:

  - public transport stops: bus and underground;
  - parking spaces reserved to disable people;
  - parking areas for private vehicles.

2. Attractors where activities that require stops longer than one hour take place. The considered attractors are:
  - hotels;
  - museums / exhibitions;
  - offices;
  - universities / schools;
  - recreational centers (cinema, theatre…).

The localization of the stations has been determined taking into account:
- the localization of intermodal interchange points,
- the availability of space.

Nine stations have been designed: 2 internal to the area and 7 on the area border. In the third car sharing system we designed 7 stations, i.e. the 7 ones on the border of the area, which are placed in correspondence of the main intermodal interchange points.

The location of the stations is shown in figure 7.8:
- the station 1 is located near the underground stop "Darsena";
- the station 2 is located in Piazza Fontane Marose, where several bus lines stop;
- the station 3 is located in Piazza de Ferrari, which is one of the most important interchange points with conventional public transport. An underground station and several bus lines stop there;
- the station 4 is located in Piazza Dante, again in correspondence of several bus stops;
- the station 5 is located in piazza Ravecca, besides an underground station;
- the station 6 is located in piazza Caricamento, another important interchange point with bus lines and with the underground;
- the station 7 is located in Corso Saffi, several bus lines stop there;
- the station 8 is located between via Garibaldi and via Cairoli, and it is inside the pedestrian area, close to several attractors (see figure 7.8);
- the station 9 is in piazza dei Giustiniani, in a baricentric area to serve both the southern part of the historical city centre and the San Lorenzo area.

A possible location for stations 1 and 8 is shown in figure 7.9.

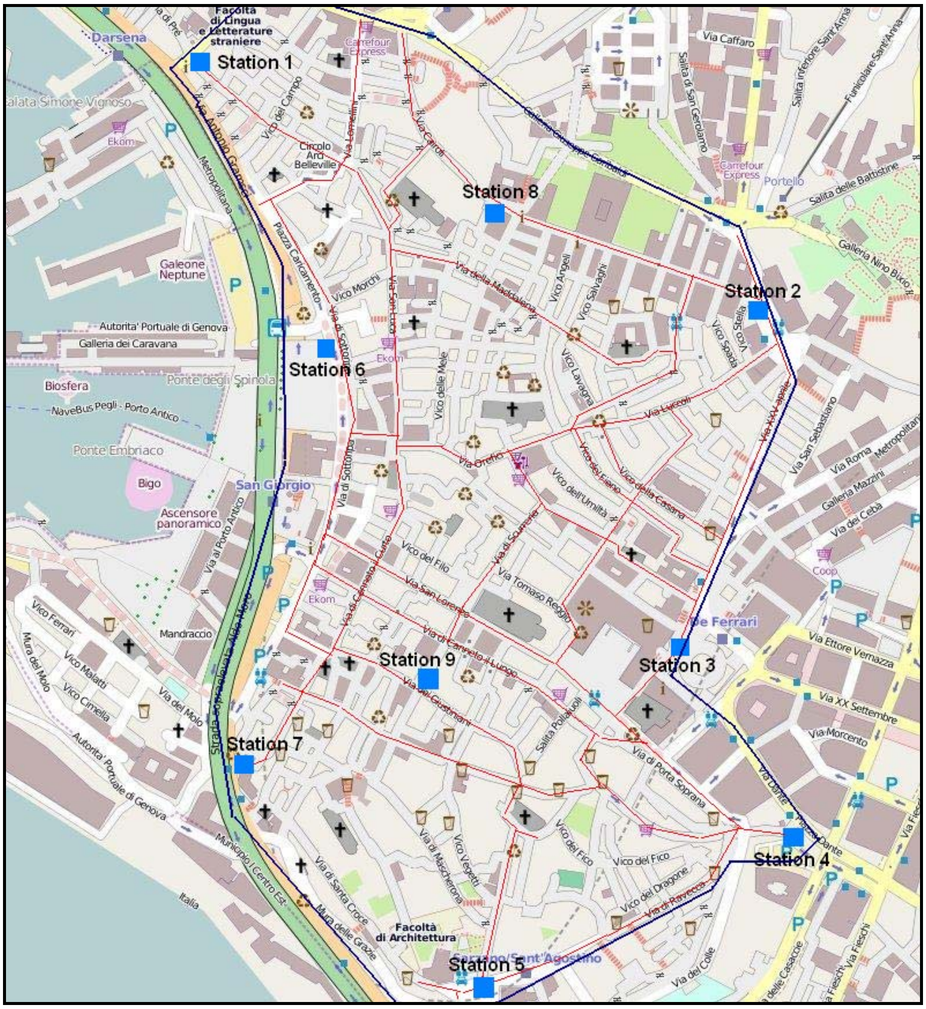


Fig. 7.8. Map of the intervention area. In red, the roads in which PICAV system can operate are underlined. The blue squares are the PICAV stations. Stations 8 and 9 do not exist in the third management strategies. The blue line circles the intervention area.

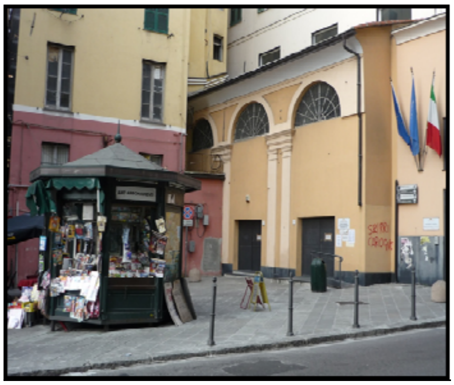
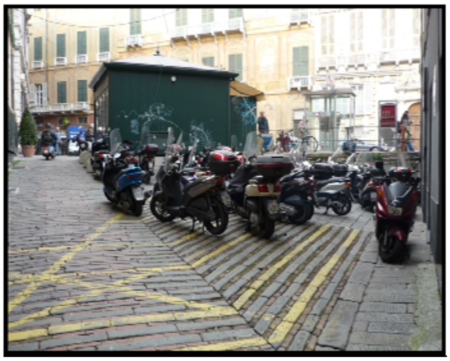



Figure 7.9. Possible sites for the stations 1 and 8.

7.2. The transport demand

7.2.1. The total transport demand

We need to know the transport demand in the historical city centre and we would like to characterize the trips carried out in the historical city centre.

As it concerns the transport demand, we'd like to know the overall number of trips by foot (the only transport mean available in the area given the width of the streets) carried out in a given time interval in the historical city centre. From the actual transport demand knowledge we will generate the PICAV transport demand (i.e. the overall number of PICAV users in the historical city centre of Genoa) in an hypothetical scenario where the means of transport available will be two: by foot or by PICAV. The PICAV transport demand is therefore minor or equal to the actual transport demand. Maybe an additional demand for the PICAV transport system will rise from people that actually cannot visit by foot the historical city centre because of mobility impairments or other impedances to covering long distances (such as high shopping loads).

It is reasonable to assume that there is equilibrium in the area, meaning that the number of people entering the historical city centre doesn't differ too much from the number of people exiting the area.

It is possible to arrive on the historical city centre border zone by public transport, by individual transport, by taxi and by car/bike sharing.

Given the urban pricing schema, the car/bike-sharing rent fees and the taxi fees, the means of transport mainly used for reaching the historical city centre are bus, metro and lifts as it concerns the public transport and motorbike as it concerns the individual transport. However, the usage of private transport is reduced for reaching the historical city centre.

This doesn't apply for disabled people as they have special permissions and can park their private cars on the historical city centre border for free; therefore the mean of transport mainly used by them is the private car.

We consider only people arriving on the area border by public transport. This seems reasonable given the limited use of private modes in the inner city centre of Genoa. We assume that 1% of the people that currently enter the historical city centre by foot will use the PICAV transport system.

The main source of information about the arrivals at the historical city centre border is a study by AMT (www.amt.genova.it/). We extrapolated the data from AMT, related to the time slice between the 6 and 9 a.m, to the whole day. We will describe in detail the temporal distribution of the PICAV demand afterwards in this section.

7.2.2. Interviews. The share of the transport demand in trip typologies

In chapter 3, it was stated that four different trip typologies are considered in the proposed transport system:

A. type A trip: trip chain having both origin and destination external to the historical city centre;
B. type B trip: single trip having origin external to the historical city centre and destination internal, or vice versa;
D. type D trip: single trip having both origin and destination internal to the historical city centre;
E. type E trip: trip chain having both origin and destination internal to the historical city centre.

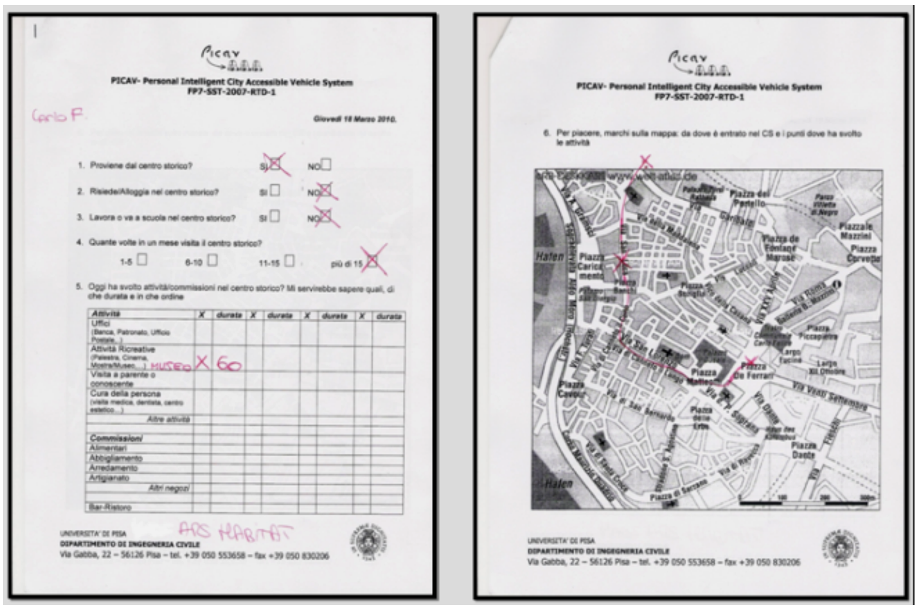
In Genoa scenario only type A and type B trips take place. In order to assess the share of the transport demand a campaign of interviews has been performed.

The estimation of the transport demand by interview is possible by two techniques: *revealed preferences* techniques, traditionally utilized, which are relative to the actual users travel behaviour in a real context, and *stated preferences* techniques based on statements made by interviewees about their preferences in different choice contexts, real, hypothetical or experimental. Since we would like to know the actual transport demand, we choose the revealed preferences technique.

Since one of the main transport means used for reaching the historical city centre is public transport and we know (from data given by AMT) the number of passengers that get on at the bus stops close to the border of the historical city centre, a first set of interviews has been carried out, in order to determine:

- the percentage of demand directed to the historical city centre
- the trips' characteristics in terms of activity travel pattern.

Two days of interviews have been performed in order to study the characteristics of the trips performed in the intervention area. The first day of interviews was carried out on the 3rd of December between 4 and 5:30 p.m.; the second one was performed on the 18th of March between 11:30 and 13:30 and between 16:30 and 18:15. We interviewed the people while they were waiting for the bus or the train at five bus stops and three underground stations close to the historical city center border. The designed questionnaire is reported in fig. 7.10.


PICAV- Personal Intelligent City Accessible Vehicle System
FP7-SST-2007-RTD-1

Carlo F.

Giovedì 18 Marzo 2010.

1. Proviene dal centro storico? [✓ SI] NO
2. Risiede/Alloggia nel centro storico? SI [✓ NO]
3. Lavora o va a scuola nel centro storico? SI [✓ NO]
4. Quante volte in un mese visita il centro storico?
   1-5 6-10 11-15 [✓ più di 15]
5. Oggi ha svolto attività/commissioni nel centro storico? Mi servirebbe sapere quali, di che durata e in che ordine

Attività	X	durata	X	durata	X	durata	X	durata
Uffici (Banca, Patronato, Ufficio Postale,..)								
Attività Ricreative (Palestra, Cinema, Mostra/Museo,...) MUSEO	X	60						
Visita a parente o conoscente								
Cura della persona (visita medica, dentista, centro estetico,..)								
Altre attività								
Commissioni								
Alimentari								
Abbigliamento								
Arredamento								
Artigianato								
Altri negozi								
Bar-Ristoro								

ARS HABITAT

UNIVERSITA' DI PISA
DIPARTIMENTO DI INGEGNERIA CIVILE
Via Gabba, 22 – 56126 Pisa – tel. +39 050 553658 – fax +39 050 830206

PICAV- Personal Intelligent City Accessible Vehicle System
FP7-SST-2007-RTD-1

6. Per piacere, marchi sulla mappa: da dove è entrato nel CS e i punti dove ha svolto le attività

Figure 7.10. An example of the questionnaire proposed (Cepolina et al., 2010).

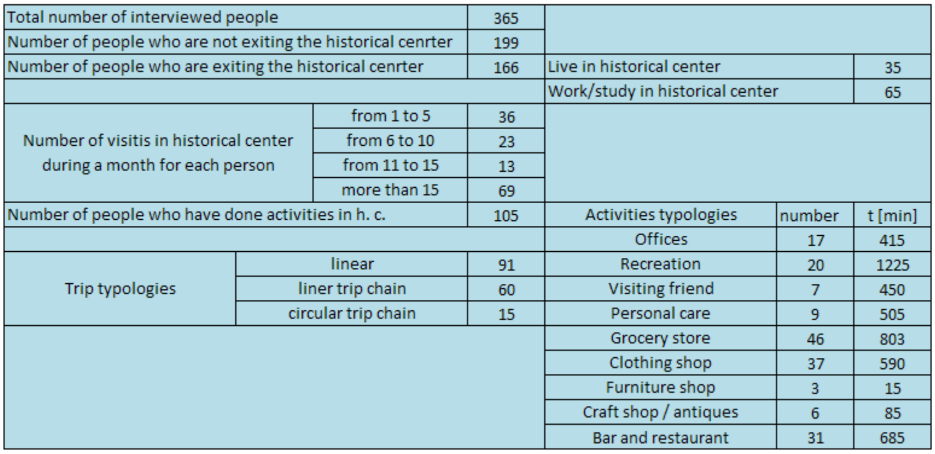
Firstly it was asked to each person if he/she was exiting the historical city center, then in case of affirmative answer to this question, a description of the activity–travel patterns carried out in the historical city center was required. The activity–travel pattern description concerns the sequence of activities carried out, the time spent for each activity, the typology of the activity and the path between two successive activities.

The number of the interviewed people is equal to 365. The results show that about 45.5% (166/365) of the interviewed people were exiting the historical city center and among them the 39% (65/166) work or study in the historical city center.

Among the 166 people coming from the historical city center, we have:

- 55% (91/166) have performed a trip without intermediate stops;
- 36% (60/166) have performed a linear trip chain;
- 9% (15/166) have performed a circular trip chain.

The summary of the collected data is shown in fig. 7.11.



Total number of interviewed people		365			
Number of people who are not exiting the historical cenrter		199			
Number of people who are exiting the historical cenrter		166	Live in historical center		35
			Work/study in historical center		65
Number of visitis in historical center during a month for each person	from 1 to 5	36			
	from 6 to 10	23			
	from 11 to 15	13			
	more than 15	69			
Number of people who have done activities in h. c.		105	Activities typologies	number	t [min]
			Offices	17	415
Trip typologies	linear	91	Recreation	20	1225
	liner trip chain	60	Visiting friend	7	450
	circular trip chain	15	Personal care	9	505
			Grocery store	46	803
			Clothing shop	37	590
			Furniture shop	3	15
			Craft shop / antiques	6	85
			Bar and restaurant	31	685

Figure 7.11. Summary of the data collected in the questionnaires (Cepolina et al., 2010)

From the results of the interviews, resulted that, among the total number of passengers who arrive on the border of the historical city center, only the 45% actually enters into the intervention area.

From the collected data, we obtained the following information about the typologies of trips performed in the historical city centre:

- 45% are trip chains; the average number of activities in the activity travel pattern is 1.7; the average duration of the activity travel pattern is 40 minutes;
- 55% of potential PICAV users perform direct trip; the average duration of the trip is 2.5 minutes;
- From the collected data, the duration of a multitask trip resulted to be about 5 times the related duration of the direct trip.

7.2.3. Temporal distribution of the transport demand

In order to determine the temporal distribution of the users during the simulation time, several videos have been recorded on the study area. In particular, some videos have been recorded of pedestrian densities in different time periods, in via San Luca, via Canneto il Curto, via Balbi and via Luccoli, four of the most important roads in the intervention area. The results have shown that the afternoon demand is about 1.45 times the morning demand, and that the afternoon period begins at about 4 p.m. According with the data collected in the field, we registered an average pedestrian density of 0.15 ped/m^2 in the off-peak period and 0.35 ped/m^2 in the peak period in Via San Luca. For the current simulation we assumed these densities are the same in all the roads in the area. The most crowded road in the area is Via San Luca and it is in favour of security to assume all the densities equal to the one in via San Luca.

According to the results of the video, to data taken from AMT website and to direct observations, two periods of demand have been assumed:

- an off-peak period, which ranges from 8 a.m. to 4 p.m;
- a peak period, from 4 p.m. to 8 p.m.
- from 8 p.m. to midnight there is no PICAV transport demand.

In fact we hypothesize for the PICAV transport system a leisure usage: visit to museums, commercial activities, etc.

The overall demand in the reference day is 1644 users.

7.2.4. Spatial distribution of the transport demand

The share of the demand having origin and destination on the border of the intervention area among stations has been determined from AMT Genoa website. The number and the frequency of public transport lines which stop in proximity of each station has been recorded. According to this, a weighting factor has been determined for each station, and the total demand has been multiplied by this factor for what regards each station. In this way the number of users entering the historical city centre from a given station in one hour has been determined. The percentages of arrivals at the stations of the area are shown in the figure 7.12.

In the first relocation strategy, users are flexible and have a choice set. In the other two, it has been assumed that the number of users exiting the historical city centre from a given station in one hour is the same as the number of users entering the historical city centre from the same station.

The demand having origin or destination inside the intervention area, has been assessed according to the attractivity of the stations.

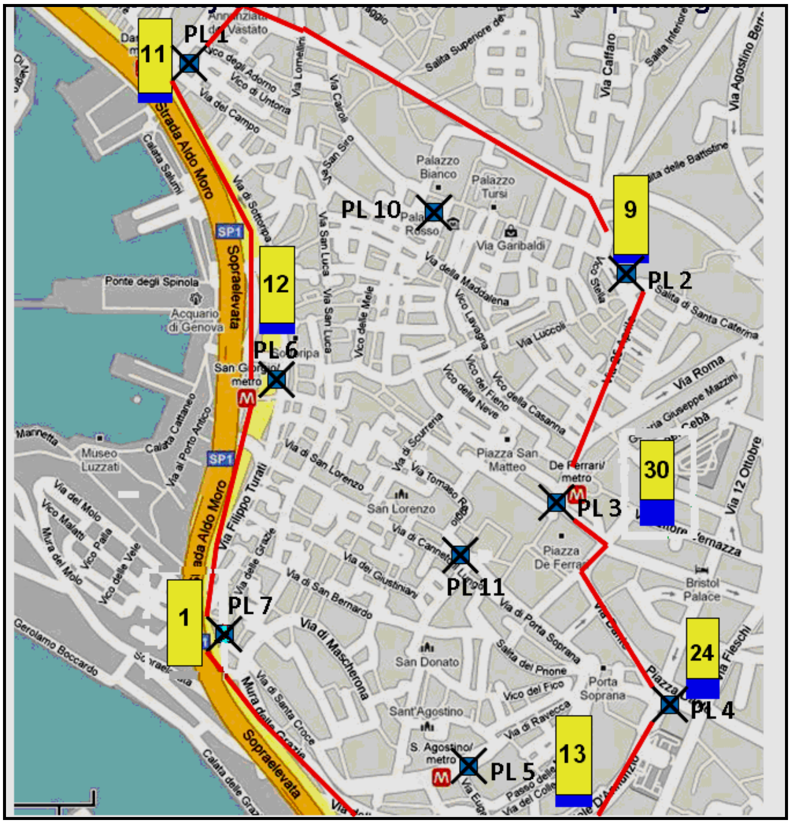


Fig. 7.12. Percentages of arrivals at the parking lots from outside the intervention area; redistribution of the demand exiting from the intervention area to the stations on the border (Bonfanti, 2010).

The share of the demand origin or destination inside the intervention area among stations, has been assessed according to the attractivity of the stations. The percentages of demand distributed to the stations is shown in figure 7.13. Each station is characterized by an attractivity: it is a function of the number and typology of attractors within the station’s influence area: e.g. shops, museums, offices, etc. The influence area is the zone for which the station is convenient, i.e. the pedestrian distance from the station to the attractor is acceptable. We assumed it of circular shape and with a radius of 300 m, as the maximum acceptable walking distance is about 400m (Smith and Butcher, 2008). The transport demand having origin and destination inside the intervention area has been assessed according to the station’s attractivity. The probability that a PICAV user, who performs a single trip, ends it in a given station is proportional to the station’s attractivity.

The attractivity of each station has been assessed as a function of the number and typology of the activities within its influence area. A weighting factor has been associated to each activity, according to the typology of the activity: a different factor has been associated for example to a grocery shop and to a clothes shop. The location of the attractors and the stations' influence areas are shown in figure 7.14.

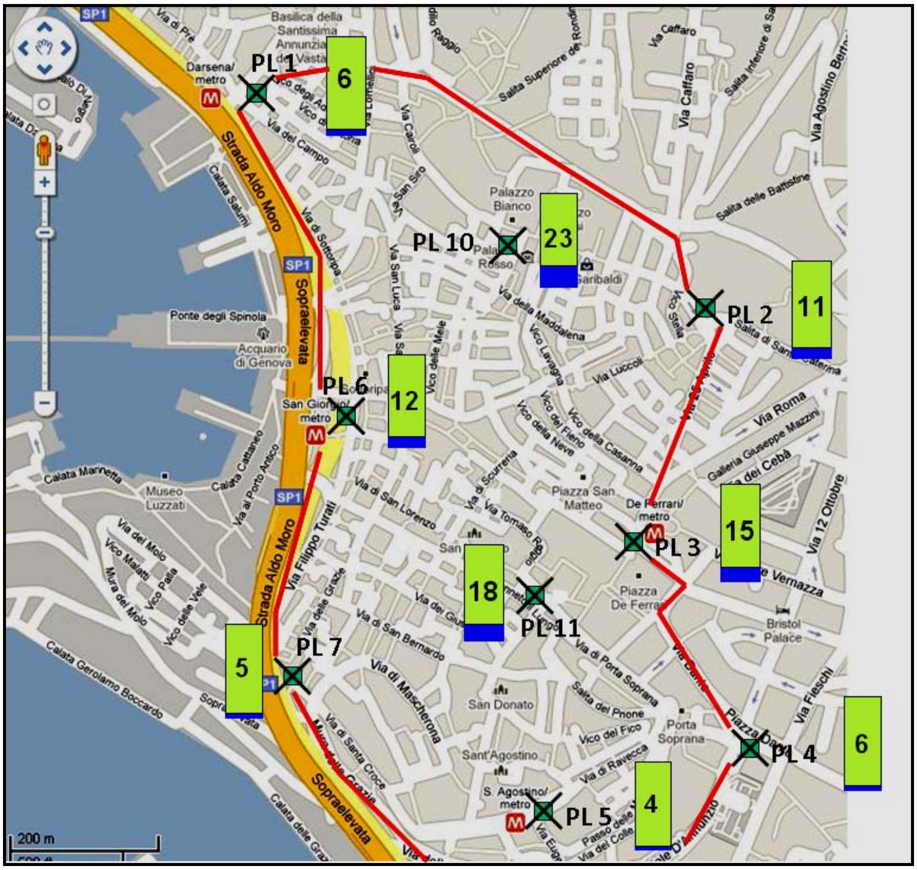


Fig. 7.13. The parking lots attractivity: redistribution of the demand exiting from the border to inside the intervention area (Bonfanti, 2010).

In reality for what regards the third management strategy:

- the demand having origin or destination on the border of the intervention area, has origin and destination at the stations;
- the demand having origin or destination inside the intervention area, has origin or destination along the roads.

As stated in chapter 3, the streets inside the pedestrian area have been divided into sections and at the center of gravity of each section a node, called unit, is located. If the user trips have origin and destination only at stations, the units are the places where a PICAV user can stop for a short time (less than one hour) to make some shopping or activity.

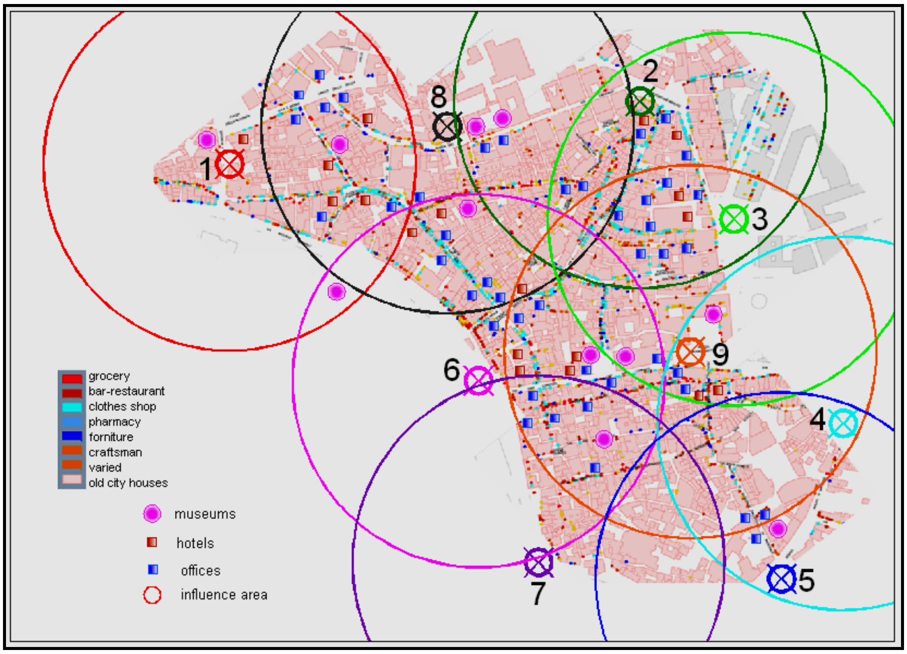


Fig. 7.14. The stations' influence area and the location of all the attractors. (Bonfanti, 2010).

If the vehicles can be returned also along the roads of the intervention area, which is the third management strategy exposed in chapter 3, then the user trips can have origin and destination also at the units. We assign an attractiveness to each unit, which is a function of the number and the typology of the attractors presents in the road section. We considered the following attractors:

- grocery stores;
- bars and restaurants;
- pharmacies;
- clothing stores;
- furniture stores;
- craft shops / antiques;
- varied.

The location of these attractors is shown in figure 7.15.

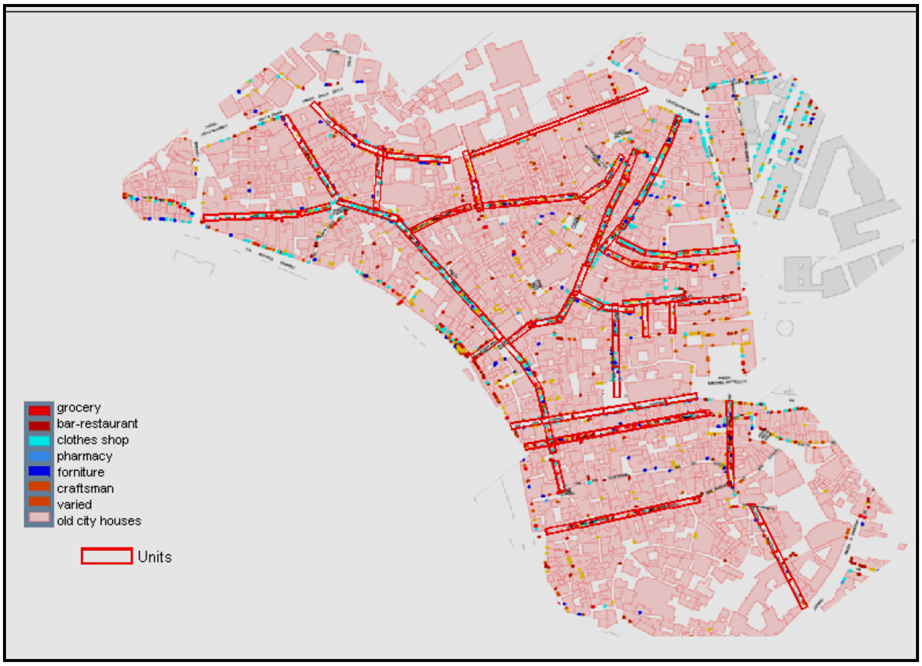


Fig. 7.15. The position of units in the intervention area and the location of attractors for short term stops (Bonfanti, 2010).

7.2.5. The assessment of the choice set of flexible users

The metropolitan area of Genoa has been divided in three macro areas: north, east and west. For each zone has been determined:

- the percentage of residents;
- the number of lines of public transport which serve each macro area.

The first information is used to assign a macro area to each user.

The second information is used to assign a choice set to each macro area. This choice set is composed by all the stops where a public transport line, suitable to reach the macro area, stop.

The choice set for each macro area is determined according to the path that public transport lines follow. For example, the choice set of users having origin or destination in the western part of Genoa has been determined taking into account the path of the lines which connect the city centre with this part of the city.

In particular, the choice set for the western part of Genoa involves the following stations: 1 (Darsena), 2 (Fontane Marose), 3 (De Ferrari), 4 (Dante) and 6 (Caricamento). The choice set for the northern part of Genoa involves the stations: 1, 2, 3, 4, 5 (Sant'Agostino), 6, 7 (Corso Saffi). The choice set for the eastern part of Genoa involves the stations: 1, 2, 3, 4.

An image of the intervention area and the related choice sets is shown in figure 7.16

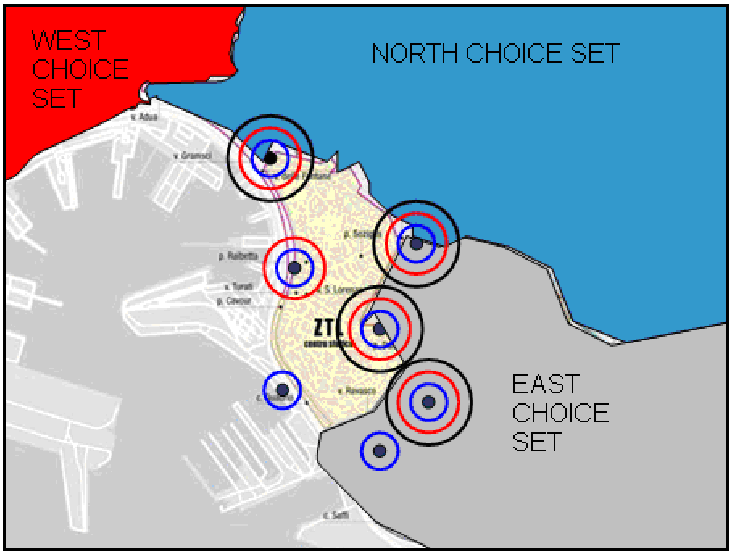


Figure 7.16. The users' choice sets. The red circles refer to the stations belonging to the west choice set; the grey circles to the east choice set; the blue circles to the north choice set (Bonfanti, 2010).

7.2.6. Demand OD matrixes

The transport demand, determined as explained the previous sections, is expressed in the form of OD matrixes. An OD matrix reports the number of trips having a given origin and a given destination, which occur in a time period. In our study, OD matrixes refer to an hour. Each OD matrix has reported: in rows the origin stations or units, in columns the destination stations or units. For example, the number written in row 1 and column 3, is the number of users who have as origin the station 1, and destination the station 3.

Several demand OD matrixes have been developed, in particular:

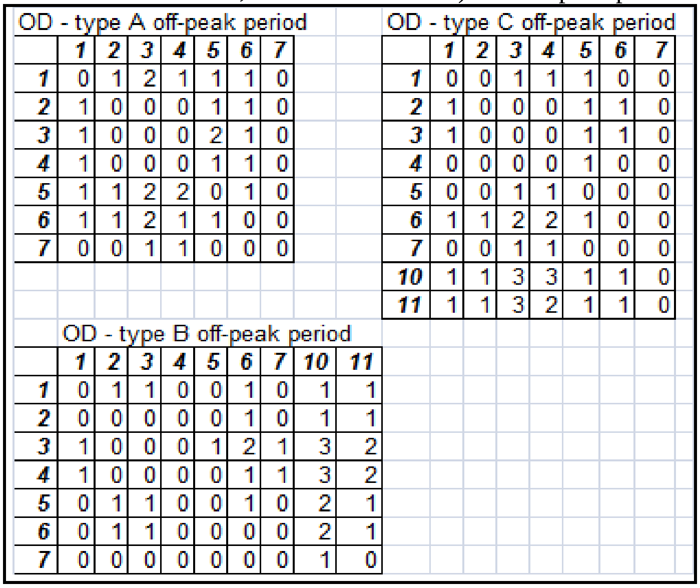
- an OD matrix for each trip typology, i.e. direct trip, and trip chain;
- an OD matrix for each period of the day;
- an OD matrix for users' choice sets, i.e. which has in rows the origin station, and in columns the choice sets.

In table 7.1, the OD matrixes for the second management strategy (i.e. origin and destination at stations, automated vehicles), are reported.

7.2.7. Arrival time instant

The OD matrix provides for each user the origin, the destination, the choice set if the user is flexible and the period of arrival. The OD matrixes of this thesis are on hourly basis, i.e. they provide the trips having a given origin and destination that take place in one hour. The exact time instant within the hour when an user arrives on the border of the historical city centre is determined from a random extraction, assuming a uniform distribution of arrivals in the hour.

Table 7.1. Sample OD matrixes. They refer to the second management strategy (i.e. origin and destination at stations, automated vehicles) and off-peak period.



OD - type A off-peak period

	1	2	3	4	5	6	7
1	0	1	2	1	1	1	0
2	1	0	0	0	1	1	0
3	1	0	0	0	2	1	0
4	1	0	0	0	1	1	0
5	1	1	2	2	0	1	0
6	1	1	2	1	1	0	0
7	0	0	1	1	0	0	0

OD - type C off-peak period

	1	2	3	4	5	6	7
1	0	0	1	1	1	0	0
2	1	0	0	0	1	1	0
3	1	0	0	0	1	1	0
4	0	0	0	0	1	0	0
5	0	0	1	1	0	0	0
6	1	1	2	2	1	0	0
7	0	0	1	1	0	0	0
10	1	1	3	3	1	1	0
11	1	1	3	2	1	1	0

OD - type B off-peak period

	1	2	3	4	5	6	7	10	11
1	0	1	1	0	0	1	0	1	1
2	0	0	0	0	0	1	0	1	1
3	1	0	0	0	1	2	1	3	2
4	1	0	0	0	0	1	1	3	2
5	0	1	1	0	0	1	0	2	1
6	0	1	1	0	0	0	0	2	1
7	0	0	0	0	0	0	0	1	0

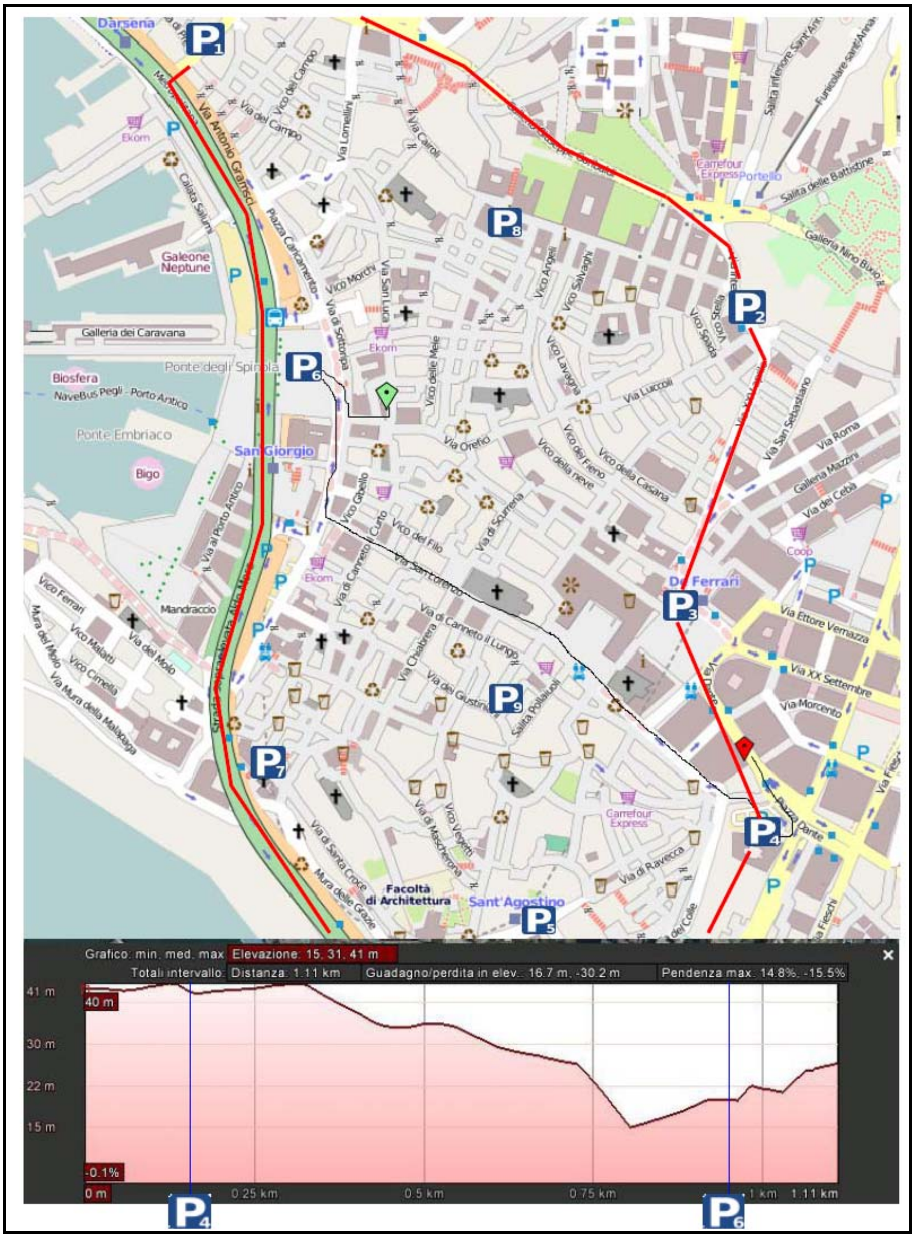


Grafico: min, med, max Elevazione: 15, 31, 41 m

Totali intervallo: Distanza: 1.11 km Guadagno/perdita in elev.: 16.7 m, -30.2 m Pendenza max: 14.8%, -15.5%

Fig. 7.17. The intervention area and the parking lot positions (above). Longitudinal profile of the path between parking lots 4 and 6 (below)

7.3. The network

The network is composed of:

- the stations;
- the units, which represent the above mentioned 50 metres – long road sections;
- the characteristics of the path between each couple of stations and units.

Between each pair of stations or units only one path has been taken into consideration. It refers to the most used path, i.e. the path where the leisure attractions, e.g. shops, are mostly concentrated.

The characteristics of the paths have been assessed from Google Maps and Google Earth. The distance between each couple of points and/or stations has been determined in Google Maps according to the figure 7.17.

The slope of each path has been determined through Google Earth by creating the longitudinal profile for each piece of road. More in detail, the path has been selected in Google Maps and imported into Google Earth where the elevation profile of the path can be viewed. As shown in figure 7.17, Google Earth writes besides the profile the average upslope and downslope of the selected path. Moreover, there is a vertical bar which can be moved along the elevation profile and Google Earth provides the punctual slope registered in the position of the bar.

All these data have been collected and stored. A summary of these data is provided in table 7.2.

The simulator receives in input: for each couple of points or stations, the distance and the average slope. Because it makes no difference, for the battery discharging process, to consider the slope of each upslope or flat part of the path and the average upslope, the average upslope has been considered. Moreover, the capability of the battery to recharge in the downslope pieces of the path has been neglected, and this choice is in favour of security, therefore all downslope parts of the path have been assumed as flat.

Table 7.2: Lengths and gradients of some routes between stations

Origin	Dest.	Total distance [m]	Up [m]	Down [m]	Flat [m]	Up slope [%]	Down slope [%]
1	2	1060.00	805.60	247.50	0.00	3.87	-2.88
1	3	976.00	717.80	258.00	0.00	3.97	-3.67
1	4	1200.00	911.00	323.00	0.00	3.99	-2.94

1	5	1400.00	682.00	719.00	0.00	4.18	-3.51
1	6	604.00	319.90	285.80	0.00	5.27	-7.17
1	7	1100.00	589.50	492.90	0.00	3.89	-5.03
1	8	489.00	408.70	77.40	0.00	6.63	-3.90
1	9	1060.00	762.30	290.90	0.00	3.71	-2.67
2	3	316.00	28.96	287.00	0.00	9.17	-2.06
2	4	846.00	611.90	233.71	0.00	2.61	-3.89
2	5	1180.00	660.70	517.00	0.00	2.52	-3.33
2	6	732.00	264.30	465.75	0.00	2.82	-4.99
2	7	1100.00	144.80	451.00	308.00	15.07	-5.75
2	8	489.00	310.50	177.76	0.00	7.35	-9.09
2	9	747.00	341.80	405.03	0.00	4.68	-4.47
3	4	414.00	255.51	158.30	0.00	6.25	-4.44
3	5	516.00	264.50	250.10	0.00	13.67	-13.4
3	6	569.00	146.90	417.10	0.00	2.97	-4.76
3	7	829.00	437.29	386.00	0.00	4.28	-6.39
3	8	563.00	399.10	163.60	0.00	6.89	-9.68
3	9	326.00	174.01	152.30	0.00	4.79	-6.28
4	5	481.00	309.06	171.80	0.00	5.68	-11.4
4	6	848.00	219.80	476.00	149.00	2.22	-6.81
4	7	1080.00	558.76	518.00	0.00	2.87	-8.28
4	8	866.00	435.80	429.96	0.00	3.48	-4.37
4	9	379.00	248.50	130.17	0.00	3.98	-11.6
5	6	834.00	231.90	603.20	0.00	4.10	-5.30
5	7	804.00	269.20	531.94	0.00	5.22	-7.11
5	8	1050.00	440.50	606.50	0.00	4.29	-3.72
5	9	372.00	210.60	160.26	0.00	6.52	-9.55

7.4. Calibration of the micro simulator’s parameters, determination of the remaining inputs

As stated in the chapter 6, the parameters of the simulator to calibrate are the ones related to the battery model and to the model of interaction vehicles – pedestrians, plus the low critical and low buffer threshold values.

The remaining simulator inputs are: the fleet dimension and the minimum battery charge level.

In this section, the determination of the fleet dimension, of the minimum battery charge level and the calibration of the low critical and buffer thresholds are reported.

The minimum charge level has been assumed equal to 10% for the first two management strategies and 12% for the third. This value has resulted from analyzing the most consuming trip and trip chains performed by users. Several runs of the micro simulator have been performed in order to check the correctness of data. In the third management strategy, the value is greater as vehicles cannot be recharged when they are idle along the roads and therefore they need to reach the nearest station to recharge.

As it concerns the fleet dimension of the three car sharing systems, it has been obtained through optimization. The best resulting fleet dimensions are:

- 91 PICAV vehicles for the first car sharing system,
- 81 PICAV vehicles for the second car sharing system,
- 77 PICAV vehicles for the third car sharing system.

In the second and third management strategies, the fleet has been assumed equally distributed among the parking lots at the beginning of the time period, as the different demand in the various parking lots is compensated through the automatic relocation. In the first management scheme, the fleet instead has been assumed distributed among the stations, at the beginning of the simulation period, according to the optimization results, in the way shown in table 7.3.

Table 7.3. Number of vehicles in each parking lot at the beginning of the simulated day, in the first car sharing system.

parking lot n°	1	2	3	4	5	6	7	8	9
n° veh available	10	11	12	12	13	9	8	8	8

The low critical and low buffer thresholds have been determined again through optimization. The resulting values are equal to respectively 1 and 5 vehicles for each parking lot in the second management strategy. In the third management strategy they have been taken equal to 2 and 5. The chosen relocation technique is the shortest time technique.

The high critical and high buffer thresholds depend on the stations' capacity and are equal to 15 for what regards the high critical thresholds and 13 for what regards the high buffer thresholds. The stations' capacity has been taken equal to 16 for all stations.

7.5. Output data

7.5.1. Performances of the proposed transport system

The performances of the proposed car sharing system for the case study of Genoa have been assessed through simulation. The outputs of the simulation will be shown herewith.

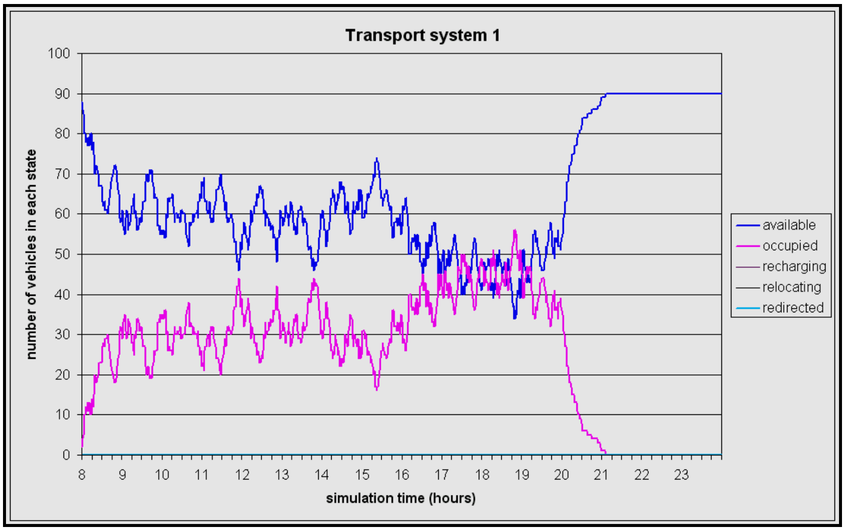


Fig. 7.18. Number of PICAVs in each state against time in the first management strategy (flexible users).

In figure 7.18- 7.19 and 7.20, the number of PICAVs in each state is plotted against time; the states taken into account are: available, occupied by users, required but not available because in charge, relocating, redirected because there is not free space in the destination parking lot (FPT occurrences). Relocation is considered also when the vehicle moves towards the user's position in the third relocation scheme. Redirection is when the vehicle reaches an unit with a battery level inferior to the minimum, therefore it cannot be occupied and moves the closest station to recharge. Time is expressed in hours, starting from 8 a.m. to midnight, when the last user returns the PICAV unit. Figure 7.18 is related to the first management strategy, figure 7.19 relates to the second management strategy, figure 7.20 is referred to the third management strategy.

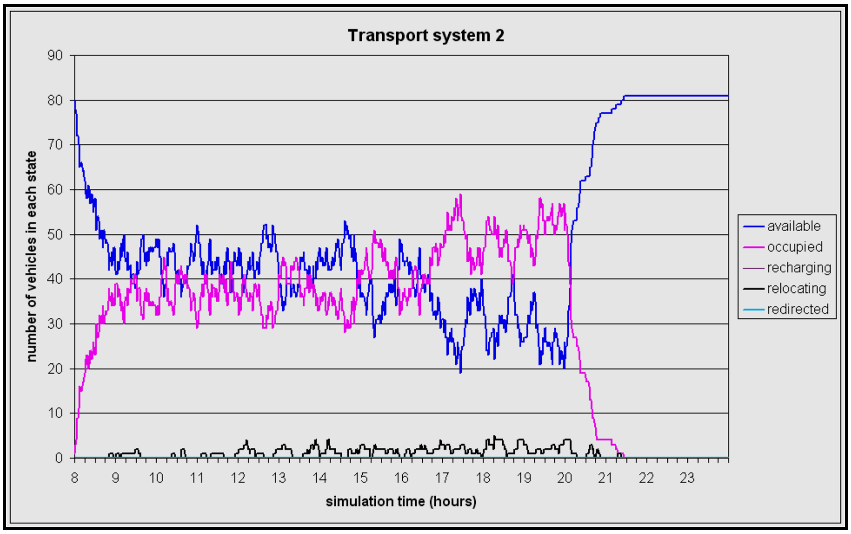


Fig. 7.19. Number of PICAVs in each state against time in the second management strategy (automated vehicles).

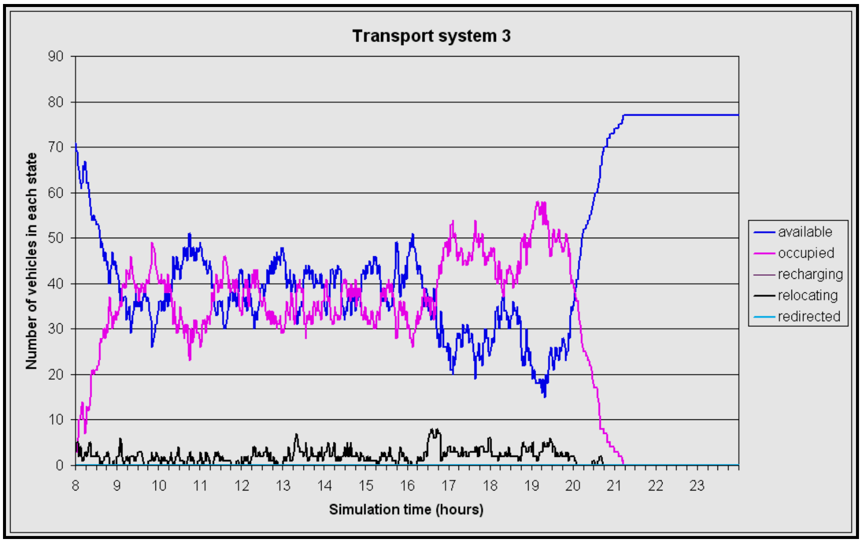


Fig. 7.20. Number of PICAVs in each state against time in the third management strategy (vehicles available also along the roads).

These diagrams show that the electric power provided to the PICAV vehicles and the selected charging technique (opportunity charging) are suitable for the case of study: all the vehicles do not reach a battery level below the minimum charge level. Therefore it results not necessary to stop the vehicles for charging purposes, but it is enough charging them while they are idle at the stations. Moreover these diagrams show that there are not redirected vehicles since the capacities of the stations are never reached. Therefore relocation strategies based on ZVT occurrences, guarantee that FPT occurrences do not take place.

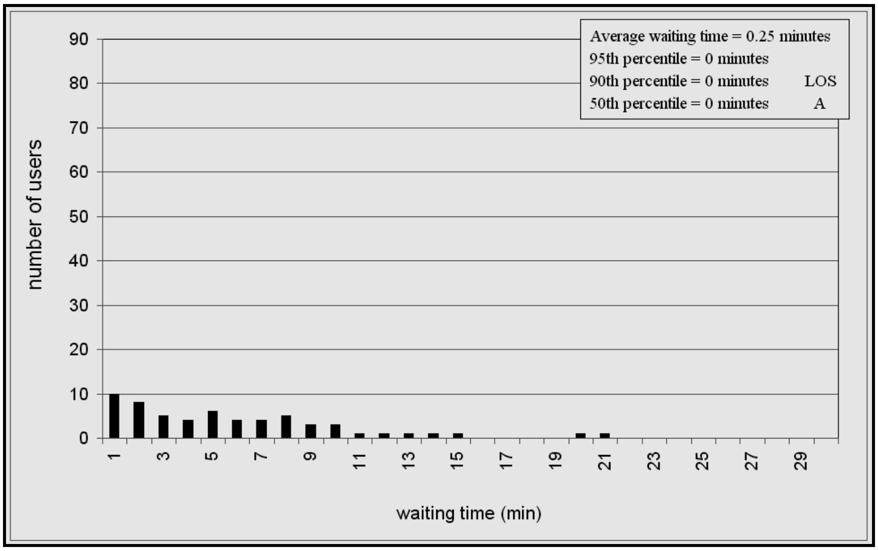


Fig. 7.21. Waiting times distribution in the first management strategy

If we consider available to users the vehicles occupied by users or available at parking lots, in the first relocation scheme 100% of the fleet is available to users in all the simulated period, since relocations are performed by users. In the second management strategy, in the peak phase, about the 97% of the fleet results available to users and only about the 3% is therefore relocating. In the third relocation scheme, always in the peak phase, about the 96% of the fleet results available to users and about the 4% is therefore relocating. This means that the simulated relocation strategies work quite well for the case of study, since the number of vehicles subtracted to users for relocations is low.

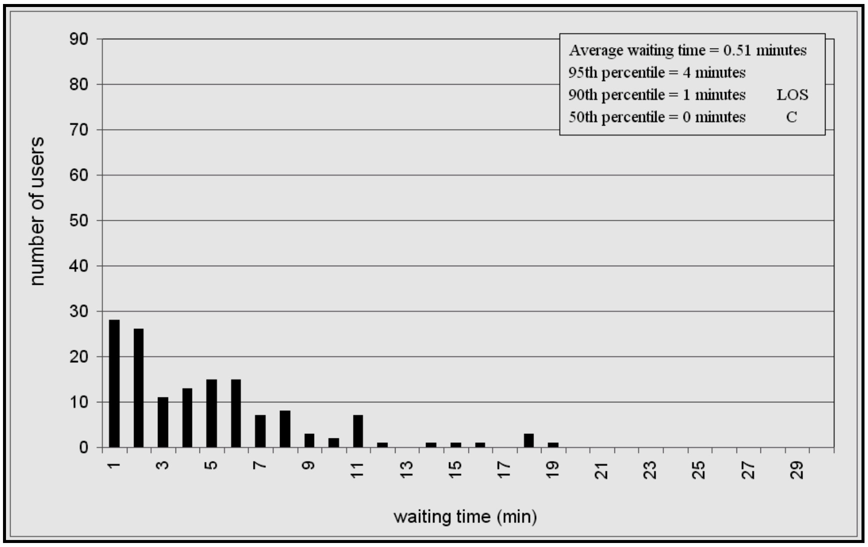


Fig. 7.22. Waiting times distribution in the second management strategy

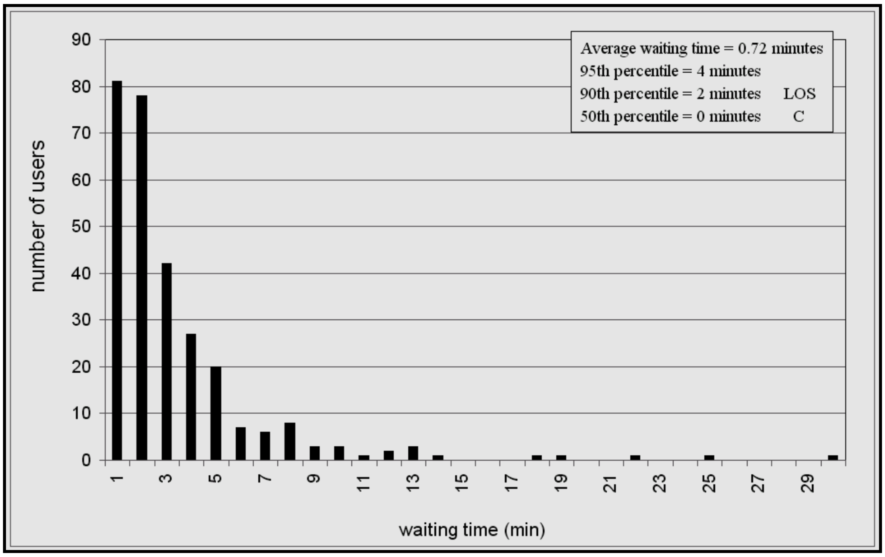


Fig. 7.23. Waiting times distribution in the third management strategy

The distribution of users waiting times is shown in figures 7.21, 7.22 and 7.23. It should be noted that the diagrams refer to only the users that have been in queue.

7.6. Conclusions

From the critical analysis of the distribution of the users waiting times, it results that the smallest average waiting time is obtained from the first management strategy which provides a LOS A, and the highest average waiting time is obtained in the third one which provides a LOS C. But it is also worth to underline that the fleet required in the first case is equal to 91, while in the other two strategies it is respectively 81 and 77.

Regarding the greater average waiting time in the third management strategy, with respect to the second one, the reason may be the following: in the second management scheme a relocation is required when the number of vehicles in a given station at a given time instant goes below the station's low critical threshold. When this condition occurs, the station has a shortage of vehicles and, aiming at avoiding a possible queue in the next future, a request for a vehicle is generated. In the third management strategy instead a request for a vehicle is made if a user is already in queue at an unit. In the second management scheme therefore the relocation strategy aims to prevent the queues and therefore to avoid users waiting at stations, whilst in the third management scheme the relocation aims to reduce as much as possible waiting times in the units, that however could not be avoided because the vehicle takes some time to reach the user's position. We have to take also into account the fact that in the third management scheme the user has to walk less than in the other proposed strategies. In fact the vehicle reaches the user in the unit where he is, and the user does not have to reach by foot the station, as it happens instead in the first and second management schemes.

As it concerns the first management strategies, the presented results are strictly linked to the fact that the transport demand is balanced, i.e. in the simulation day the total number of users that enter and exit the pedestrian area is the same. If instead the demand is unbalanced, the consequences are severe on the users waiting times, especially in the two inner stations, which do not belong to any choice set. In these stations in fact several vehicles will gather, while in the stations on the area border there will be several users in queue and relocations will be impossible. Moreover, in the inner stations the capacity may be reached. The second and third relocation schemes instead work well also in case of unbalanced PICAV transport demand.

The optimized values of the objective function are displayed in table 7.4. All the other observations on the objective function are described in detail in the chapter 9, "sensitivity analysis".

As stated in the chapter 5, the cost of each minute of waiting time has been taken equal to 0.1 €/minute. Moreover, the relocation cost has been taken equal to 0.01 €/minute.

Table 7.4. The optimized value for the objective function, and its components, in the three management strategies

Management strategy	1st	2nd	3rd
Objective function	436.57	477.84	485.61
Cost of waiting	41.10	90.42	117.37
Cost of relocation	0.00	33.12	30.41
Cost of the fleet	395.47	354.30	337.83

Other researchers studied the performance of shared vehicle systems. In particular Barth and Todd (1999) developed a model and applied it to a resort community in Southern California. Their model shows that the shared vehicle system is most sensitive to the vehicle-to-trip ratio. Barth and Todd considered that for the majority of the cases they analysed, the best number of vehicles to place in the system ranges from 3 vehicles per 100 trips to 6 vehicles per 100 trips. We have 1644 trips per day. The optimum fleet dimension for the three strategies analyzed are:

1. 91 vehicles, which provides a vehicle-to-trip ratio value equal to 0.055
2. 81 vehicles, which provides a vehicle-to-trip ratio equal to 0.049
3. 77 vehicles, which provides a vehicle-to-trip ratio equal to 0.047

Therefore the outcomes of the proposed transport system keep in the range described by Barth and Todd. This is obviously not enough to assess the validity of the proposed system: in fact a coherence analysis has been performed in the chapter 6.

A last aspect to underline is the following:

- in the second management strategy the low critical threshold is equal to 1 and the low buffer threshold is equal to 5;
- in the third management strategy the low critical threshold is equal to 2 and the low buffer threshold is equal to 5.

The different value of the low critical threshold registered in the second and third management strategy is not casual. Indeed, in the third management strategy the system can be accessed also along the roads. When the user arrives at an unit, he requires a vehicle, and the nearest PICAV reaches the user’s position. The nearest PICAV can be available both along the road or at a station.

At first we assumed that a station could be able to send a vehicle only if its number of available vehicles is above the low buffer threshold. But because of that, in the great majority of cases units, and not stations, provided the vehicles to the users. This lead to longer trips, as the vehicle in the majority of cases wasn’t provided by a closer station but by a far unit. Moreover, in several cases users had to wait longer for the vehicle, because no stations and no units could deliver him a vehicle.

As a result, we assumed that stations could provide vehicles to users waiting along the roads also if their number of available vehicles is still above the low critical threshold, but below the low buffer threshold. This explains why in the third management strategy the low critical threshold is greater than in the second one. In fact, in the second management strategy vehicles can be accessed only at stations and the low critical threshold is used to determine only when and where a relocation is necessary.

Chapter 8. Barreiro case study.

8.1. Introduction. The study case.

The second scenario analyzed in the present thesis is Barreiro city centre. Barreiro is a suburb of Lisbon, Portugal. This village is on the edge of a peninsula, on the left side of the Tago River. Lisbon is on the right side, in front of Barreiro. The intervention area, i.e. the old part of Barreiro, is about 1 square kilometer. The area is almost flat. Roads form a grid and vehicles are allowed to circulate in the area. The road pavement is quite irregular and therefore the speed is limited. Several parking spaces exist in the most peripheral parts of the intervention area and the parking here is for free. However, in the most inner part of the intervention area, due to narrow roads and lack of spaces, the parking is restricted.

The main ways to access Barreiro centre are: by car, by boat, by train, by bus. Regarding the access by car, there are three main roads connecting Barreiro centre with the outer area. Along the river, a wide road, called Avenida Bento Gonçalves, connects Barreiro centre with Lavradio (see fig. 8.1) and with the industrial area (north-east direction). In perpendicular, two roads, respectively rua Miguel Bombarda and rua Miguel Pais, connect the Barreiro centre with the mainland (south-east direction). In figure 8.1 the Barreiro surrounding areas are shown. In figure 8.2, the PICAV intervention area is shown.

By *boat*, Barreiro can be accessed from the fluvial terminal. The fluvial terminal is located in south-west side of the old town and a high frequency service by boat connects Barreiro with Lisbon. In the morning, people living in Barreiro take the boat to reach Lisbon, and in the evening return home after the working day again by boat.

It is possible to reach Lisbon from Barreiro also by *car*, but it takes several kilometres, along the XXV April bridge, which connects Barreiro to the city centre of Lisbon, and the bridge Vasco de Gama, which connects Barreiro to the northern part of Lisbon.

By *train*, Barreiro is accessible from the main station of Barreiro. The railway line passes on the border of the old part of Barreiro and two are the railway stops of interest: Barreiro Centre, referred by Ferreira (2009) as “Barreiro A”, and the one placed at the fluvial terminal, as shown in figure 8.2. Trips by train connect Barreiro to the villages in the mainland, on the other side of Lisbon. Commuters who work in Lisbon and live in the villages of the Barreiro area take the train, get off at Barreiro fluvial terminal and take the boat to Lisbon. The demand to/from Barreiro centre by train is much low, as it only consists on the Lisbon’s suburbs.

Several *bus* lines cross the old part of Barreiro, and several stops are also in the intervention area. The roads travelled by buses within Barreiro centre are: Avenida Bento Gonçalves, Rua Miguel Bombarda, Rua Miguel Pais, Avenida Alfredo da Silva. See figure 8.3 and 8.5.

The old part of Barreiro is in part a residential area (zones 1, 3 and 4 in figure 8.3), where most of the population (about 18000 residents) is aged and in part a commercial area (partially zone 2 and mainly zone 5 in figure 8.3). Zone 5 is characterised by a big new commercial centre recently built, called Forum Barreiro.

In the Barreiro old town, PICAV vehicles are expected to travel on sidewalks (see figure 8.2 a-b).

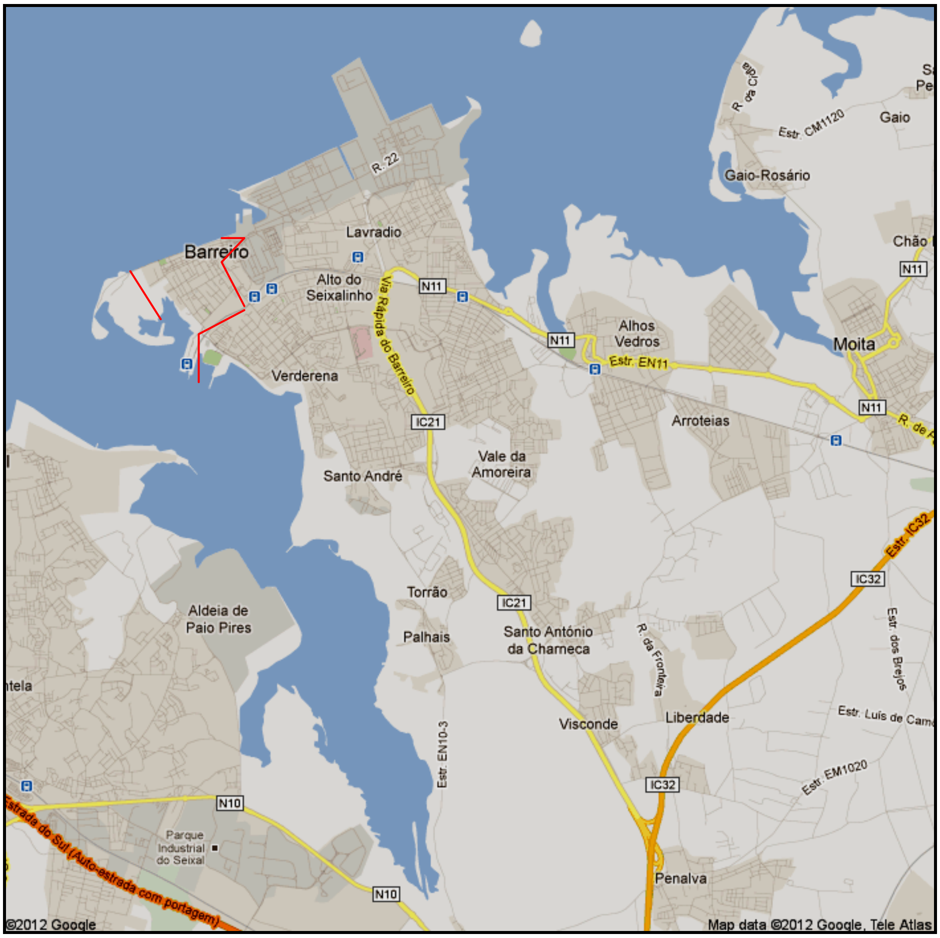


Fig. 8.1. Map of Barreiro municipality: the intervention area is circled through a red line.

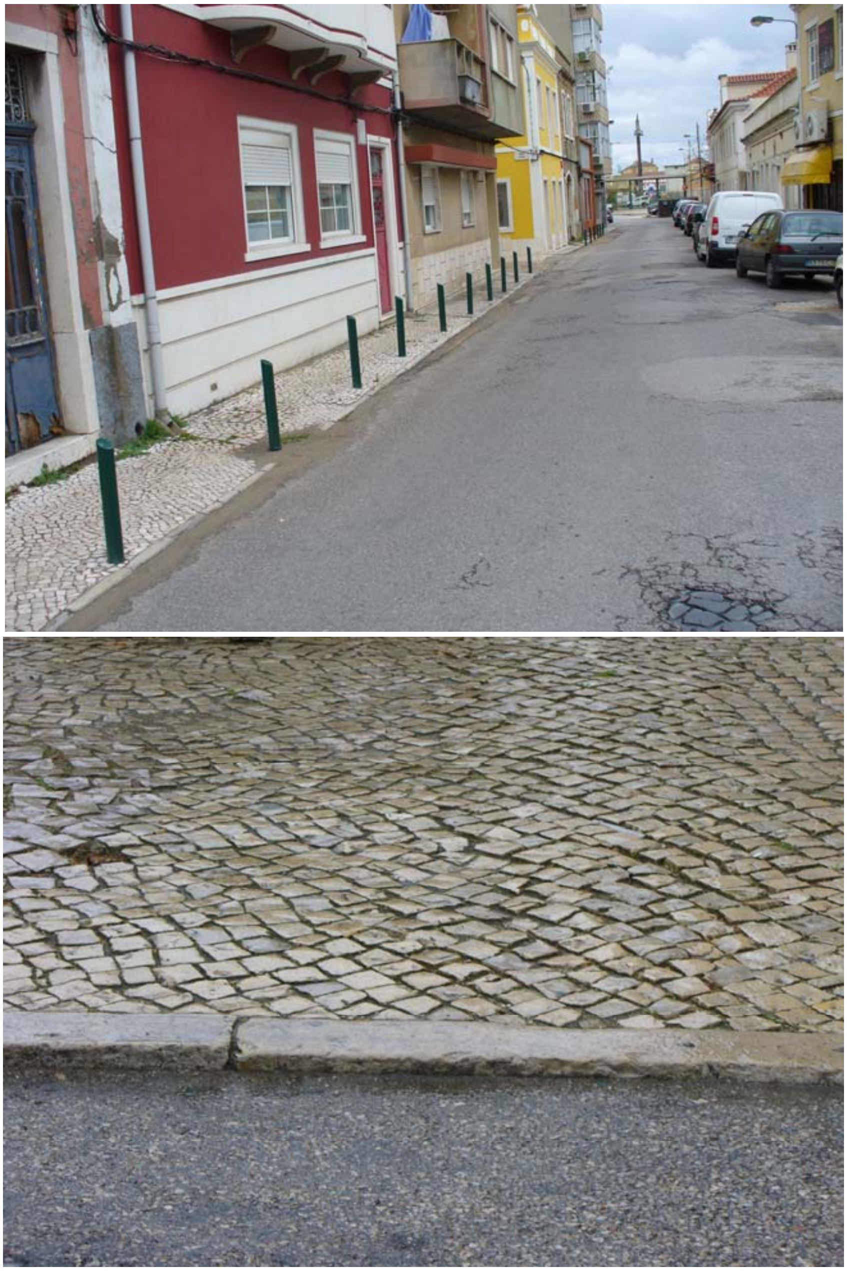


Fig. 8.2 a-b. Images of sidewalks in Barreiro.

8.2. Data collection.

8.2.1. The transport demand

The transport demand in the PICAV intervention area in Barreiro has been assessed with reference to the data provided by Transportes Collectivos do Barreiro (TCB) (Ferreira, 2009). The trips that interest the area are: trips that have the intervention area as origin or destination and trips internal to the area, that have origin and destination inside the area. There are not trips that cross the area and have origin and destination outside it.

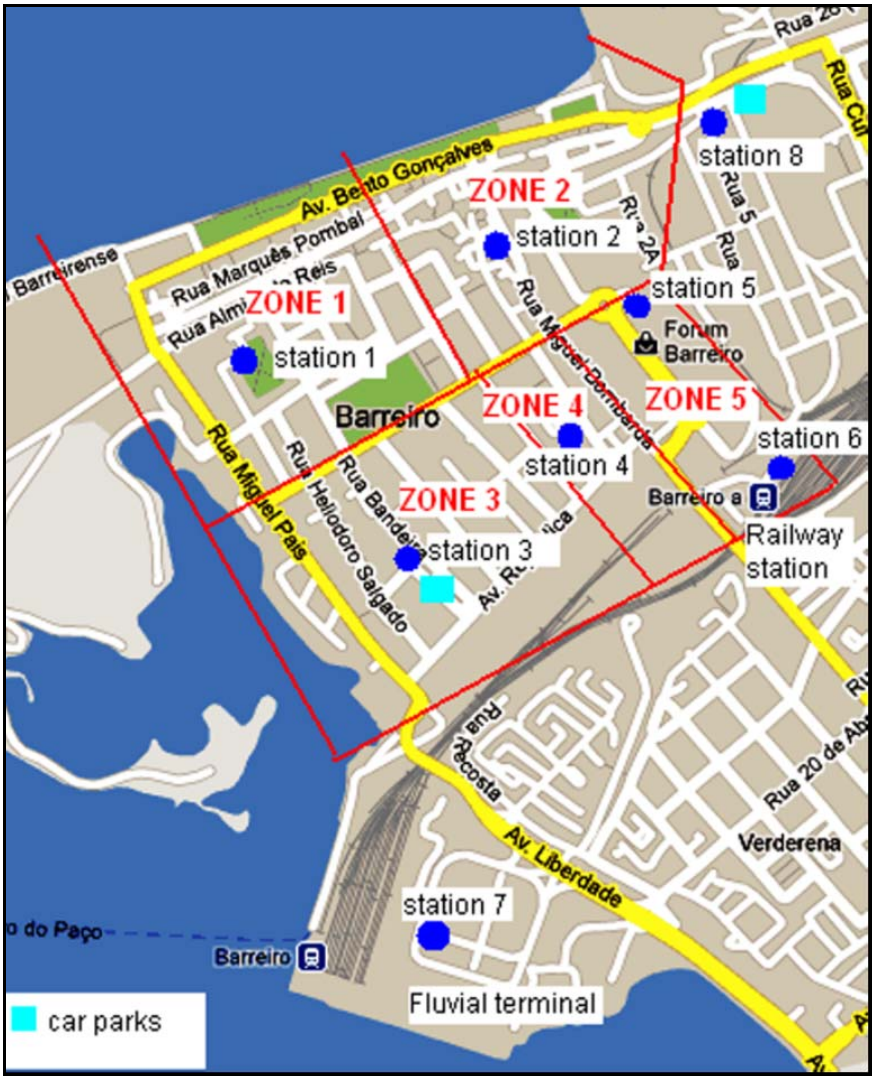


Fig. 8.3. The PICAV intervention area.

Trips to/from Barreiro happen mainly:

- by boat, to/from the fluvial terminal. The trips by boat connect Barreiro with Lisbon.
- by train: to/from the main Barreiro station. Trips by train connect Barreiro to the villages in the mainland, on the other side of Lisbon.
- by car.

Internal trips take place mainly by bus and by foot.
In the figure 8.4 the displacements for study/work reasons in the morning peak hour from Barreiro to the neighbour municipalities are reported.

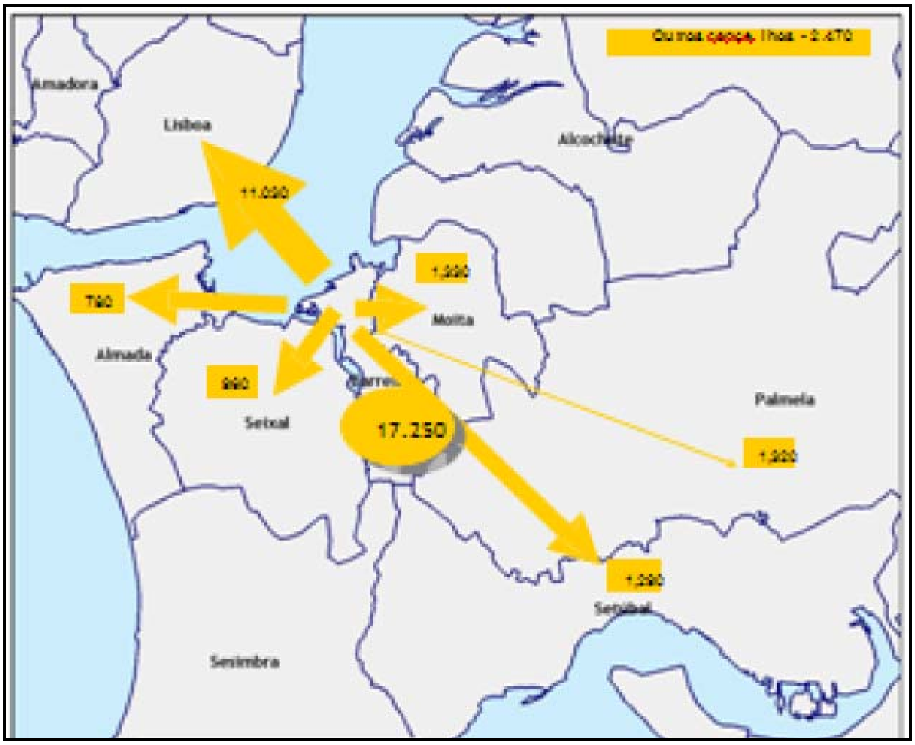


Fig. 8.4. Trips from Barreiro municipality in the morning peak for work/study reasons (Ferreira, 2009).

In the trips to Lisbon, about 79% of trips are performed by public transport (TC), whereas only 19% of trips take place by individual transport. Among the trips carried out through public transport, about 60% take place by boat and 34% by coach (along the highway and XXV April Bridge); the trips which take place by train instead are only 1% (Ferreira, 2009).

The trips from Barreiro to the other municipalities, are performed about 50% through public and 50% through individual transport. The modes of public transport mainly used are bus and train. The railway station is indicated in figure 8.2 “Railway station” besides the station 6 of PICAV. The other station besides the fluvial terminal is used only to commute from train to boat in trips to/from Lisbon, and not to visit Barreiro old town. During the morning rush hour, trips are mainly performed by train from Setubal and from the other municipalities to Barreiro fluvial station, and then the trips from Barreiro fluvial station to Lisbon are performed by boat.

8.2.1.1. Road transport

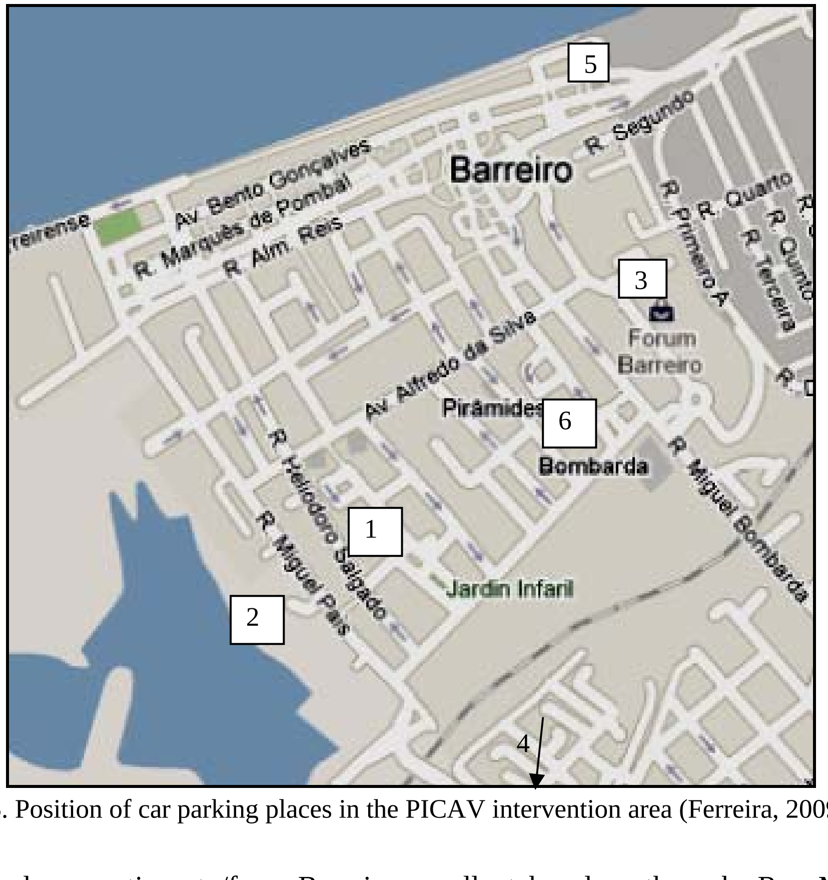


Fig. 8.5. Position of car parking places in the PICAV intervention area (Ferreira, 2009)

The road connections to/from Barreiro usually take place through: Rua Miguel Bombarda, Rua Miguel Païs and Avenida Bento Gonçalves. From an analysis carried out by TCB (the public transport company of Barreiro), it has been assumed a traffic flow of: 262 vehicles in the morning period in Rua Miguel Païs, 1300 vehicles in Rua Miguel Bombarda and 1214 in Avenida Bento Gonçalves. In the afternoon, traffic flows are equal to 140 vehicles per direction in Rua Paìs, 1490 vehicles per direction in Rua Bombarda and 1126 vehicles in Av. Bento Gonçalves.

There are two wide spaces for car parking on the border of the intervention area, displayed in figure 8.5:

1. near the the palace of Coimbra (square 1),
2. all along Rua Miguel Pais, on the side of the river (square 2),

There are also other parking spaces:

3. besides the Forum Barreiro, but it is dedicated to it (square 3),
4. besides the fluvial station,
5. along Avenida Bento Gonçalves (square 5), and
6. in the roads besides Av. da Republica (square 6).

8.2.1.2. Bus transport

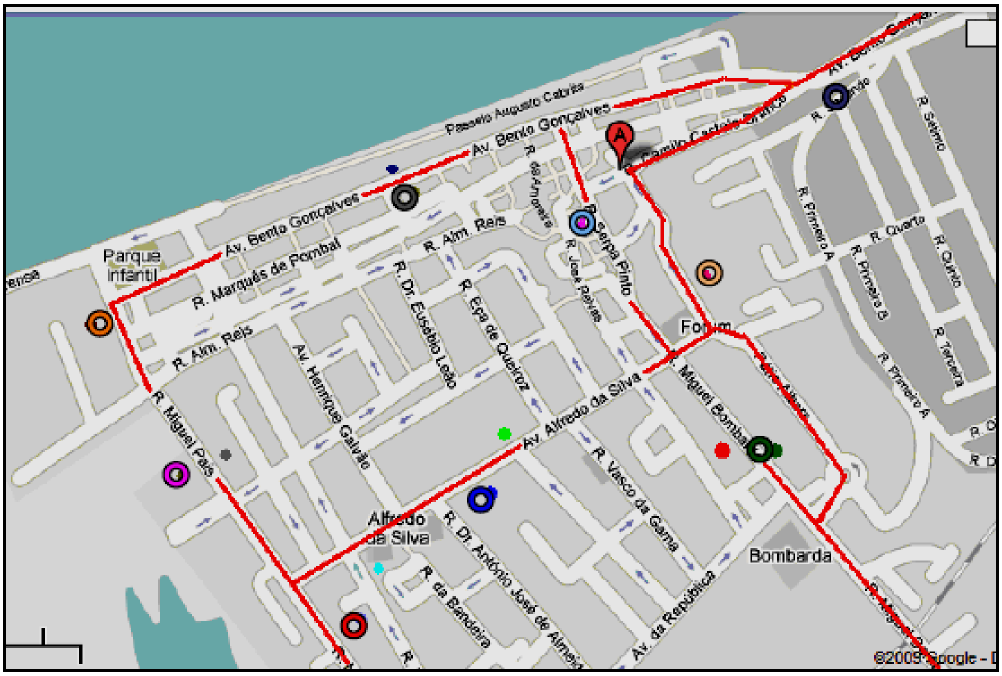


Fig. 8.6. Position of bus stops within the intervention area (Ferreira, 2009).

The localization of bus stops within the PICAV area has been assessed (Ferreira, 2009). The position of bus stops in the intervention area is reported in the figure 8.6; the number of passengers entering and exiting in each bus stop during the peak morning period is shown in table 8.1. The afternoon demand has been assumed equal to the opposite of the morning demand, i.e., passengers getting on the bus in the morning are the same who get off the bus in the afternoon, and vice versa. The data reported in the table 8.1 refer to the morning peak period, i.e. from 7 am to 9 am.

Table 8.1: Number of passengers entering and exiting in each bus stop during the peak morning period (Ferreira, 2009).

Symbol	Bus Stop	Passengers	
		in	out
◎	Escola Alfredo da Silva	748	661
◎	Rua Miguel Pais (1)	69	69
◎	Rua Miguel Pais (2)	571	825
◎	Av. Alfredo da Silva	4428	4504
◎	Rua Miguel Bombarda	2644	2308
◎	Rua D. Manuel de Melo	39	6
◎	Travessa de St.ª Cruz	9	12
◎	Largo das Obras	152	92
◎	Av. Bento Gonçalves	156	182

However, as shown in figure 8.5, the bus lines cross Barreiro intervention area, therefore it is unlikely that PICAV users may reach the intervention area by bus. Indeed travelers would prefer to stay on board for a couple of stops more rather than taking the PICAV vehicle.

8.2.1.3. Rail transport

In Barreiro there are two stations, i.e. Barreiro A, which serves the centre of Barreiro and the PICAV intervention area, and the fluvial terminal/rail–road interface, in which passengers get off the train and commute to river boat transport to reach Lisbon. The number of passengers to/from Barreiro Centre (i.e. getting off/on the train at Barreiro A) is equal to 50 passengers/day. There are no passengers who get on the train in Barreiro A and get off the train at fluvial terminal, as it is much easier to get to the fluvial terminal by different means of transport. All trips by train departing from Barreiro have as destination the municipalities behind Barreiro, as Setùbal, Moita, etc. The number of trips bay train during the day are reported in the fig. 8.7:

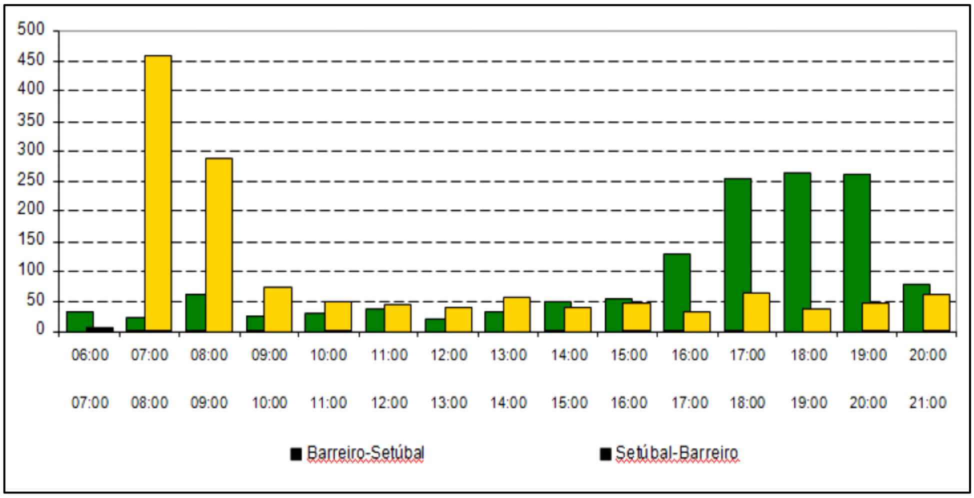


Fig. 8.7. Number of passengers on the trains arriving to or departing from Barreiro A station. In green the trips by train from Barreiro (A and fluvial terminal) to Setubal, and in yellow the trips by train from Setùbal to the two stations of Barreiro (Ferreira, 2009). In the x axis are the hours of the day, in the y axis is the number of trips.

8.2.1.4. *Fluvial transport*

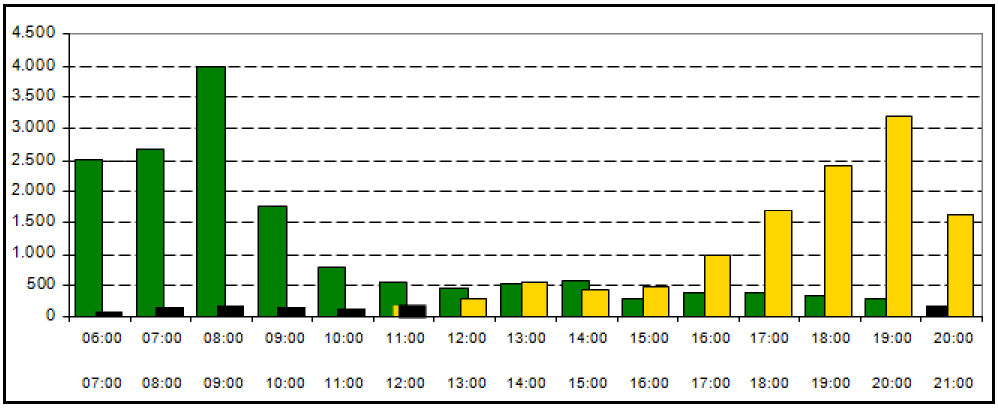


Fig. 8.8. Number of passengers entering and exiting at Barreiro fluvial station in each hour of the day. All these trips have Lisbon as origin or destination (Ferreira, 2009). The bars in the left refer to entrances to the fluvial station, the bars on the right refer to the exits.

Regarding the fluvial terminal, 2700 passengers/day have been registered, in each direction, during a working day, having origin or destination in Barreiro centre. It must be pointed out that, for trips performed by river and by train, but specially for trips by river, the area called “Barreiro centre”, does not refer only to the PICAV intervention area but to a much grrater area of Barreiro municipality. In particular, the trips having the PICAV intervention area as origin or destination are equal to the 30% of trips having origin or destination at Barreiro centre. The number of passengers entering and exiting at Barreiro fluvial station are shown in figure 8.8.

8.2.1.5. The assessment of the demand OD matrix

After several discussions with the public transport company in Barreiro (TCB), the following assumption have been done for assessing the PICAV transport demand:

- 10% of commuters arriving in Barreiro centre by car will be PICAV users;
- 20% of commuters arriving in Barreiro centre by boat and by train will be PICAV users;
- 20% of trips internal to the intervention area take place by PICAV. In this case users are mainly elderly and disabled people;
- 0% of commuters arriving in Barreiro centre by bus will be PICAV users.

Bus users have been considered not likely to use the PICAV vehicle, as there are several stops inside the intervention area and therefore they reach their final destination directly by bus.

The overall number of daily trips by PICAV, under the previous hypothesis, results 1296.

The simulation time period is a working day and two phases have been identified: a morning period from 7 a.m. to 1 p.m. and an afternoon period from 1 p.m. to 7 p.m. The demand has been assumed constant in each period. The overall number of trips is the same in the two periods but the OD matrix in the second period is the transpose matrix of the first period.

We identified 8 PICAV stations in the area: each one refers to a centroid. The exact localization of stations takes into account the available space and stations are placed as close as possible to interchange points. Their localization is shown in figure 3.

We identified 5 internal centroids and 3 external ones. Their positions are shown in figure 8.3. The three external centroids correspond to the fluvial terminal (centroid 6), the Barreiro A railway station (centroid 7) and a big car park space (centroid 8).

In particular, the station 6 is mainly an interchange point with the railway station; the demand to/from station 6 is indeed very little. The station 7 is an interchange point with the fluvial terminal. The stations 3 and 4 are interchange points with private car, in particular with the users coming from Rua Miguel Bombarda and Rua Miguel Paìs. In particular, station 3 is in proximity of a wide car park space, station 4 has in its neighbourhood several roads with much free parking space. They are also in a barycentric position in the area that they serve, in order to be easily reached by all houses presents in the area. Station 8 is also placed in an area close to some parking spaces, available in Avenida Bento Gonçalves, and it only plays the role of interchange point with private car.

Therefore: stations 3 and 4 play the role of both interchange points and internal stations. Station 5 serves the Forum Barreiro, the biggest commercial area in the village. Stations 6 to 8 only play the role of interchange points. Stations 1 and 2 are only internal stations, and are placed in the barycentric position of each zone. Station 2 in particular is placed in the old Barreiro central square.

The demand OD matrix for the morning period is displayed in the table 8.2.

Table 8.2. Demand OD matrix for the morning period

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**
1	0	1	7	3	3	0	5	1
2	0	0	0	0	0	0	0	0
3	7	1	0	3	3	0	5	1
4	2	0	3	0	1	0	2	1
5	2	0	3	1	0	0	2	1
6	0	0	0	0	0	0	0	0
7	5	0	5	2	2	0	0	0
8	1	0	1	0	0	0	0	0

8.2.2. The network

In the simulation, pedestrian density has been assumed the greatest one registered in Barreiro footpaths which is equal to 0.2 pedestrians per square meter.

The station capacity has been assumed equal to 15 PICAV for each station. The simulator receives in input the distances among the various stations through which the network has been modelled. There is only one path between each couple of stations, which is, as in Genoa case study, the most used path, along which the greatest number of attractors (i.e. bars, restaurants, shops, etc.) are concentrated. Distances have been assessed again through Google Maps. The slopes of the paths has not been taken into account as Barreiro is almost flat.

8.2.3. Calculation of the other micro simulator’s inputs

In this section, the determination of the fleet dimension, of the minimum battery charge level and the calibration of the low critical and buffer thresholds are reported.

The first management strategy in which users are flexible has not been taken into account as the hypothesis on the users choice set is not consistent in Barreiro: nearly all stations cannot be interchangeable for users. Indeed, users who are going to take the boat cannot be redirected to any other station different from the fluvial terminal, and the same can be stated for users who have to return to their car or who have to take the train. The only exception may be for some close stations, such as station 4 and station 5, or station 2 and station 5, but for the user it is however a disutility because it increases the user’s walking distance. Another exception may regard users who have parked the car in some street between the stations 3 and 4, and the distance between the user’s car and the two stations is almost the same: but this is a very specific case.

Because also of the tiny dimension of the network, and the concentration of the points of interest, it has been considered not worthwhile to take into account the third management strategy, in which PICAVs are available also along the roads.

For the reasons exposed above, the only relocation strategy taken into account for Barreiro is the second one, i.e. automatic relocation among stations.

At the beginning of the simulation period, the fleet has been assumed equally distributed among stations.

The low critical and low buffer thresholds, as well as the fleet dimension, have been determined through the optimization procedure.

The optimum fleet dimension resulted equal to 80 vehicles, i.e. 10 vehicles per station. And this is consistent to the results by Barth and Todd (1999), for whom the vehicle-to-trip ratio is usually comprised between about 0.03 and 0.06 vehicles per trip: being the trips 1296 per day, the vehicle to trip ratio results equal to 0.061.

The optimized low critical thresholds are the same for all stations and equal to 2, the optimized low buffer thresholds are equal to 4. Again, the opportunity charging technique has been chosen, and the minimum level of battery charge is equal to 6%.

The high critical threshold is equal to 14 vehicles for each station and the high buffer threshold is equal to 12 vehicles for each station. These values guarantee to keep high the number of supporting stations, and on the other hand to avoid that a station in a given instant accepts a vehicle and in the following instant reaches the FPT situation.

8.3. Output of the simulator and conclusions

The *distribution of users' waiting times* has been assessed through micro simulation. The average waiting time results equal to 0.20 minutes. The 95th percentile of users waiting times results equal to 1.84 minutes, the 90th percentile equal to 0.04 minutes and the 50th percentile equal to 0 minutes. This provides a level of service C.

As it concerns the *efficiency* of the optimised transport system, in figure 8.7 the number of PICAVs is each state (occupied by users, available, in charge and relocating) is plotted against time. The time is expressed in minutes starting from midnight, therefore a time of 480 refers to 8a.m, a time of 960 refers to 4p.m. and a time of 1200 refers to 8p.m.

Figure 8.10 shows the distribution of user waiting times: the total number of users is 1296 and 1082 users have not been in queue. The optimised threshold values result efficient since: on one side, relocations are possible in fact waiting times are low (as shown in figure 8.9); on the other hand the number of relocations is kept low as it results in figure 8.9.

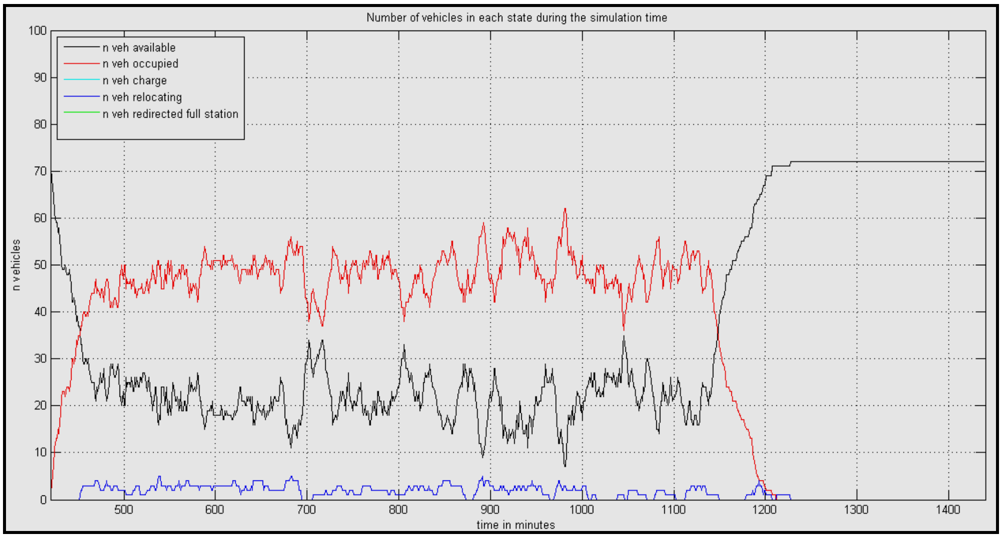


Fig. 8.9. Number of vehicles in each state during the simulation period.

The relocation cost is equal to 21.36 €/day, the cost of user waiting times is equal to 33.544 €/day and the objective function results equal to 405.08 €/day. It results evident that the cost of users is much lower that the system cost, this is due to the fact that the cost of waiting times decrease strongly with the fleet dimension.

The cost of each minute of waiting is equal to 0.10 €/minute (Cherchi et al.,); the cost of each minute of relocation is equal to 0.01 €/minute. This choice for the relocation cost has been made, in the same way as in the Genoa case study, in order to make the users' cost and the relocation's cost comparable in the range of the objective function close to the optimum.

Conversely, the relocation cost increases slightly with the fleet dimension. However it should be noted that the function values do not have any meaning since the function does not take into account flat costs, like for instance the ticket prices.

The results of the simulation clearly show the effectiveness of the automatic relocation, because, with low staff costs, it allows users a high level of satisfaction.

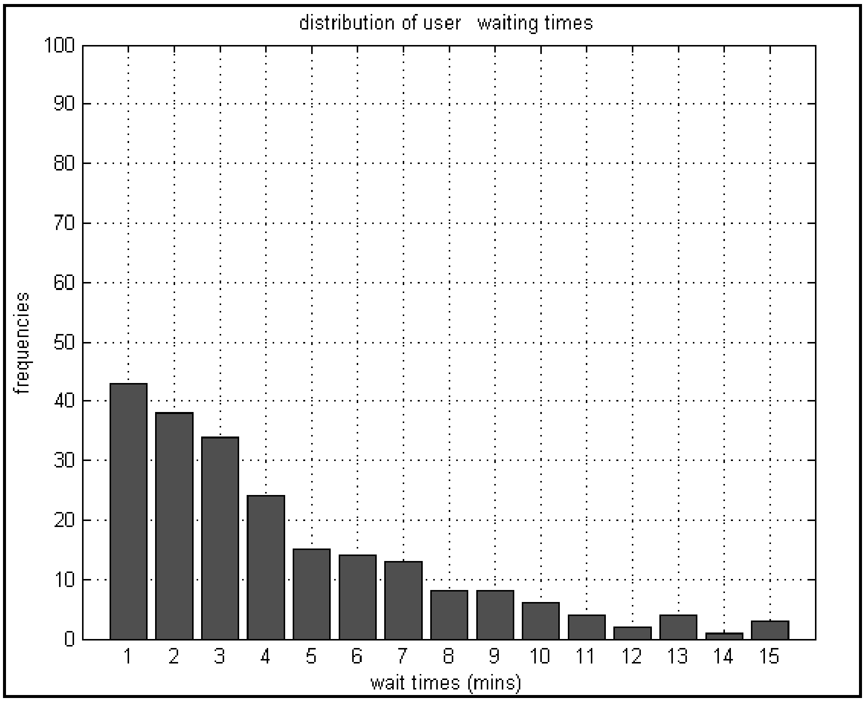


Fig. 8.10. Distribution of user's waiting times.

Chapter 9. Sensitivity analysis

Introduction

With reference to the two scenarios described above, i.e. Genoa and Barreiro, a sensitivity analysis has been performed for understanding to which extent each input affects the outputs in the micro simulator. This sensitivity analysis has been performed before writing the optimization code, in order to better understand which variables are more important and what is worth to optimize. The sensitivity analysis has been performed also after the optimization in order to control the coherence of the optimization's outputs.

From the sensitivity analysis it has resulted that it is not convenient to take different values for the thresholds in the various stations, as the influence on the cost function is of the same scale as the stochastic effects. For example, if instead of taking in the Genoa scenario all the critical thresholds equal to 1, but for example the threshold in the station 3, which in Genoa has the highest demand, is taken equal to 2, this does not affect the cost function noticeably. Moreover, it has resulted that no relationship exists between low critical thresholds and low buffer thresholds: they all must be optimized separately. For example, taking as a rule that all low buffer thresholds are greater of 2 than the low critical thresholds, this is incorrect as the cost function results greater. Also taking an initial different number of vehicles in the various stations is incorrect in the two management strategies involving automatic relocation: the relocation procedure vanishes the benefit from a better initial disposition of vehicles.

For the first management strategy, i.e. the one which involves flexible users, several values for the fleet dimension and several distributions of vehicles among stations have been taken. The model is much sensitive on both these parameters.

Another sensitivity analysis has been performed through modifying the transport demand. The results have shown that, for what regards the first management strategy, the system is heavily sensitive on the demand unbalancement, and that if the demand is strongly unbalanced, the system goes in crisis despite the relocation. For what regards the other two strategies, the relocation is able to compensate the demand unbalancement and therefore the users' waiting times increase as well as the relocations, but the system does not go into crisis.

9.1. Sensitivity analysis for what regards fleet dimension and thresholds

A sensitivity analysis has been performed separately for each management strategy under study, and it regards both Genoa and Barreiro cases of study.

9.1.1. No relocation

Firstly, some runs of the simulator have been performed in order to assess the necessity of relocation. Without any relocation scheme, with a fleet of 81 veh/h, the distribution of user waiting times results:

- average waiting time = 19.88 minutes;
- 50th percentile of users' waiting times = 81.52 minutes;
- 90th percentile of users' waiting times = 53.18 minutes;
- 95th percentile of users' waiting times = 9.05 minutes.

The resulting Level of Service is F, therefore the need for relocation.

9.1.2. First management strategy: flexible users. Genoa case study.

For what regards the first management strategy, several values of the fleet dimension have been taken, in order to have a first idea, point by point, of the trend of the objective function as a function of the size of the fleet. All these values are reported in the table 9.1.

- in the first case, i.e. for fleet dimension equal to 73 vehicles, its distribution among the stations is equal to: [8, 9, 10, 10, 11, 7, 6, 6, 6];
- in the second case, the number of vehicles in each station is increased by one: [9,10,11,11,12,8,7,7,7];
- and so on.

In the figure 9.1, the values of the objective function with reference to the fleet dimension are shown, for the first management strategy. The optimum fleet dimension is equal to 91. The two components of the objective function are explicited: the cost of the system, expressed in terms of the fleet cost, and the cost of the users. As the fleet dimension increases, the cost of the fleet increases, and the cost of waiting decreases. However, while the fleet's cost increases always of the same quantity, the cost of waiting does not improve significantly as the fleet dimension goes above its optimum value.

9.1.3. Second management strategy: automated vehicles. Genoa case study

The values of objective function have been calculated, by varying the fleet dimension and the low critical and low buffer thresholds. The results of this sensitivity analysis are shown in table 9.2. As stated in the chapter 5, the cost of each minute of waiting time has been taken equal to 0.1 €/minute. Moreover, the relocation cost has been taken equal to 0.01 €/minute.

Low critical and low buffer thresholds values have been taken the same for all stations. This is performed because no sensible improvement is achieved to the cost function under a high increase in the problem's complexity.

If different thresholds values are taken in the various stations, i.e. for example in the 1st station the low critical threshold is equal to 1 and in the 2nd station it is equal to 3, this does not result in a significant modification on the objective function. The variations of the objective function are in fact of the same greatness order as stochastic effects. Moreover, if different buffer threshold values are taken for the various stations, and their average is for example 4, then the distribution of user waiting times is not different from the case of all buffer threshold values equal to 4 in all stations. In the figure 9.2, the variation of the objective function against the fleet dimension is shown, for what regards the second management strategy. The objective function has been calculated by considering the optimized low critical and low buffer thresholds. In this case, the cost of the system involves also the cost of relocations.

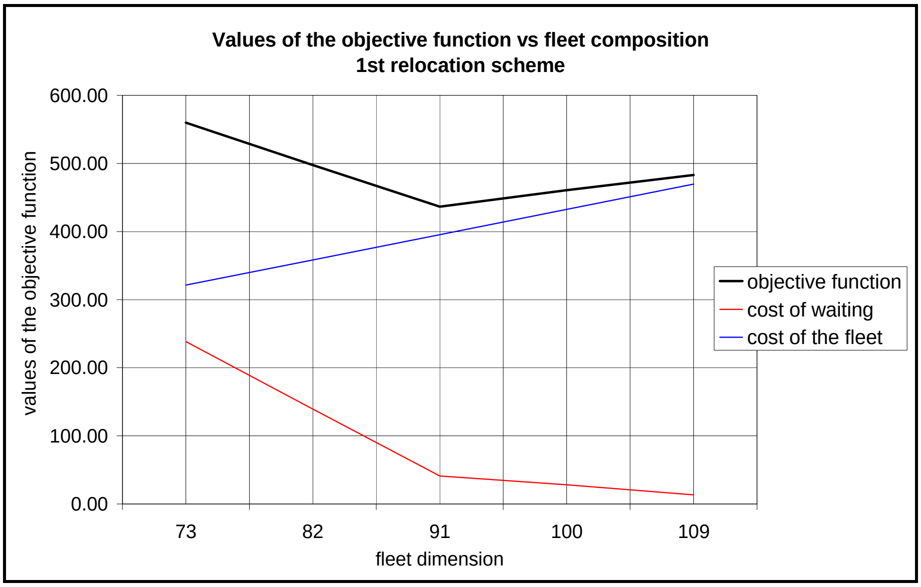


Fig. 9.1. Values of the objective function with reference to the fleet dimension, for the first management strategy

Table 9.1. Values of the objective function and its components versus fleet dimension. The optimum values are reported in bold.

fleet	objective function	cost of waiting	cost of the fleet
73	559.89	238.53	321.36
82	497.80	139.39	358.41
91	**436.57**	**41.10**	**395.47**
100	460.70	28.18	432.52
109	483.02	13.45	469.57

As shown in the figure, the cost function shows a minimum for a fleet dimension equal to 81. The fleet's cost increases linearly, as in the first management strategy, and the cost of waiting decreases in hyperbolical manner.

Table 9.2. Values of the objective function, and of its components, with reference to the fleet dimension, to low critical and low buffer thresholds values. Thresholds have been taken equal for all stations. The optimum values are reported in bold. Genoa case study.

fleet	low critical threshold	low buffer threshold	objective function	cost of waiting	cost of relocation	cost of the fleet
54	1	5	948.62	688.63	16.85	243.14
63	1	5	579.24	277.53	21.00	280.19
72	1	5	502.58	150.60	34.73	317.24
81	**1**	**5**	**477.84**	**90.42**	**33.12**	**354.30**
90	1	5	478.95	56.99	30.61	391.35
99	1	5	483.63	31.96	23.27	428.40
108	1	5	493.52	26.47	16.60	465.46
81	1	2	590.84	142.61	93.94	354.30
81	1	3	521.41	111.01	56.10	354.30
81	1	4	486.63	94.87	37.46	354.30
81	**1**	**5**	**477.84**	**90.42**	**33.12**	**354.30**
81	1	6	482.45	100.38	27.78	354.30
81	1	7	487.58	110.44	22.84	354.30
81	1	8	497.35	123.39	19.66	354.30
81	1	9	515.19	143.91	16.98	354.30
81	1	10	543.94	174.44	15.20	354.30
81	1	11	580.54	213.69	12.56	354.30
81	**1**	**5**	**477.84**	**90.42**	**33.12**	**354.30**
81	2	5	485.40	87.75	43.36	354.30
81	3	5	531.12	113.75	63.08	354.30

The relocation cost firstly increases and then decreases with the increase of the fleet: indeed for a fleet of 54 vehicles it is equal to 16.85 €/day, for a fleet of 72 vehicles it is equal to 34.73 € per day, while for a fleet of 99 vehicles it is equal to 23.27 € per day. It means that the total relocation time has increased from 1685 to 3473 minutes and then decreased from 3473 minutes to 2327 minutes, as the cost of relocation has been assumed equal to 0.01 €/minute. This result is expected. In fact, as the fleet is lower, the number of required relocations increases but the number of relocations which can actually take place is lower. As the fleet dimension increases above the optimum, the number of required relocations decreases. For what regards the user cost, similarly to the first relocation scheme, it decreases relevantly until the optimum fleet dimension is reached. For further increases in the fleet, the cost of the users decreases slightly.

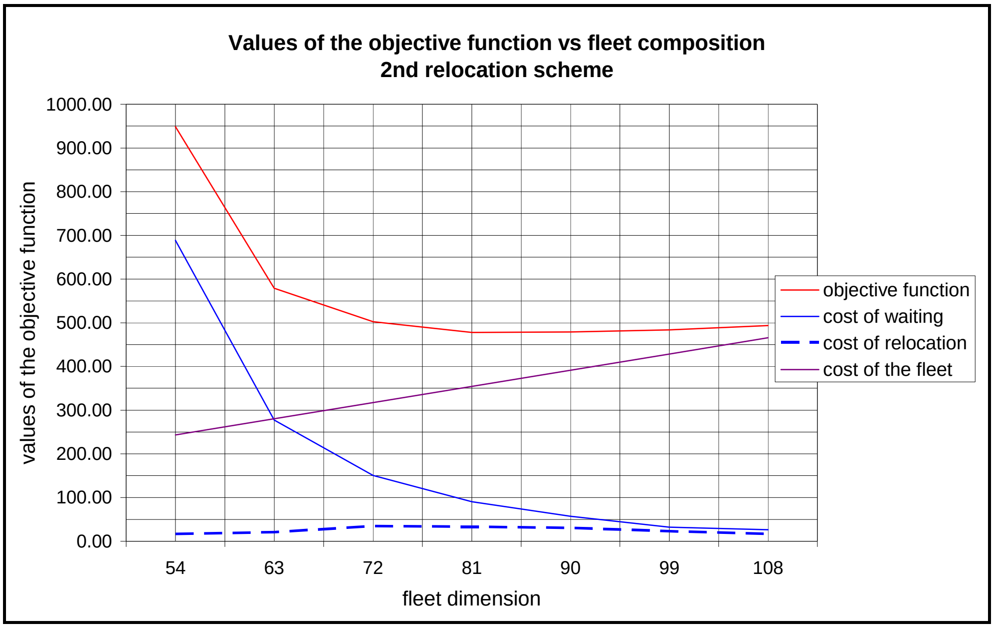


Fig. 9.2. Values of the objective function with reference to the fleet dimension, for the second management strategy. Thresholds are kept fixed and equal to their optimized values

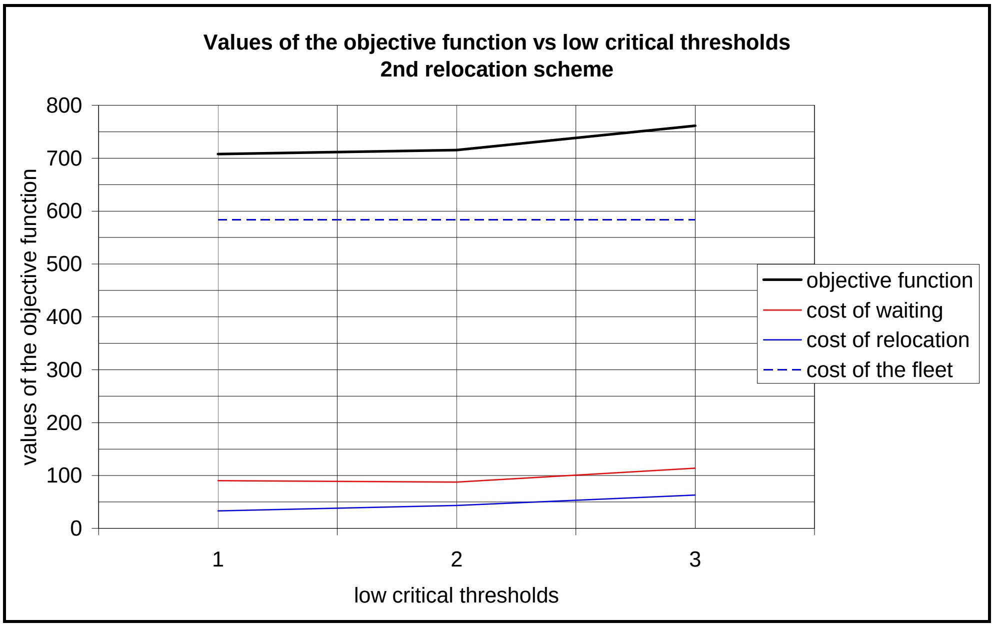


Fig. 9.3. Values of the objective function with reference to the low critical thresholds, for the second management strategy. The fleet dimension is fixed and equal to the optimized value. Low buffer thresholds are also kept fixed and equal to their optimized values. Genoa case study.

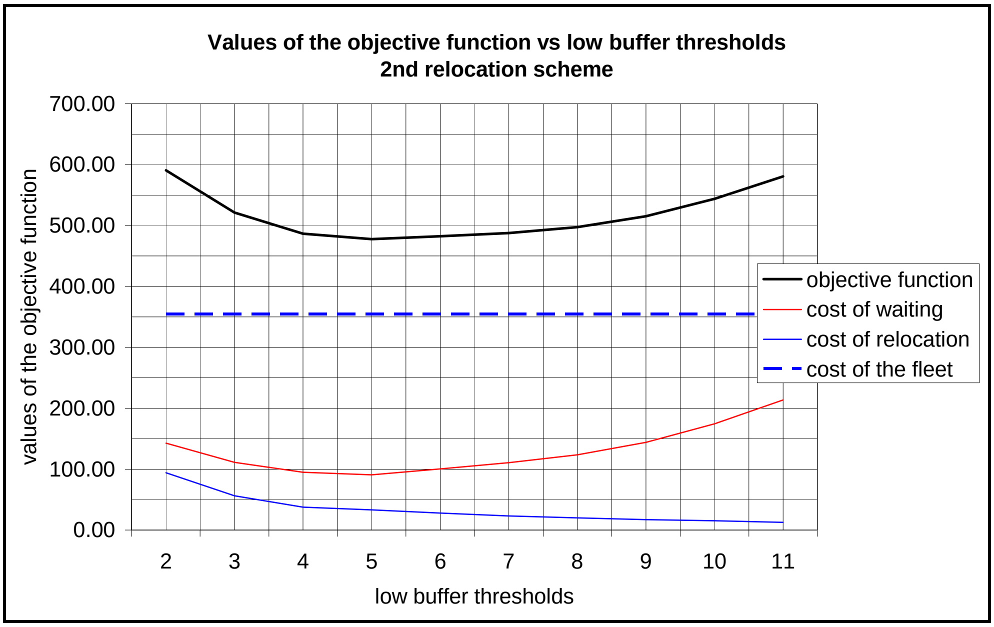


Fig. 9.4. Values of the objective function with reference to the low buffer thresholds, for the second management strategy. The fleet dimension is fixed and equal to the optimized value. Low critical thresholds are also kept fixed and equal to their optimized values. Genoa case study.

In the figures 9.3-9.4, the values of the objective function with reference to the low critical (figure 9.3) and low buffer (figure 9.4) thresholds' values are displayed. Again, the objective function has been calculated considering: the optimized fleet dimension, and the optimum buffer thresholds values for figure 9.3 and the optimum fleet dimension and optimum critical thresholds values for figure 9.4. From the sensitivity analysis, it resulted that both the users' costs and the relocation costs increase as the low critical threshold increases. Indeed, as low critical thresholds grow, the number of required relocations increases and vehicles are available to users for a shorter amount of time. This results in the fact that fewer required relocations can be performed and therefore the users' waiting times increase.

Moreover, as shown clearly by figures 9.3 and 9.4, the objective function is slightly sensitive to thresholds, and as thresholds vary it changes much less than as the fleet varies.

For what regards the low buffer thresholds, the cost of waiting is minimum for the optimum threshold value; the cost of the fleet keeps constant as the fleet has not been changed; the cost of relocation decreases. This can be explained because:

- If the low buffer thresholds keep beyond the optimum, the number of required relocations which take place increase, and this explains the greater relocation cost. However, it may often occur that a station provides a vehicle to another station in defect, and a few minutes after is itself in defect. This also increases the number of required relocations. Moreover, because the zero vehicle time situation occurs more often, also the users' waiting times increase.

- If the low buffer threshold is higher than the optimum value, the number of required relocations decrease, therefore the lower cost of relocations. Moreover, it very rarely occurs that a station provides a vehicle to another station in defect, and a few minutes after is itself in defect: and this also keeps the number of required relocations lower. However, also the number of required relocations which cannot take place increases, as the number of supporting stations decreases. And this last fact increases the user waiting times.

9.1.4. Second management strategy: automated vehicles. Barreiro case study

For what regards Barreiro, only the second management strategy is considered, i.e. automated vehicles, system accessed only at stations. A similar sensitivity analysis as in Genoa case study has been performed and it has shown similar results. The values obtained through variation of the fleet dimension are shown in table 9.3 and figure 9.5. The thresholds values are taken constant and equal to their optimum value. As the fleet dimension increases, the relocation cost also decreases, as expected: much less relocation procedures are needed to avoid the zero vehicle time occurrences.

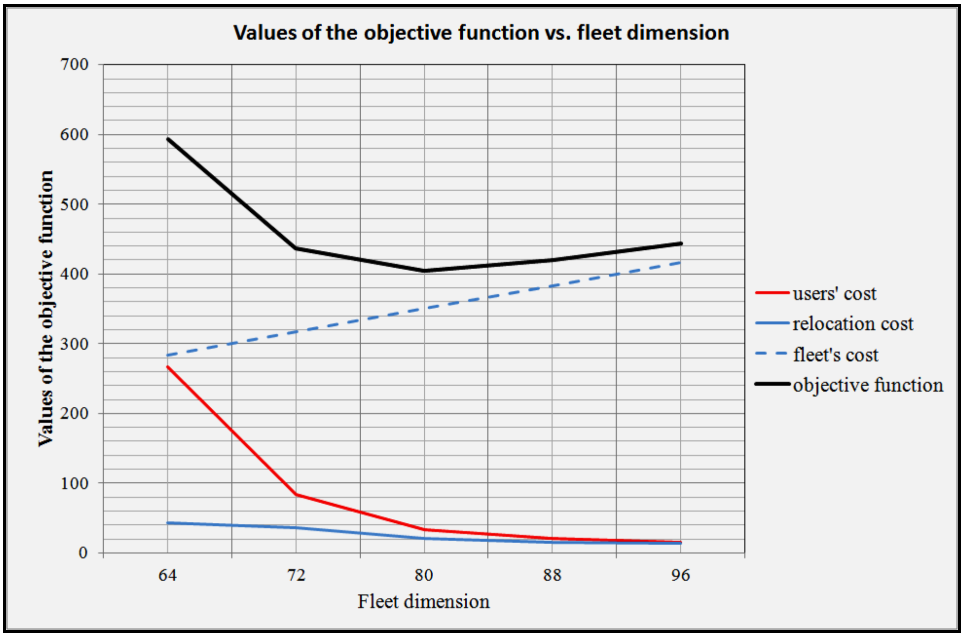


Fig. 9.5. Values of the objective function with reference to the fleet dimension. The low critical and low buffer thresholds are taken constant and equal to their optimized values. Barreiro case study.

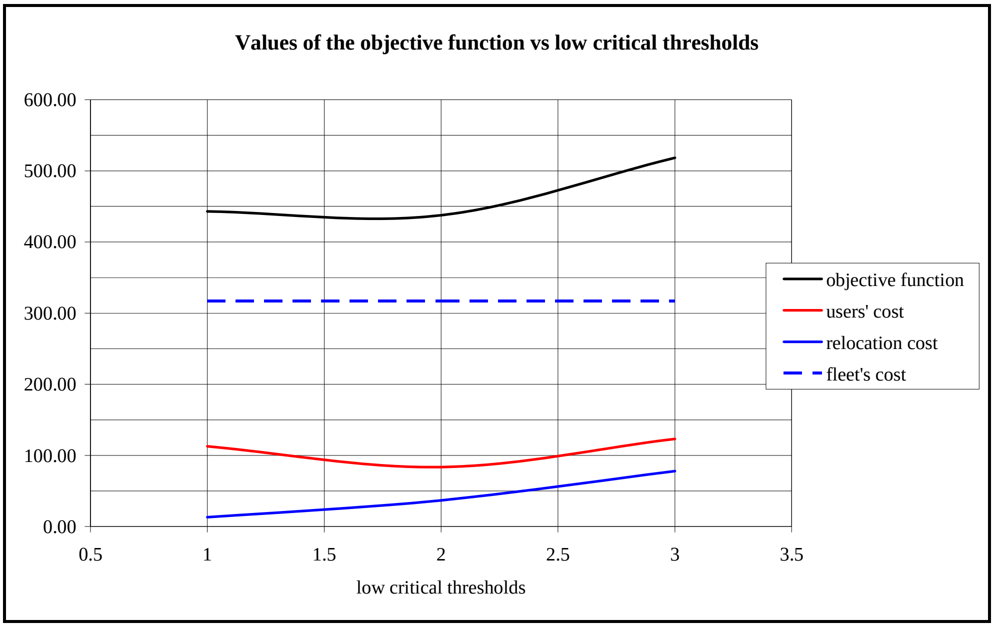


Fig. 9.6. Values of the objective function against low critical thresholds. Fleet dimension and the low buffer thresholds are kept constant. Barreiro case study.

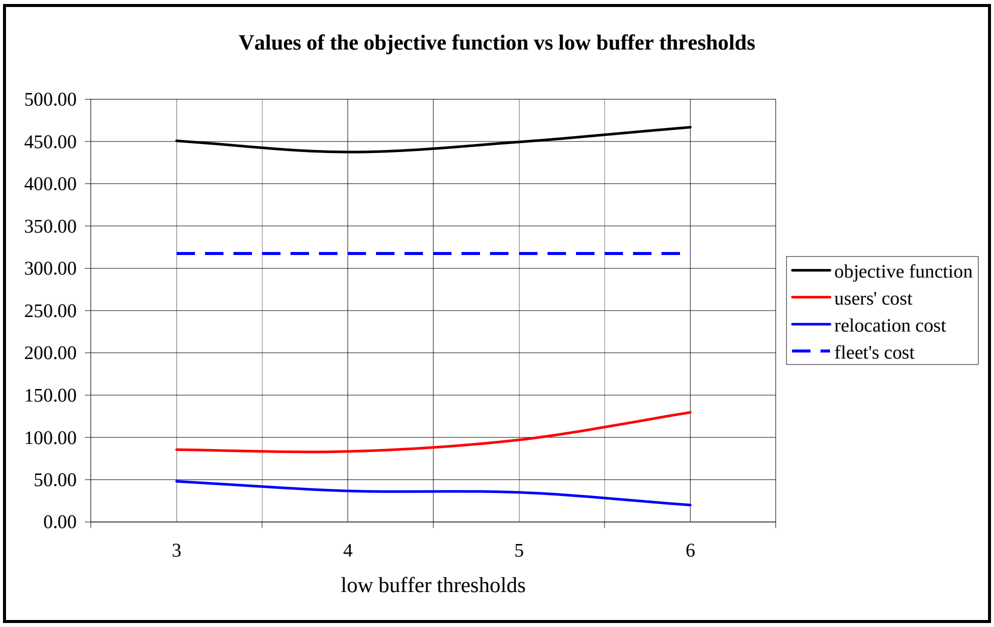


Fig. 9.7. Values of the objective function against low buffer thresholds. Fleet dimension and the low critical thresholds are kept constant. Barreiro case study.

As stated several times in this thesis, the cost function is much sensitive to the fleet dimension: here, if the fleet goes below 72 vehicles, the cost function peaks.

For what regards the variation of the objective function with thresholds, it shows the same behaviour as in Genoa case study, and the same things could be stated. The model works properly because, applied in different scenarios, it must show the same behaviors. The only difference is the value of the optimum low critical threshold, i.e. 2 in Barreiro and 1 in Genoa, and this does not constitute a problem because threshold values are parameters which depend on the specific case study.

Table 9.3. Sensitivity analysis on the fleet dimension. The optimum values are in bold. Barreiro case study.

fleet	low critical threshold	low buffer threshold	average waiting time	percentile			LOS	total relocation time	users' cost	relocation cost	fleet's cost	cost function
				95th	90th	50th						
64	2	4	1.62	10.1	7.1	0	F	4295.16	266.69	42.95	284.31	593.95
72	2	4	0.50	4.9	2.4	0	C	3681.14	83.38	36.81	317.24	437.43
80	**2**	**4**	**0.20**	**1.8**	**0.1**	**0**	**A/B**	**2135.86**	**33.54**	**21.36**	**350.18**	**405.08**
88	2	4	0.13	0.6	0	0	A	1580.44	21.21	15.80	383.11	420.13
96	2	4	0.09	0.1	0	0	A	1346.98	14.73	13.47	416.05	444.25

9.1.5. Third management strategy: capillarity. Genoa case study.

In table 9.5 and figure 9.8, the values of the objective function with reference to the fleet dimension is shown, for the third management strategy. As in the other two strategies, the objective function shows to be sensitive to the fleet dimension, and the cost of waiting, of the fleet and of relocations shows the same ongoing as in the other two strategies.

In particular, for fleet dimension smaller than a certain quantity, the user cost peaks, and this is similar to the other two relocation procedures described above.

In figures 9.9 and 9.10, with the fleet dimension kept constant and equal to the optimized value, the objective function and its components are displayed with reference to the low critical and low buffer thresholds values.

The objective function shows the same behavior as in the second strategy, also for what regards all its components (i.e. cost of the fleet, of the relocation, of the users).

Table 9.4. Sensitivity analysis on the low critical and low buffer thresholds. Barreiro case study.

fleet	low critic. threshold	low buffer threshold	average wait time	percentile			LOS	n° relocations	total relocation time	users' cost	relocation cost	fleet's cost	objective function
				95th	90th	50th							
72	1	2	0.95	5	2	0	C	220	2000	123.12	20.00	317.25	460.37
72	1	3	0.94	5	2	0	C	170	1500	121.82	15.00	317.25	454.07
72	1	4	0.87	4	2	0	C	120	1300	112.75	13.00	317.25	443.00
72	1	5	0.97	4	2	0	C	110	1500	125.71	15.00	317.25	457.96
72	1	6	1.21	5	2	0	C	100	1400	156.82	14.00	317.25	488.07
72	2	3	0.66	3	1	0	B	430	4800	85.54	48.00	317.25	450.79
72	2	4	0.55	3	1	0	B	300	3681	83.38	36.81	317.24	437.43
72	2	5	0.75	3	1	0	B	230	3500	97.20	35.00	317.25	449.45
72	2	6	1	5	2	0	C	150	2000	129.60	20.00	317.25	466.85
72	3	4	0.95	4	3	0	C	580	7800	123.12	78.00	317.25	518.37
72	3	5	0.86	4	2	0	C	360	5100	111.46	51.00	317.25	479.71
72	3	6	1.03	5	3	0	C	230	3500	133.49	35.00	317.25	485.74
72	3	7	1.55	7	4	0	D	130	2000	200.88	20.00	317.25	538.13

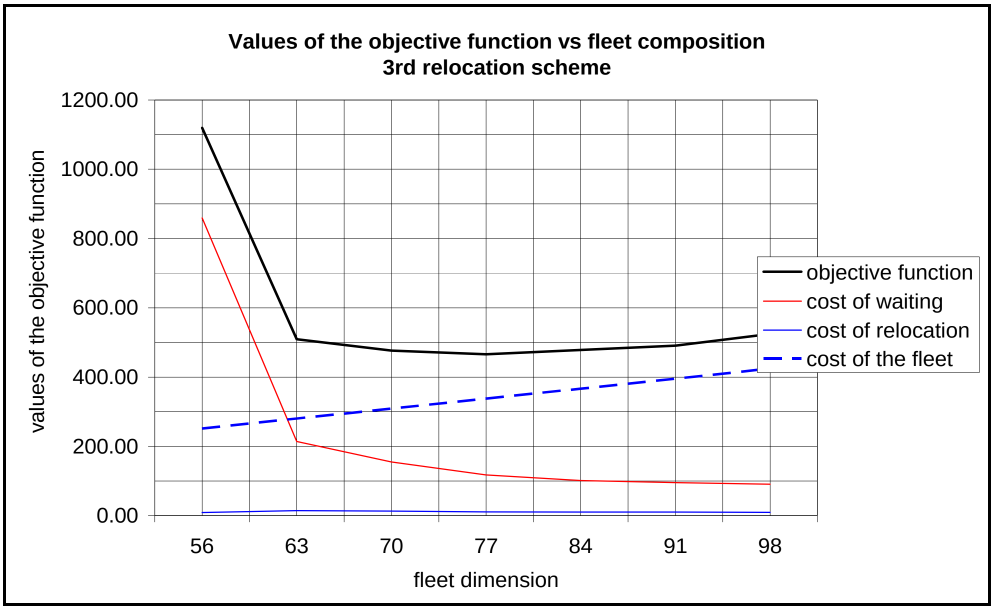


Figure 9.8. Values of the objective function with reference to the fleet dimension, for the third management strategy. Thresholds are kept fixed and equal to their optimized values. Genoa case study.

Table 9.5. Values of the objective function with reference to the fleet dimension for the third management strategy. The optimum values are reported in bold. Genoa case study.

fleet	low critical threshold	low buffer threshold	objective function	cost of waiting	cost of relocation	cost of the fleet
56	2	5	1129.38	859.4	18.61	251.37
63	2	5	528.87	213.94	24.74	280.19
70	2	5	486.63	154.59	23.03	309.01
77	**2**	**5**	**475.61**	**117.37**	**20.41**	**337.83**
84	2	5	488.15	101.36	20.14	366.65
77	1	5	560.19	117.19	105.17	337.83
77	**2**	**5**	**475.61**	**117.37**	**20.41**	**337.83**
77	3	5	487.91	123.16	26.92	337.83
77	4	5	537.04	156.11	43.10	337.83
77	2	3	499.20	137.18	24.19	337.83
77	2	4	485.16	125.53	21.80	337.83
77	**2**	**5**	**475.97**	**117.37**	**20.77**	**337.83**
77	2	6	488.70	131.48	19.39	337.83
77	2	7	504.19	147.70	18.66	337.83
77	2	8	523.04	167.07	18.14	337.83

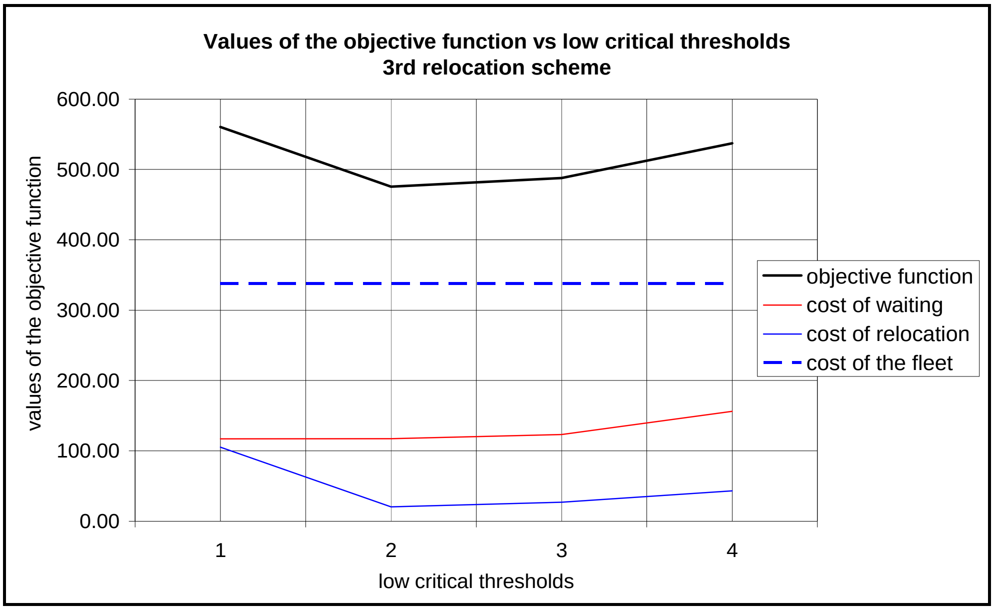


Figure 9.9. Values of the objective function with reference to the low critical thresholds, for the third management strategy. The fleet dimension is fixed and equal to the optimized value. Low buffer thresholds are also kept fixed and equal to their optimized values. Genoa case study.

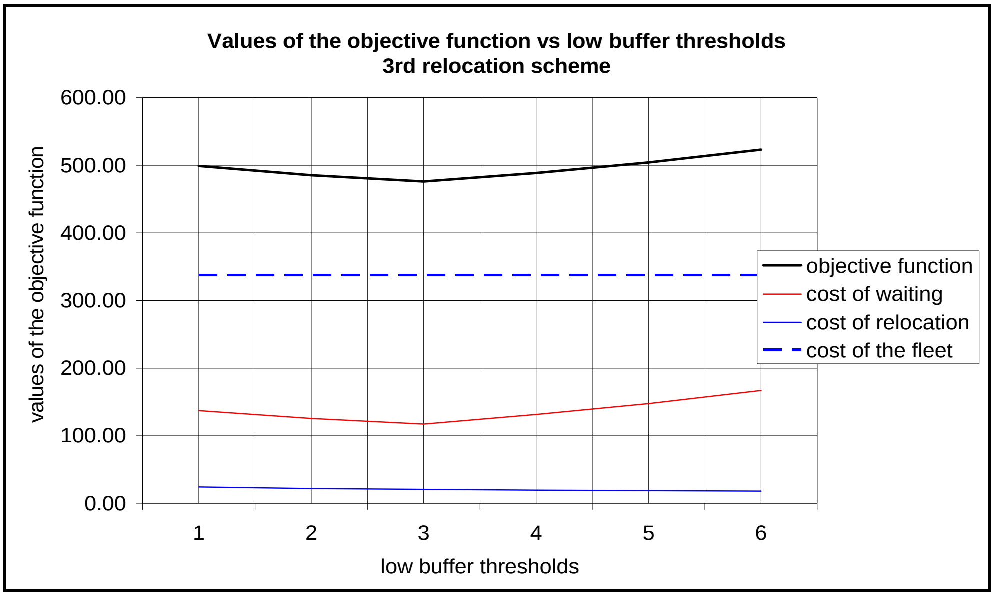


Figure 9.10. Values of the objective function with reference to the low buffer thresholds, for the third management strategy. The fleet dimension is fixed and equal to the optimized value. Low critical thresholds are also kept fixed and equal to their optimized values. Genoa case study.

9.2. Sensitivity analysis on the transport demand

9.2.1. Increase of all the transport demand by 30%.

9.2.1.1. Genoa case study

The total transport demand has been increased by 30% and the number of vehicles at each station by 2. The fleet has been increases as well in order to better distinguish the effects of the unbalancement of demand and the effects of a general increase of the transport demand. The system has shown to cope perfectly with this increase and to provide higher waiting times but not too high.
For the first system, the objective function provides a value of 634.07, with a total wait cost of 201.55 and a fleet cost of 432.52.
For the second system, the objective function assumes a value of 1121.71, with a wait cost of 710.45, a fleet cost of 391.35 and a relocation cost of 19.91.
For the third system, the objective function assumes a value of 1143.10, with a wait cost of 764.12, a fleet cost of 366.65 and a relocation cost of 12.33.
The LOS has changed from

9.2.1.2. Barreiro case study

Also in Barreiro, the demand has been increased by 30%, the fleet has been increased by 2 vehicles for each station, i.e. it became of 80 vehicles. The objective function assumes a value of 806.98, with a wait cost of 399.78 (average wait time of 2.22 minutes), a fleet cost of 374.19 € and a relocation cost of 57.02 €. As expected, Barreiro scenario showed similar results as Genoa case study.

9.2.2. Unbalancement of the transport demand.

9.2.2.1. Genoa case study.

The same demand, but the 8th row of all the demand OD matrixes has been increased by 30%, while the 8th column decreased by 30%.

For the first management strategy, the results are the following:

- 95th percentile of users' waiting times = 26.38
- 90th percentile = 21.38
- 50th percentile = 3.0
- total_wait_cost = 1361.59
- objective function = 46539.7804 with the contribution of the penalty function;
- objective function = 1757.06 without the contribution of the penalty function

For the other two management strategies in the Genoa case study:

Management strategy	objective function	cost of waiting	cost of relocation	cost of the fleet
Second management strategy	497.74	97.80	45.64	354.30
Third management strategy	496.18	121.10	37.25	337.83

With reference to the values previously obtained:

- for the second management strategy, the total users' waiting time has increased from 904 to 978 minutes, and the total relocation time from 331 to 456 minutes;
- for the third management strategy, the total users' waiting time has increased from 1174 to 1211 minutes, and the total relocation time from 331 to 372 minutes.

Therefore the unbalancement of demand is resolved by increasing the number of relocations and the user waiting times slightly increase because greater is the number of vehicles relocating.

9.2.2.2. Barreiro case study.

For what regards Barreiro scenario, the objective function assumes a value of 605.47, with a wait cost of 275.05 (average wait time of 2.12 minutes), a fleet cost of 284.08 € and a relocation cost of 27.50 €. As expected, Barreiro scenario shows similar results as Genoa case study.

Conclusions and future work

In the present thesis, a new generation car sharing system has been studied. This system is based on a fleet of intelligent vehicles which can be rented for short term periods (usually a couple of hours) and are shared through the day by different users. This system is meant to overcome the barriers of traditional car sharing systems, where vehicles need to be booked beforehand and must be returned at the same pick-up stations. In particular the system has been planned and modeled in order to guarantee open ended reservation, instant access and one way trips. These three features provide users a high level of flexibility. But on the other hand, the risk of unbalancement in the number of available vehicles at stations is high, therefore, relocation is necessary.

Three different kinds of relocation strategies have been proposed to keep distribution balance of parked PICAV vehicles among the area. The first relocation strategy adopts a user-based relocation scheme whilst the second and third ones adopt a fully vehicle based relocation strategy, because the level of automation of vehicles allows them to move in an automatic way. The manual relocation of vehicles among stations has not been taken into account because it registers severe staff costs. In the first and in the second management strategies vehicles can be accessed only at stations, while in the third one vehicles are available to users from any point of the intervention area. Regarding this third strategy, it is worth to underline that capillarity (i.e. the possibility that vehicles are available also along the roads) is a very good way to better satisfy user demand.

An object – oriented simulator has been developed in order to provide transport managers a useful tool to test the proposed transport systems in different realities. The simulator allows to analyse “what if” scenarios and aims to be a precious decisions support tool. The characteristics of the road network, the degree of congestion on the roads, the PICAV transport demand and the transport system parameters are the simulator inputs. The simulator follows each user and each vehicle within the simulation period, and gives the actual user waiting times and the number of vehicles available at each station in each simulation time instant, at the end of the simulation time period. The simulator gives in output the distribution of user waiting times and the total amount of time spent in relocation, from which the level of service and the transport system performance could be assessed.

These two quantities are the input of the optimization algorithm. An optimization algorithm has been developed in order to determine:

- the best fleet dimension and its distribution among stations at the beginning of the simulation, for what concerns the first management strategy;
- the best fleet dimension and the best low critical and low buffer thresholds values, for what regards the other two strategies.

The purpose is to minimize the users costs, expressed in terms of waiting times at the stations, and the system costs, expressed in term of cost of relocation and cost of purchasing the fleet. For what regards the last two management strategies, as the objective function is heavily sensitive on the fleet dimension, and slightly sensitive on the thresholds values, in order to obtain a better solution in a shorter amount of time, a parallel optimization of the fleet in one processor, and of the thresholds in the other processor, has been implemented. However, the Simulated Annealing is a heuristic algorithm and the solution determined is not the exact solution but a good solution. And the purpose of all the optimization procedure is to find a solution which can be close enough to the optimum solution.

The micro simulation model contains two sub models: the model of the battery and the model of interaction vehicles–pedestrians (vehicles speed–pedestrian density). The parameters of the simulator, as well as the parameters of these sub models, have been calibrated. The simulation model cannot be validated as no similar system exists in reality. However, the sub models have been validated and a test of coherence of the simulator's hypothesis has been performed.

The proposed transport system has been simulated and applied for the case study of the historical city centre of Genoa. The above mentioned output data, provided by the simulator, i.e. the level of service and the performance of the system, allow a comparison among the three strategies. The first management strategy, i.e. flexible users, shows the lowest waiting times. The third management strategy, in which vehicles are available also along the roads, registers the highest waiting times, but it also provides users with the highest level of satisfaction, being the vehicles accessible from any point of the intervention area.

The proposed transport system has also been applied to the old town of Barreiro, a suburb of Lisbon, Portugal. For Barreiro scenario, only the second strategy has been adopted. The results of the simulation clearly show the effectiveness of the automatic relocation, because, with low staff costs, it allows users a high level of satisfaction.

Finally, a sensitivity analysis has been performed, in order to study the behavior of the system if input data and parameters are changed. The sensitivity analysis has been performed for all the three management strategies and for both cases of study (Genoa and Barreiro). Firstly, the sensitivity of the system to the modifications of the fleet and of the thresholds has been performed. After, a sensitivity analysis has been performed on the demand has been done. The simulation shows that for this reality, the first management strategy (user based relocation) is very sensitive to demand unbalancement; the second strategy (automatic relocation) provides higher waiting times but it is able to cope with unbalanced demand.

Future work can be the following.

Firstly, it could be worthwhile to develop some indicators of users' satisfaction, and of proper integration between the PICAV transport system and the conventional public transport. More in detail, the disutility of users to be flexible, and the higher satisfaction of users because of the capillarity of the proposed system; and the seamless integration of car sharing and conventional transport system, could be quantified. These indicators could be developed following a fuzzy logic.

Secondly, several other aspects regarding the physical settlement of the system have been neglected. All these aspects have been already mentioned in chapter 2 and regard: membership, methodologies for check in and check out, methodologies for booking, techniques for the identification of the user, techniques for payment, pricing system, security, methods for the localization of vehicles, technologies for communication. A state of the art on existing technologies must be performed. After, the application of these techniques to the development of the proposed transport system must be studied.

Thirdly, a more careful study on transport demand is necessary in order to update the available data. The estimation of the transport demand by interviews is possible by two techniques: revealed preferences techniques, traditionally utilized, which are relative to the actual users travel behaviour in a real context, and stated preferences techniques, based on statements made by interviewees about their preferences in different choice contexts, real, hypothetical or experimental. Since we need to know the actual transport demand, we choose the revealed preferences technique.

REFERENCES:

Arpaweb Rimini, 2006. *Studio OMS – ARPAT effetti dell'inquinamento a lungo termine.* Available at:
http://www.arpa.emr.it/cms3/documenti/_cerca_doc/aria/rn_studio_oms_arpat_effetti_smog/rn_smog_traffico_arpat_2006.pdf

Astengo, G., 1966. *Urbanistica*. In: *Enciclopedia Universale dell'Arte*, vol. 14, Venezia, Sansoni, 1966.

Barth, M., Todd, M., 1999. *Simulation model performance analysis of a multiple station shared vehicle system*. Transportation Research Part C, Vol. 7, pp. 237-259, 1999.

Barth, M., Todd, M., Murakami, H., 2000. *Using Intelligent Transportation System Technology in a Shared Electric Vehicle Program*. Transportation Research Record 1731, 88-95.

Barth, M., Todd, M., 2003. *UCR IntelliShare: an intelligent shared electric vehicle testbed at the University of California, Riverside*. In: IATSS Research, 27 (1). June, 2003.

Barth, M., Todd, M., Xue, L., 2004. *User-Based Vehicle Relocation Techniques for Multiple-Station Shared-Use Vehicle Systems*. In: Proceedings of the 2004 Transportation Research Board Annual Meeting, Washington D.C., January 2004.

Barth, M., Shaheen, S., Fukuda, T., Fukuda, A., 2006. *Carsharing and station cars in Asia: an overview of Japan and Singapore*. Transportation Research Record No. 1986, pp. 106-115.

Bazaraa, M. S., Sherali, H. D., Shetty, C. M., 1993. *Nonlinear Programming: Theory and Algorithms*, Second Edition, John Wiley & Sons Inc., N.J., U.S.

Bellman, R., 1961. *Adaptive control processes: A guided tour*. Princeton, NJ: Princeton University Press.

Bonfanti, M. *Gestione di una flotta di veicoli di un sistema car-sharing: applicazione al centro urbano di Genova*. Master thesis, Department of Civil Engineering, University of Pisa, 2010.

Britton, E., 1999. *Carsharing 2000 – A Hammer For Sustainable Development*, Journal of World Transport Policy & Practice, Vol. 5, n.
3, pp. 9-15.

Castangia M. and Guala L., 2011. *Modelling and simulation of PRT networks*. In: Pratelli, A., Brebbia, C.A. (Eds.), *Urban Transport XVII*. WIT Press, Southampton, UK, pp. 459-472.

Ceccarelli, G., 2010. *Il bike sharing in Italia. Report 2010*. Available at:
http://www.uomoplanetario.org/wordpress/2010/03/il-bike-sharing-in-italia-report-2010/

Cepolina E. M., De Luca S., Tyler N. (2008). *Un modello microscopico per la simulazione del deflusso pedonale in corridoi di stazioni ferroviarie*. In: Proceedings of the XIV SIDT Scientific Seminar. Franco Angeli, Milano.

Cepolina, E. M., 2009. *D1.1. PICAV System Requirements.* PICAV internal report. Available at: http://www.dimec.unige.it/PMAR/picav/

Cepolina, E. M., 2010. *D2.2. Report on system conceptual design and operative modes*. PICAV internal report. Available at: http://www.dimec.unige.it/PMAR/picav/

Cepolina, E.M. *Modelling and Simulation of the full electric personal vehicle PICAV*. Submitted for publication to the European Transport Research Review: An Open Access Journal, 2012.

Cherchi, E., 2003. *Il valore del tempo nella valutazione dei sistemi di trasporti: teoria e pratica*. Franco Angeli, Milano, Italy.

Ciari F., Balmer, M., Axhausen, K.W., 2009. *Concepts for a large scale car-sharing system: Modelling and evaluation with an agent-based approach*. Proceedings of the 88th Annual Meeting of the Transportation Research Board, Washington, D.C., January 2009.

CityCarShare, 2007. *Bringing Carsharing to your Community*. Available at: http://www.communauto.com/images/03.coupures_de_presse/CCS_BCCtYC_Long.pdf

Crainic, T.G., Toulouse, M., 2009. *Parallel Meta-heuristics*. In: Gendreau, M., Potvin, J.Y. (Eds.), *Handbook of Metaheuristics*. Springer, Second Edition, pp. 497-541.

DIME, 2012. *PICAV final report*. Available at: http://www.dimec.unige.it/pmar/picav/

Ferreira, N., 2009. *D1.1 Appendix E. Study of Circulation, Transports and Parking for the Centre of Barreiro*. PICAV internal report. Available from: http://www.dimec.unige.it/pmar/picav/

Firnkorn, J., Müller, M. 2011. *What will be the environmental effects of new free-floating car-sharing systems? The case of car2go in Ulm*. In: Ecological Economics, 70 (8), pp. 1519-1528.

Goldberg, 1989. *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison-Wesley, Reading, MA

Highway Capacity Manual (HCM), 2004.

ISTAT, 2012. *Road accidents*. Available at: http://www.istat.it/en/archive/73773

Kek, A. G. H., Cheu, R. L., Chor, M. L., 2006. *Relocation Simulation Model for Multiple-Station Shared-Use Vehicle Systems*. In: Transportation Research Record: Journal of the Transportation Research Board, 1986 (13), pp. 81-88, 2006.

Laarhoven, P.J., Aarts, E.H., 1987. *Simulated Annealing: Theory and Application.* Kluwer Academic Publishers, Dordrecht, Netherlands.

Law, A.M., Kelton, W.D., 1991. *Simulation modelling and analysis.* McGraw-Hill, New York, Second Edition.

Lewis, J., Loftus, W., 2008. *Java Software Solutions Foundations of Programming Design.* 6th edition. Pearson Education Inc., section 1.6 "Object-Oriented Programming".

Lindholm, M., Behrends, S., 2010. *A holistic approach to challenges in urban freight transport planning*. In: Selected Proceedings of 12th World Conference on Transportation Research-WCTR 2010, Viegas, J.M. and Macario, R. (eds.), Lisbon, Portugal.

Masood, J., Zoppi, M., Molfino, R.M., 2001. *Multi-Terrain Vehicle Active Suspension Control Design and Synthesis*. In: Proceedings of The 7th International ASME/IEEE Conference on Mechatronics & Embedded Systems and Applications, MESA 2011, Washington DC.

MAZEL Ingenerios, (Moreno, M.G., Dominguez, J.A.), 2011. *D3.7, Power system and electric layout description.* PICAV internal report. Available from http://www.dimec.unige.it/PMAR/picav/

Millard-Ball, A., Murray, G., Ter Schure, J., Fox, C., and Burkhardt, J., 2005. *Car-Sharing: Where and How it Succeeds. TCRP Report 108*. Transportation Research Board, Washington, DC. Available from: http://www.trb.org/Main/Blurbs/156496.aspx

Monaci, M., 2012. *Algoritmi euristici*. Dipartimento di Ingegneria dell'Informazione, Università di Padova http://www.dei.unipd.it/~monaci/euristici_rev21.pdf

Nguyen, L., Barth, M., 2006. *Improving Automatic Vehicle Location Efficiency through Aperiodic Filtering.* In: Proceedings of the IEEE ITSC 2006, 2006 IEEE Intelligent Transportation Systems Conference, Toronto, September 17-20, 2006.

Schwieger, B., 2003. *International Developments towards a "Second Generation" Car-Sharing*, Ph.D. Dissertation, TU Berlin, Berlin.

Shaheen, S.A., Sperling, D., Wagner, C., 1998. *Carsharing in Europe and North America: Past, Present, and Future*. In: Transportation Quarterly, 52 (3), pp. 35 -52.

Shaheen, S.A., Rodier, C. J., 2005. *Travel Effects of A Suburban Commuter-Carsharing Service: A CarLink Case Study*. In: Transportation Research Record: Journal of the Transportation Research Board. TRB, National Research Council.

Shaheen, S. A., Cohen, A.P., Roberts, J.D., 2005. *Carsharing in North America: Market Growth, Current Developments, and Future Potential*. In Transportation Research Record: Journal of the Transportation Research Board, No. 1986, pp. 116–124.

Shaheen, S. A., Cohen, A. P., 2007. *Growth in worldwide carsharing – An international comparison*. Transportation Research Record 1992, 81-89.

Shaheen, S., Cohen, A., Chung., M., 2009. *North American Carsharing: A Ten-Year Retrospective*. In Transportation Research Record: Journal of the Transportation Research Board, Washington, D.C., 2110, pp. 35-44.

Shaheen, S., Guzman, S., Zhang, H., 2010. *Bikesharing in Europe, the Americas, an Asia: Past, Present, and Future*. In: Transportation Research Record. 2010 Transportation Research Board Annual Meeting. March 15, 2010.

Shaheen, S., Guzman, S., 2011. *Worldwide Bikesharing*. Transportation Sustainability Research Center, University of California. Berkeley. Available at: http://tsrc.berkeley.edu/worldwide%20bikesharing

Transport For London, 2008. *Feasibility study for a central London cycle hire scheme*. Available at:
http://www.tfl.gov.uk/assets/downloads/businessandpartners/cycle-hire-scheme-feasibility-full-report-nov2008.pdf

Van Rossum, G., 2006. *Python Tutorial. Release 2.5*. [online]. Available at: http://docs.python.org/release/2.5/tut/tut.html

Access-to-all: http://www.access-to-all.eu/ (Last accessed: October 2012)

Aeneas: http://www.aeneas-project.eu/?page=home (Last accessed: October 2012)

Ask-it: http://www.ask-it.org/ (Last accessed: October 2012)

Autolib': https://www.autolib.eu/stations/ (Last accessed: January 2013)

Bikemi: www.bikemi.com (Last accessed: January 2013)

Car2go: www.car2go.com (Last accessed: January 2013)

CityCarClub: http://www.citycarclub.co.uk/ (Last accessed: December 2012)

CityCarShare.org: https://www.citycarshare.org/ (Last accessed: December 2012)

Cultural heritages in Bologna: http://www.archeobo.arti.beniculturali.it (Last accessed: December 2012)

Drivenow: www.drive-now.com (Last accessed: December 2012)

Easymotion.pl: http://easymotion.pl/

European Demographic Data Sheet 2012. Re-evaluating population ageing in European countries. Available at:

http://www.oeaw.ac.at/vid/datasheet/index.html (Last accessed: Novemer 2012)

Genoa public transport: http://www.amt.genova.it/ (Last accessed: November 2012)

Greenwheels: www.greenwheels.nl (Last accessed: January 2013)

Ibilek: http://www.ibilek.es/es/ (Last accessed: January 2013)

Iniziativa car-sharing: http://www.icscarsharing.it/main/ (Last accessed: January 2013)

Mediate: http://www.mediate-project.eu/ (Last accessed: October 2012)

PICAV project: http://www.dimec.unige.it/pmar/picav/ (Last accessed: January 2013)

Respiro: http://www.respiromadrid.es/ (Last accessed: January 2013)

Stadtmobil: http://www.stadtmobil.de/ (Last accessed: January 2013)

Transport For London: www.tfl.gov.uk (Last accessed: January 2013)
Bike sharing is available at: http://www.tfl.gov.uk/roadusers/cycling/14808.aspx
Road pricing is available at: http://www.tfl.gov.uk/roadusers/default.aspx

Wikipedia. Car sharing: http://en.wikipedia.org/wiki/Carsharing (Last accessed December 2012)

Zipcar website: http://www.zipcar.com/ (Last accessed: January 2013)

ARTICLES WRITTEN BY THE AUTHOR ABOUT THE TOPICS REPORTED IN THE THESIS:

1. Cepolina E. M., Bonfanti M., Farina A., 2010. *A new transport system for reducing external costs in historical city centres*. SIDT Scientific seminar 2010, External costs of transport systems, Theory and application, Roma.

2. Cepolina E.M., Farina A., Holloway C., 2011. *A car sharing system for urban areas with fully automated personal vehicles*. The 13th International Conference on Harbor, Maritime & Multimodal Logistics Modelling and Simulation", HMS 2011, Roma, 12-14 settembre 2011.

3. Cepolina E.M, Tyler N., Holloway C., Farina A., Boussard C., *Modelling of urban pedestrian environments for simulation of the motion of small vehicles*. The 13th International Conference on Harbor, Maritime & Multimodal Logistics Modelling and Simulation", HMS 2011, Roma, 12-14 settembre 2011.

4. Cepolina E.M., Farina A., 2012. *A new shared vehicle system for urban areas*. Transportation Research part C: Emerging Technologies, 21(1), pp 230-243.

5. Cepolina E.M., Farina A., 2012. *Urban car sharing: an overview of relocation strategies.* In: Longhurst, J.W.S., Brebbia, C.A. (Eds.), Urban Transport XVII. WIT Press, Southampton, UK, pp. 419-431.

Papers submitted or under publication:

1. Cepolina, E.M., Farina, A. *Innovative strategies for urban car-sharing systems and a simulator to assess their performances*. Sent for publication to Transportation Research Board

2. Cepolina, E.M., Farina, A. *The optimization of a new urban car sharing system with fully automated personal vehicles*. Draft.

Appendix A: Code of the micro simulator

Below, the simulator and its input files are provided.

The file “inputdata.py”

```
import csv
reader = csv.reader(open('input.csv','rb'),delimiter = ',')
writer = csv.writer(open('input.csv','rb'),delimiter = ',')
j = 0
for row in reader:
    if j == 0:
        case_study = row
    if j == 1:
        Tmax = row
    if j == 2:
        TI1_min = row
    if j == 3:
        TI1_max = row
    if j == 4:
        TI2_max = row
    if j == 5:
        dens_morn = row
    if j == 6:
        dens_aft = row
    if j == 7:
       dens_even = row
    if j == 8:
        number_picav_beginning = row
    if j == 9:
        low_critical_threshold = row
    if j == 10:
        low_buffer_threshold = row
    if j == 11:
        high_critical_threshold = row
    if j == 12:
        high_buffer_threshold = row
    if j == 13:
        stations_capacity = row
    if j == 14:
        number_stations = row
    if j == 15:
        number_unit = row
    if j == 16:
        coeff_activity_pattern = row
    if j == 17:
        mean_activity_duration = row
    if j == 18:
        deviation_activity_duration = row
    if j == 19:
        scenario_under_study = row # 1 = Genoa, 2 = Barreiro
    j = j+1


case_study = int(case_study[0])
Tmax = int(Tmax[0])
TI1_min = int(TI1_min[0])
TI1_max = int(TI1_max[0])
TI2_max = int(TI2_max[0])
dens_morn = float(dens_morn[0])
dens_aft = float(dens_aft[0])
dens_even = float(dens_even[0])
number_stations = int(number_stations[0])
number_unit = int(number_unit[0])
coeff_activity_pattern = float(coeff_activity_pattern[0])
mean_activity_duration = float(mean_activity_duration[0])
deviation_activity_duration = float(deviation_activity_duration[0])
scenario_under_study = int(scenario_under_study[0])
number_hours_morning = TI1_max - TI1_min
number_hours_afternoon = TI2_max - TI1_max
TI0_min = 0
TI1_min = TI1_min * 60
TI0_max = TI1_min - 1
TI1_max = TI1_max * 60
TI2_min = TI1_max + 1
TI2_max = TI2_max * 60
TI3_min = TI2_max + 1
Tmax = Tmax * 60
TI3_max = Tmax
```

```
NPP = []
for j in range(len(number_picav_beginning)):
    NPP.append(int(number_picav_beginning[j]))

low_critical_thresholds = []
for j in range(len(low_critical_threshold)):
    low_critical_thresholds.append(int(low_critical_threshold[j]))

high_critical_thresholds = []
for j in range(len(high_critical_threshold)):
    high_critical_thresholds.append(int(high_critical_threshold[j]))

low_buffer_thresholds = []
for j in range(len(low_buffer_threshold)):
    low_buffer_thresholds.append(int(low_buffer_threshold[j]))

high_buffer_thresholds = []
for j in range(len(high_buffer_threshold)):
    high_buffer_thresholds.append(int(high_buffer_threshold[j]))

station_capacity = []
for j in range(len(stations_capacity)):
    station_capacity.append(int(stations_capacity[j]))


number_picav = 0
for j in range(number_stations):
    number_picav = number_picav + NPP[j]
```

The file “OD_matrix.py”

```
from inputdata import *

OD_A_1= [[0,1,2,1,1,1,0],
        [1,0,0,0,1,1,0],
        [1,0,0,0,2,1,0],
        [1,0,0,0,1,1,0],
        [1,1,2,2,0,1,0],
        [1,1,2,1,1,0,0],
        [0,0,1,1,0,0,0]]

OD_B_1= [[0,1,1,0,0,1,0,1,1],
        [0,0,0,0,0,1,0,1,1],
        [1,0,0,0,1,3,1,3,2],
        [1,0,0,0,0,1,1,2,2],
        [0,1,1,0,0,1,0,2,1],
        [0,1,1,0,0,0,0,2,1],
        [0,0,0,0,0,0,0,1,0]]

OD_D_1 = []
OD_E_1 = []

OD_A_2 = []
OD_B_2 = []
OD_D_2 = []
OD_E_2 = []

factor_afternoon = 1.45
# say factor afternoon = 0 if you don't declare it. In this case, define OD_A_2, etc.
explicitly

if case_study == 3:
    from demand_units import *

if case_study == 2:
    OD_A_choiceset1 = [[1,1,1],
                  [1,1,1],
                  [3,2,1],
                  [3,2,1],
                  [2,1,1],
                  [2,1,1],
                  [1,1,1]]

    OD_B_choiceset1 = [[1,1,0],
                  [1,1,1],
                  [1,1,1],
                  [0,0,0],
                  [1,0,0],
                  [3,2,1],
                [1,1,0],
                [6,4,3],
                [4,2,2]]

    OD_A_1 = OD_A_choiceset1
```

The file “distances_slopes.py”

```
distances_stations = [[0,1060,976,1200,1400,604,1100,489,1060],
                [1060,0,316,846,1180,732,1100,489,747],
                [976,316,0,414,516,569,829,563,326],
                [1200,846,414,0,481,848,1080,866,379],
                [1400,1180,516,481,0,834,804,1050,372],
                [604,732,569,848,834,0,678,458,464],
                [1100,1100,829,1080,804,678,0,961,610],
                [489,489,563,866,1050,458,961,0,670],
                [1060,747,326,379,372,464,610,670,0]]

slopes_stations = [[0,2.28,1.95,2.17,0.23,-0.60,-0.17,4.95,1.95],
              [-2.28,0,-1.03,0.81,-0.05,-2.16,-0.69,1.36,-0.28],
              [-1.95,1.03,0,2.16,0.50,-2.75,-0.72,2.08,-0.38],
              [-2.17,-0.81,-2.16,0,-0.42,-3.96,-2.49,-0.42,-1.35],
              [-0.23,0.05,-0.50,0.42,0,-2.69,-2.97,-0.35,-0.42],
              [0.60,2.16,2.75,3.96,2.69,0,-2.68,5.04,3.26],
              [0.17,0.69,0.72,2.49,2.97,2.68,0,2.65,3.21],
              [-4.95,-1.36,-2.08,0.42,0.35,-5.04,-2.65,0,-0.93],
              [-1.95,0.28,0.38,1.35,0.42,-3.26,-3.21,0.93,0]]


distances_points_points = []
slopes_points_points = []
distances_points_stations = []
distances_stations_points = []
slopes_points_stations = []
slopes_stations_points = []

from distances_slopes_units import *
```

The file input_manipulation.py

```
# -*- coding: cp1252 -*-

from numpy import *
from math import *


# -----------------  DEMAND PART -------------------
# Do not act here, do not touch this code

afternoon_factor = 0
from OD_matrix import *
# The OD matrix for the four types of trips is given
# Type A trip: trip chain inside the intervention area, origin and destination on the border
# Type B trip: single mission trip with origin on the border and destination inside the
intervention area
# Type C trips: does not exist anymore, it was the return trip of type B trip, which is
assumed for hypothesis the opposite of type B.
# Type D trip: single mission trip with origin and destination at units
# Type E trip: trip chain with origin and destination at units
# Case study 1- 2: only type A + type B trips
# Case study 3: all

# The afternoon factor is given, i.e. the factor which allows to calculate the afternoon
OD_matrixes directly without giving it in advance

# We consider in the border of the intervention area that in case study 3 the trips have
origin and end at the stations,
# while inside the intervention area they have origin and end at some parking lots


class demands:
    name = 999999
    origin = 999999
    destination = 999999
    typology = 999999
    activity_pattern = 999999

demand = []
dom_1 = {}
```

```
c = 0


# Takes the matrixes A,B,D,E for the morning period and converts them into objects. Puts
the objects in a dictionary, dom_1
# in order that they can be easily taken from the simulator. Also the return trips of Type B
will be defined

for k in range(len(OD_A_1)):
   PP = []
   PP = OD_A_1[k]
   for j in range(len(PP)):
      if PP[j]>0:
          counter = PP[j] +1
          counter = counter - 1
          while counter > 0:
            d = demands()
            d.name = c
            d.origin = k+1
            d.destination = j+1
            d.typology = 1
            if case_study == 3:
               d.typology_origin = 1
               d.typology_destination = 1
            if case_study == 2:
               d.origin = k+1
               d.provenience = j+1
               d.destination = 999999
            d.activity_pattern = 1
            dom_1[c] = d
            c = c+1
            counter = counter - 1

for k in range(len(OD_B_1)):
   PP = []
   PP = OD_B_1[k]
   for j in range(len(PP)):
      if PP[j]>0:
         counter = PP[j]+1
         counter = counter - 1
         while counter > 0:
            d = demands()
            d.name = c
            d.origin = k+1
            d.destination = j+1
            d.typology = 1
            if case_study == 3:
               d.typology_destination = 2
               d.typology_origin = 1
               d.destination = j
            if case_study == 2:
               d.provenience = 999999
            d.activity_pattern = 2
            dom_1[c] = d
            c = c+1
            counter = counter - 1

OD_B_1_provv = OD_B_1
if case_study == 2:
   OD_B_1 = OD_B_choiceset1

for k in range(len(OD_B_1)):
   PP = []
   PP = OD_B_1[k]
   for j in range(len(PP)):
      if PP[j]>0:
         counter = PP[j]+1
         counter = counter - 1
         while counter > 0:
            d = demands()
            d.name = c
            d.origin = j+1
            d.destination = k+1
            d.typology = 2
            if case_study == 3:
               d.typology_origin = 2
               d.typology_destination = 1
               d.origin = j
            if case_study == 2:
               d.origin = k+1
               d.provenience = j+1
               d.destination = 999999
            d.activity_pattern = 2
            dom_1[c] = d
            c = c+1
            counter = counter - 1

for k in range(len(OD_D_1)):
   PP = []
   PP = OD_D_1[k]
   for j in range(len(PP)):
      if PP[j]>0:
         counter = PP[j]+1
         counter = counter - 1
         while counter > 0:
            d = demands()
            d.name = c
            d.origin = k
            d.destination = j
            d.typology = 3
            d.typology_origin = 2
            d.typology_destination = 2
            d.activity_pattern = 2
            dom_1[c] = d
            c = c+1
            counter = counter - 1

for k in range(len(OD_E_1)):
   PP = []
   PP = OD_E_1[k]
   for j in range(len(PP)):
      if PP[j]>0:
         counter = PP[j]+1
         counter = counter - 1
         while counter > 0:
            d = demands()
            d.name = c
            d.origin = k
            d.destination = j
            d.typology = 3
            d.typology_origin = 2
            d.typology_destination = 2
            d.activity_pattern = 1
            dom_1[c] = d
            c = c+1
            counter = counter - 1

# Determines total number of arrivals in 1 morning hour
# The number of arrivas is set equal to the length of the dictionary
number_users_hour = len(dom_1)

# creates some auxiliary quantities for calculating the morning demand

nor_morning = []

for j in range(number_hours_morning):
    nor_morning.append(number_users_hour*(j+1))

number_users_morning = number_users_hour * number_hours_morning

# Appends the dictionary nor_morning in the demand list.
# Creates the demand for the morning period
for j in range(len(nor_morning)):
    for k in range(number_users_hour):
        demand.append(dom_1[k])


# Calculates the afternoon matrixes in case the afternoon factor is given.

OD_B_1 = OD_B_1_provv
if case_study == 2:
    OD_B_choiceset2 = []

if factor_afternoon > 0:
    for i in range(len(OD_B_1)):
        BET = OD_B_1[i]
        for k in range(len(BET)):
            BET[k] = round(BET[k] * factor_afternoon)
        OD_B_2.append(BET)
        BET = []

    if case_study == 2:
        for i in range(len(OD_B_choiceset1)):
            BET = OD_B_choiceset1[i]
            for k in range(len(BET)):
                BET[k] = round(BET[k] * factor_afternoon)
            OD_B_choiceset2.append(BET)
            BET = []


    for i in range(len(OD_A_1)):
        BET = OD_A_1[i]
        for k in range(len(BET)):
            BET[k] = round(BET[k] * factor_afternoon)
        OD_A_2.append(BET)
        BET = []


    for i in range(len(OD_D_1)):
        BET = OD_D_1[i]
        for k in range(len(BET)):
            BET[k] = round(BET[k] * factor_afternoon)
        OD_D_2.append(BET)
        BET = []

    for i in range(len(OD_E_1)):
        BET = OD_E_1[i]
        for k in range(len(BET)):
            BET[k] = round(BET[k] * factor_afternoon)
        OD_E_2.append(BET)
        BET = []

# Takes the matrixes A,B,C,D,E for the afternoon period and converts them into objects.
Puts the objects in a dictionary, dom_1
# Also the return trips of Type B and Type C will be defined

dom_2 = {}
c = 0

for k in range(len(OD_A_2)):
    PP = []
    PP = OD_A_2[k]
    for j in range(len(PP)):
        if PP[j]>0:
            counter = PP[j]+1
```

```
         counter = counter - 1
         while counter > 0:
            d = demands()
            d.name = c
            d.origin = k+1
            d.destination = j+1
            d.typology = 1
            if case_study == 3:
               d.typology_origin = 1
               d.typology_destination = 1
            if case_study == 2:
               d.origin = k+1
               d.provenience = j+1
               d.destination = 999999
            d.activity_pattern = 1
            dom_2[c] = d
            c = c+1
            counter = counter - 1


for k in range(len(OD_B_2)):
    PP = []
    PP = OD_B_2[k]
    for j in range(len(PP)):
        if PP[j] > 0:
            counter = PP[j]+1
            counter = counter - 1
            while counter > 0:
               d = demands()
               d.name = c
               d.origin = k+1
               d.destination = j+1
               d.typology = 1
               if case_study == 3:
                  d.typology_origin = 1
                  d.typology_destination = 2
                  d.destination = j
               if case_study == 2:
                  d.provenience = 999999
               d.activity_pattern = 2
               dom_2[c] = d
```

```
                c = c+1
                counter = counter - 1

if case_study == 2:
    OD_B_2 = OD_B_choiceset2


for k in range(len(OD_B_2)):
    PP = []
    PP = OD_B_2[k]
    for j in range(len(PP)):
        if PP[j] > 1:
            counter = PP[j]+1
            counter = counter - 1
            while counter > 0:
                d = demands()
                d.name = c
                d.origin = j+1
                d.destination = k+1
                d.typology = 2
                if case_study == 3:
                    d.typology_origin = 2
                    d.typology_destination = 1
                    d.origin = j
                if case_study == 2:
                    d.origin = k+1
                    d.provenience = j+1
                    d.destination = 999999
                d.activity_pattern = 2
                dom_2[c] = d
                c = c+1
                counter = counter - 1


for k in range(len(OD_D_2)):
    PP = []
    PP = OD_D_2[k]
    for j in range(len(PP)):
        if PP[j] > 1:
            counter = PP[j]
            counter = counter - 1
```

```
                while counter > 0:
                    d = demands()
                    d.name = c
                    d.origin = k
                    d.destination = j
                    d.typology = 3
                    d.typology_origin = 2
                    d.typology_destination = 2
                    d.activity_pattern = 2
                    dom_2[c] = d
                    c = c+1
                    counter = counter - 1

for k in range(len(OD_E_2)):
    PP = []
    PP = OD_E_2[k]
    for j in range(len(PP)):
        if PP[j]>0:
            d = demands()
            d.name = c
            d.origin = k
            d.destination = j
            d.typology = 3
            d.typology_origin = 2
            d.typology_destination = 2
            d.activity_pattern = 1
            dom_2[c] = d
            c = c+1
            if PP[j] > 1:
                counter = PP[j]
                counter = counter - 1
                while counter > 0:
                    d = demands()
                    d.name = c
                    d.origin = k
                    d.destination = j
                    d.typology = 3
                    d.typology_origin = 2
                    d.typology_destination = 2
                    d.activity_pattern = 1
                    dom_2[c] = d
                c = c+1
                counter = counter - 1


# Determines the number of arrivals by hour in the afternoon
number_users_hour = len(dom_2)
nor_afternoon = [] # dictionary of the arrivals by hour

for j in range(number_hours_afternoon):
    nor_afternoon.append(number_users_hour*(j+1))

number_users_afternoon = number_users_hour * number_hours_afternoon
number_users = number_users_morning + number_users_afternoon

for j in range(len(nor_afternoon)):
    for k in range(number_users_hour):
        demand.append(dom_2[k])

for j in range(len(nor_afternoon)):
    nor_afternoon[j] = nor_afternoon[j]+ number_users_morning


# CALCULATES THE ATTRACTIVITY OF UNITS IN ORDER TO DETERMINE THE
UNIT IN
# WHICH THE SUPERVISOR IS CALLED
from attractivity_units import *
total_att = 0
for j in range(number_units):
    units[j].cumul_att = units[j].attr + total_att
    total_att = units[j].cumul_att


  # --------------- DISTANCES AND SLOPES PART ------------------------------

class distance:
    origin = 999999
    destination = 999999
    value = 999999

from distances_slopes import *
```

```
#initialization of all dictionaries

dist_pls_pls = {}
pend_pls_pls = {}
dist_points_pls = {}
pend_points_pls = {}
dist_pls_points = {}
pend_pls_points = {}
dist_points_points = {}
pend_points_points = {}

# distances and slopes dictionaries creation. Case considered:
# stations - stations, stations - units, units - stations, units - units

c = 0
for k in range(len(distances_stations)):
    PP = []
    PP = distances_stations[k]
    for j in range(len(PP)):
        d = distance()
        d.origin = k+1
        d.destination = j+1
        d.value = PP[j]
        dist_pls_pls[c] = d
        c = c + 1

w = 0
for k in range(len(slopes_stations)):
    RR = []
    RR = slopes_stations[k]
    for j in range(len(RR)):
        s = distance()
        s.origin = k+1
        s.destination = j+1
        s.value = RR[j]
        pend_pls_pls[w] = s
        w = w + 1

if case_study == 1:
    distances_points_stations = {}
    distances_stations_points = {}
    distances_points_points = {}
    slopes_points_stations = {}
    slopes_stations_points = {}
    slopes_points_points = {}

c = 0
for k in range(len(distances_points_stations)):
    PQ = []
    PQ = distances_points_stations[k]
    for j in range(len(PQ)):
        d = distance()
        d.origin = k
        d.destination = j+1
        d.value = PQ[j]
        dist_points_pls[c] = d
        c = c + 1

c = 0
for k in range(len(slopes_points_stations)):
    PQ = []
    PQ = slopes_points_stations[k]
    for j in range(len(PQ)):
        d = distance()
        d.origin = k
        d.destination = j+1
        d.value = PQ[j]
        pend_points_pls[c] = d
        c = c + 1

c = 0
for k in range(len(distances_stations_points)):
    PQ = []
    PQ = distances_stations_points[k]
    for j in range(len(PQ)):
        d = distance()
        d.origin = k+1
        d.destination = j
        d.value = PQ[j]
        dist_pls_points[c] = d
        c = c + 1
```

```
c = 0
for k in range(len(slopes_stations_points)):
    PQ = []
    PQ = slopes_stations_points[k]
    for j in range(len(PQ)):
        d = distance()
        d.origin = k+1
        d.destination = j
        d.value = PQ[j]
        pend_pls_points[c] = d
        c = c + 1

c = 0
for k in range(len(distances_points_points)):
    PQ = []
    PQ = distances_points_points[k]
    for j in range(len(PQ)):
        d = distance()
        d.origin = k
        d.destination = j
        d.value = PQ[j]
        dist_points_points[c] = d
        c = c + 1

c = 0
for k in range(len(slopes_points_points)):
    PQ = []
    PQ = slopes_points_points[k]
    for j in range(len(PQ)):
        d = distance()
        d.origin = k
        d.destination = j
        d.value = PQ[j]
        pend_points_points[c] = d
        c = c + 1

dom_1[j].name,dom_1[j].origin,dom_1[j].destination,dom_1[j].typology_origin,dom_1[j].t
ypology_destination,dom_1[j].typology,dom_1[j].activity_pattern
```

The file “simulatore.py”

```
# -*- coding: cp1252 -*-
from numpy import *
from numpy.random import *
from numpy.oldnumeric.random_array import *
from math import *
from input_manipulation import *


class simulator:

#------------------ INITIALIZATION -------------------

    time_queue = []
    max_time_queue = []
    users = {}
    vehicles = {}
    stations = {}
    unit = {}
    counter_vehicles = {} # takes memory of vehicles
    queue = [] # queue of the stations: it is a list made of some sub-lists, one for each station
    queue_unit = [] # queue of users at units
    users_rejected = 0
    number_stations = 0
    number_users_queue = []
    time_queue = []
    max_time_queue = []
    number_picav_state1 = [] # number of vehicles in state 1 = occupied
    number_picav_state2 = [] # number of vehicles in state 2 = in charge
    number_picav_state3 = [] # number of vehicles in state 3 = relocating
    number_picav_state4 = [] # number of vehicles in state 4 = redirected if the station is full
    number_picav_state5 = [] # number of vehicles in state 5 = approaching the user
    number_picav_state0 = [] # number of vehicles in state 0 = available for all
    number_total_relocations = [] # number of total relocations
    number_impossible_relocations = [] # number of impossible relocations
    number_vehicles_available = [] # number of available vehicles in p.l.s
    time_stayed_queue = [] # auxiliary list for statistics
    coeff_activity_pattern = 1
    total_relocation_time = 0
    supervis = []
    max_waiting_time = []
    counter_veh_recharging = []

    def simulation(self):

#-------------- INITIALIZATION OF ALL THE PARAMETERS -----------------

    # and importation of all the parameters from the input data to the simulator method.

        self.case_study = case_study
        self.supervis = []
        self.number_unit = number_unit
        self.number_users = number_users
        if scenario_under_study == 1:
            self.choice_sets = choice_sets
        self.number_simulated_users = [] #number of users in simulation
        self.time_queue = [] #time in the queue
        self.max_time_queue = [] #max time in the queue
        self.users = {} # dictionary of the users
        self.vehicles = {} # dictionary of vehicles
        self.stations = {} # dictionary of stations
        self.counter_vehicles = {} # counter of vehicles
        self.queue = [] # queues at stations
        self.users_rejected = 0 # users are rejected when the system closes
        self.number_stations = number_stations
        self.number_simulated_users = [] # number of users in simulation in each p.l.
        self.time_queue = [] # time in the queue for each p.l.
        self.max_time_queue = [] # max time in the queue for each p.l.
        self.number_picav_state1 = [] # number of vehicles in state 1
        self.number_picav_state2 = []
        self.number_picav_state3 = []
        self.number_picav_state0 = []
        self.number_picav_state4 = []
        self.number_picav_state5 = []
        # states of vehicles are the following: 1 = occupied, 0 = free, 2 = in charge, 3 =
relocating between stations,
        # 4 = redirected if the station destination is full, 5 = goes encounter the user at the
user's unit
        self.number_vehicles_available = [] # number of available vehicles in p.l.s
        self.time_stayed_queue = []
        self.unit = {}
        self.NPP = NPP # number of picavs in each parking lot at the beginning of the
simulation
        self.low_critical_thresholds = low_critical_thresholds
        self.low_buffer_thresholds = low_buffer_thresholds
        self.high_critical_thresholds = high_critical_thresholds
        self.high_buffer_thresholds = high_buffer_thresholds
        if case_study != 2: # if case study = 2 there is the call of the supervisor before the last
trip
            self.coeff_activity_pattern = 5
        elif self.case_study == 2:
            self.coeff_activity_pattern = 4
        self.number_stations = number_stations

        self.total_relocation_time = 0
        self.number_relocations = 0
        self.number_total_relocations = [] # number of total relocations
        self.number_impossible_relocations= [] # number of impossible relocations

        # the following are some stathistical dictionaries, initialized for the online or offline
statistics
        for j in range(number_stations):
            self.time_queue.append(0)
            self.max_time_queue.append([])
            self.number_simulated_users.append(0)
            self.number_vehicles_available.append([])
            self.counter_veh_recharging.append([])
            self.number_users_queue.append([])


        # Parameter of the model: definition of the number of picavs for every parking
        self.NPP = NPP

        # ------------------ Calculations ------------------------

        # calculation of the number of PICAVs
        number_picav = 0
        for j in range(number_stations):
            number_picav = number_picav + NPP[j]
        self.number_picav = number_picav
```

```
# ----------- RELATION SPEED DENSITY ---------------

        B = 1.57751
        A = -1.44677
        B_autom = 1.37751
        self.vel_morning = A * dens_morn + B
        self.vel_aftern = A * dens_aft + B
        self.vel_autom_morn = A * dens_morn + B_autom
        self.vel_autom_aft = A * dens_aft + B_autom
        self.vel_evening = A * dens_even + B
        self.vel_autom_even = A * dens_even + B_autom


# -------------- main -----------------

        # creates the permanent objects parkings

        for k in range(number_stations):
            p = parking()
            p.name = k+1
            p.n_vehicles = self.NPP[k] # defines the number of vehicles in each station
            p.n_veh_relocation = self.NPP[k] # n veh relocation is the number of vehicles
assigned to each station.
            # This last indicator avoids that a station in defect of vehicles continuously recalls
vehicles until its
            # number of vehicles is arise. Without this indicator, this station passes from e.g. 0
vehicles to 10 vehicles
            p.capacity = station_capacity[k]
            self.stations[k] = p
            self.queue.append([])

        #creates the permanent objects units
        unit = {}
        for k in range(number_units):
            un = sections()
            un.name = k
            un.n_vehicles = 0
            if scenario_under_study == 1:
                un.attr = units[k].attr # attractivity of a unit (in %)
                un.cumul_att = units[k].cumul_att # the cumulative attractivity is created because
of this.
```

```
            # the last stop of the trip chain, in which the supervisor is recalled, is assessed
through a random
            # extraction in an uniform distribution between 0 and sum of all units attractivities
            self.unit[k] = un
            self.queue_unit.append([])


        # creates the object supervisor, if required
        sup = supervisor() #creates the supervisor
        self.supervis.append(sup)


        # Makes that users hereditate demand characterics and gives them the arrival instant
        # creates the objects users

        for i in range(number_users):
            u = user()
            u.name = i
            # imports all the attributes of the demand of transport
            u.origin = demand[i].origin
            u.destination = demand[i].destination
            if self.case_study == 2:
                u.provenience = demand[i].provenience
            else:
                u.provenience = 999999
            if self.case_study == 1:
                if u.origin == u.destination:
                    # print 'error'
                    STOPP
            u.activity_pattern = demand[i].activity_pattern
            u.typology = demand[i].typology
            if self.case_study == 3:
                u.typology_origin = demand[i].typology_origin
                u.typology_destination = demand[i].typology_destination
            else:
                u.typology_origin = 1
                u.typology_destination = 1
# users are in order, hour by hour. The following method istant arrival defines the exact
minute
# in which a given user appears at its origin station/unit.
            u.istant_arrival(i,self)
            self.users[i] = u

        # creates the objects picav

        for k in range (self.number_picav):  #(number_picav)
            p = picav()
            p.name = k
            self.vehicles[k] = p
            # presence picav assigns to each station the right number of picavs
            self.vehicles[k].presence_picav(k,self)

        # time queue

        for t in range (0,Tmax):
            if t > TI0_max:
                # print '--------------------------------------------------------'
                hhh = t / 60
                mmm = t - hhh*60
                # print 'T =',t,'; hh mm',hhh,':',mmm

                rl = 0 # counter of impossible relocations
                rt = 0 # counter of total relocations
                    # Checks if some of vehicles has finished his single mission trip or trip chain
                    # If so, calls the method "end occupation" of the vehicle
                for j in range(self.number_picav):
                    if ((self.vehicles[j].travel_time == t)&(self.vehicles[j].activity_pattern == 
2)&(self.vehicles[j].state == 1)):
                        # print 'the vehicle',self.vehicles[j].name,'is arrived at the station/unit'
                        self.vehicles[j].end_occupation(t,self)
                    if ((self.vehicles[j].activity_time == t)&(self.vehicles[j].activity_pattern == 
1)&(self.vehicles[j].state == 1)):
                        self.vehicles[j].end_occupation(t,self)
                        # print 'the vehicle',self.vehicles[j].name,'is arrived at the station/unit'

                        # Checks if some of vehicles has finished its redirection trip. If so, calls the 
method "end redirection" of the vehicle
                    if ((self.vehicles[j].redirection_time == t)&(self.vehicles[j].state == 4)):
                        # print 'the vehicle',self.vehicles[j].name,'is arrived at the station'
                        self.vehicles[j].end_redirection(t,self)

                for j in range(self.number_picav):
```

```
                # The vehicle has been recalled by the user and has begun its trip towards the
user. The following checks if the vehicle has
                # reached the user and this trip is ended.
                if ((self.vehicles[j].travel_time == t)&(self.vehicles[j].state ==
5)&(self.vehicles[j].unit == 999999)&(self.vehicles[j].parking == 999999)):
                    # print '---'
                    # print 'the picav has reached the user'
                    self.vehicles[j].end_occupation(t,self)
                if ((self.vehicles[j].parking < 99999)&(self.vehicles[j].unit <
99999))|((self.vehicles[j].parking > 99999)&(self.vehicles[j].unit >
99999)&(self.vehicles[j].state == 0)):
                    print 'error: the vehicle cannot be contemporary in a station and in a unit, or
be nowhere'
                    stop

            for j in range(self.number_users):
                if ((self.users[j].travel_time == t)&(self.users[j].activity_pattern ==
2)&(self.users[j].state == 1)):
                    self.users[j].exits(self)
                if ((self.users[j].activity_time == t)&(self.users[j].activity_pattern ==
1)&(self.users[j].state == 1)):
                    self.users[j].exits(self)



                # Checks if some vehicle has finished to relocate
                # If so, calls the method "end relocation" of the destination station
            for j in range(self.number_picav):
                if (self.vehicles[j].relocation_time == t)&(self.vehicles[j].state == 3):
                    for k in range(self.number_stations):
                        if (self.stations[k].name ==
self.vehicles[j].destination)&(self.vehicles[j].destination < 99999):
                            self.stations[k].end_relocation(self.vehicles[j],t,self)
                            break

            # Checkes if some users are at the last stop of the trip chain. If so, the supervisor
is called.
            # The method "choice final destination" of the supervisor is recalled
            if (self.case_study == 2):
                for j in range(self.number_picav):
                    if (self.vehicles[j].final_time == t)&(self.vehicles[j].activity_pattern ==
1)&(self.vehicles[j].state == 1):
```

```
                        self.supervis[0].choice_final_destination(self.vehicles[j],self,t)

            # Updates the state of charge of vehicles. If the vehicle is in state 1,3,4,5 i.e. it is
moving, it discharges the battery.
            # If it is in state 0, it recharges.
            for k in range(self.number_picav):
                if (self.vehicles[k].state == 1)|(self.vehicles[k].state == 3)|(self.vehicles[k].state
== 4)|(self.vehicles[k].state == 5):
                    self.vehicles[k].discharge(t,self)  # discharges the battery if vehicle is
occupied or relocating
                if (self.case_study <3)|((self.case_study == 3)&(self.vehicles[k].parking <
99999)):
                    if (self.vehicles[k].state == 2)&(self.vehicles[k].unit >
99999)&(self.vehicles[k].parking < 99999):
                        self.vehicles[k].recharge(t,self) # recharges the battery if vehicle is idle at
pl
                    if (self.vehicles[k].state == 0)&(self.vehicles[k].level_charge <
1)&(self.vehicles[k].unit > 99999)&(self.vehicles[k].parking < 99999):
                        self.vehicles[k].recharge(t,self) # recharges the battery if vehicle is idle at
pl


            # Checks if some of vehicles has enough charge to be occupied or to relocate.
            # If so, calls the method "available vehicle" of the station where the vehicle is
            for k in range(self.number_picav):
                if (self.vehicles[k].state == 2)&(self.vehicles[k].level_charge >=
self.vehicles[k].minimal_charge):
                    for j in range(self.number_stations):
                        if (self.stations[j].name ==
self.vehicles[k].parking)&(self.vehicles[k].parking < 99999):
                            # print 'the vehicle',self.vehicles[k].name,'has finished to recharge at the
station',self.stations[j].name
                            self.stations[j].n_veh_relocation = self.stations[j].n_veh_relocation + 1
                            self.stations[j].ready_vehicle(self.vehicles[k],t,self)
                            break

            # Checks if some user arrives at the current instant or has finished the trip chain
            # If so, calls the method "user arrival" of the stations or of the units the users
arrive to
            for j in range(self.number_users):
                if (self.users[j].t_arrival == t):
```

```
            if ((self.case_study == 3)&((self.users[j].typology ==
2)|(self.users[j].typology == 3))):
                for k in range(self.number_unit):
                    if (self.unit[k].name == self.users[j].origin)&(self.users[j].origin <
99999):
                        # print '---'
                        # print 'arriva user',self.users[j].name
                        self.unit[k].arrival_user(self.users[j],t,self)
                        break
            else:
                for k in range(self.number_stations):
                    if (self.stations[k].name == self.users[j].origin)&(self.users[j].origin <
99999):
                        # print '---'
                        # print 'arriva user',self.users[j].name
                        self.stations[k].arrival_user(self.users[j],t,self)
                        break


        # At the end of the day, the users still in queue go home. All stations queues are
set empty.
        if t > 1300:
            for i in range(number_users):
                if (self.users[i].state == 2)|(self.users[i].state == 4):
                    self.users[i].state = 3  # state 3 = gone away
                    self.users[i].time_queue = t - self.users[i].time_entrance_queue #
calculates the queuetime for all the users
                    # print 'The user',self.users[i].name,'renounces and goes.
Queue:',self.users[i].parking
##                      if self.case_study == 3: # tells if the user was in queue at a unit or at a
station
##                          if self.users[i].typology_origin == 1:
##                              # print 'origin at a pl'
##                          else:
##                              # print 'origin at a unit'
                    self.users[i].pikav = 999999
                    self.users_rejected = self.users_rejected + 1  # updates the number of
rejected users. A user is rejected only
                    # when the end of the day is reached, not when the user is in queue for
more e.g. of 15 mins.
            for i in range(len(self.queue)): #  sets all queues empty
```

```
                        self.queue[i] = []
                    for i in range(len(self.queue_unit)):
                        self.queue_unit[i] = []



                # checks if vehicles charging level is positif. If not, makes an error message.
                for k in range(self.number_picav):
#                   print 'vehicle charge level',self.vehicles[k].name,'=',
self.vehicles[k].level_charge
                    if self.vehicles[k].level_charge <= 0:
                        print 'error charge'
                        STOP

                # Beginning of relocation procedure
                # If in some stations the number of vehicles is below or above the thresholds, a
request of relocation
                # is generated. The method "begin relocation" of the stations in request is recalled
                if (t < 1330)&((case_study == 1)|(case_study == 3)):
                    for j in range(self.number_stations):
                        if (self.stations[j].n_veh_relocation <= self.low_critical_thresholds[j]):
                            self.problem = 1 # problem = zero vehicle time: request of a vehicle
                            # print '-----'
                            # print 'the number of vehicles assigned at the
station',self.stations[j].name,'is ',self.stations[j].n_veh_relocation
                            # print 'the low critical threshold is',self.low_critical_thresholds[j]
                            # print 'the high critical threshold is',self.high_critical_thresholds[j]
                            # print 'problem ',self.problem
                            self.stations[j].begin_relocation(t,self)
                        elif (self.stations[j].n_vehicles >= self.high_critical_thresholds[j]):
                            self.problem = 2 # problem = full port time: offer of a vehicle
                            # print '-----'
                            # print 'the number of vehicles currently at the
station',self.stations[j].name,'is ',self.stations[j].n_vehicles
                            # print 'the low critical threshold is',self.low_critical_thresholds[j]
                            # print 'the high critical threshold is',self.high_critical_thresholds[j]
                            # print 'problem ',self.problem
                            self.stations[j].begin_relocation(t,self)

                # checks of the picavs, if their final time or travel time or activity time has a
clearly wrong value.
```

```
            # this prevents from loosing vehicles "on the way"
            for j in range(self.number_picav):
               if self.vehicles[j].state == 1:
                  if (self.vehicles[j].final_time > 10000)&(self.vehicles[j].travel_time > 
10000)&(self.vehicles[j].activity_time > 10000):
                     print 'error'
                     STOP
                  if (self.vehicles[j].final_time < t)&(self.vehicles[j].travel_time < 
t)&(self.vehicles[j].activity_time < t):
                     print 'error'
                     STOP
            # prints online statistics
            self.online_statistics(t)
      self.offline_statistics() #prints offline statistics



   def online_statistics(self,t):
      # LENGTH OF QUEUES AND AVAILABLE VEHICLES at each station in each 
time instant
      self.queue_lengths = []  # creates the statistics lists
      self.vehicles_available = []
      self.vehicles_ideally_present = [] # ideally present is related to the counter "n veh 
relocation" exposed above
      # Ideally presents are the vehicle assigned to a unit (not actually present at a unit). For 
example,if a vehicle
      # begins relocating, the destination station has a vehicle ideally present more, but not 
an available vehicle more.
      # the vehciles available number will be updated when the vehicle effectively arrives.
      for i in range(len(self.queue)):
         self.queue_lengths.append(len(self.queue[i]))
      # print 'lengths of queue',self.queue_lengths
      for i in range(self.number_stations):
         self.vehicles_available.append(self.stations[i].n_vehicles)
      for i in range(self.number_stations):
         self.vehicles_ideally_present.append(self.stations[i].n_veh_relocation)
      # print 'vehicles available',self.vehicles_available
      # print 'vehicles ideally present',self.vehicles_ideally_present
      # controls that in any station the number if vehices available is negative
      for k in range(number_stations):
         if (self.vehicles_ideally_present[k] < 0)|(self.vehicles_available[k] < 0):
            print 'error severe: impossible to have negative vehicles available'
            stop
        # severe case: controls that a vehicle is available "nowhere"
        for i in range(self.number_picav):
          if (self.vehicles[i].state == 0)&(self.vehicles[i].parking >
9999)&(self.vehicles[i].unit > 9999):
            # print
'vehicle',self.vehicles[i].name,'state',self.vehicles[i].state,'parking',self.vehicles[i].parking,'u
nit',self.vehicles[i].unit
            print 'error very severe'
            STOP
        if case_study == 3:
        # the name ideally of the following list is an unhappy choice. However, it contains the
number of vehicle that are present at a unit, i.e.
        # along the road in proximity of it. It checks that the number of vehicles available at
the units is positive.
          self.vehicles_ideally_unit = []
          for k in range(number_unit):
            self.vehicles_ideally_unit.append(self.unit[k].n_vehicles)
          # print 'vehicles available units',self.vehicles_ideally_unit
          for k in range(number_unit):
            if self.vehicles_ideally_unit[k] < 0:
              print 'error severe: impossible to have negative vehicles available'
              stop





        # NUMBER OF VEHICLES IN EACH STATE

        # initialization of counters of all the states
        self.picav_user = 0 # occupied by a user
        self.picav_relocation = 0 #relocating
        self.picav_charge = 0 # recharging
        self.picav_available_all = 0 # available
        self.picav_encounter = 0 # going towards the user
        self.picav_full_redirect = 0 # redirected because the station is full

        # update of the 5 counters
        for j in range(self.number_picav):
            if self.vehicles[j].state == 1:
                self.picav_user = self.picav_user + 1
            if self.vehicles[j].state == 2:
                self.picav_charge = self.picav_charge + 1
            if self.vehicles[j].state == 3:
                self.picav_relocation = self.picav_relocation + 1
            if self.vehicles[j].state == 0:
                self.picav_available_all = self.picav_available_all + 1
            if self.vehicles[j].state == 4:
                self.picav_full_redirect = self.picav_full_redirect + 1
            if self.vehicles[j].state == 5:
                self.picav_encounter = self.picav_encounter + 1

# update of the lists. Each list contains the number of picavs in one of the 4 states at the
current time instant.
# not considered states 4 and 5.
        self.number_picav_state1.append(self.picav_user)
        self.number_picav_state2.append(self.picav_charge)
        self.number_picav_state3.append(self.picav_relocation)
        self.number_picav_state0.append(self.picav_available_all)
        self.number_picav_state4.append(self.picav_full_redirect)
        self.number_picav_state5.append(self.picav_encounter)

    # calculates the total number of relocations, and also the number of impossible
relocations
        rt = 0
        rl = 0
        for j in range(number_stations):
            rt = rt + self.stations[j].rt # updates the relocations' counter for the current time
instant
            rl = rl + self.stations[j].rl # rt = counter of total relocations; rl = counter of
impossible relocations
            self.stations[j].rt = 0
            self.stations[j].rl = 0
##          if (self.case_study == 1)|(self.case_study == 3):
            # print 'total relocations',rt
            # print 'impossible relocations',rl
    # updates the two lists which keep the number of impossible relocations and the total
number
    # of relocations at each time instant, for all the simulation period
```

```
        self.number_impossible_relocations.append(rl)
        self.number_total_relocations.append(rt)
        self.calc_max_waiting(t)
        for i in range(number_stations):
            self.number_vehicles_available[i].append(self.stations[i].n_vehicles)
            self.max_time_queue[i].append(self.stations[i].max_waiting_time)
            self.number_users_queue[i].append(len(self.queue[i]))



    def offline_statistics(self):

    # determines the total and average time in queue spent by users
        for i in range(number_users):
            self.time_stayed_queue.append(self.users[i].time_queue)
        self.total_time_queue = 0.0
        for i in range(number_users):
            self.total_time_queue = self.total_time_queue + self.time_stayed_queue[i]
        self.average_time_queue = float(self.total_time_queue / number_users)

    # # prints some final statistics
        # print 'total time in queue (min) =',self.total_time_queue
##          if (self.case_study == 1)|(self.case_study == 3):
            # print 'total relocation time (min) =',self.total_relocation_time
            # print 'number of relocations =',self.number_relocations
        # print 'average wait time (min) =',self.average_time_queue
        # print 'number of users',number_users

        # Calculates percentiles and creates the list of all wait times of all users
        self.time_stayed_queue.sort()
        # print 'wait times =',self.time_stayed_queue
        lll_lista = len(self.time_stayed_queue)
        self.provv_95 = int(lll_lista*0.95+0.5)
        self.provv_90 = int(lll_lista*0.90+0.5)
        self.provv_50 = int(lll_lista*0.50+0.5)
        self.percentile_95 = self.time_stayed_queue[self.provv_95]
        self.percentile_90 = self.time_stayed_queue[self.provv_90]
        self.percentile_50 = self.time_stayed_queue[self.provv_50]
        self.lista = self.time_stayed_queue
        self.total_queuetime = self.total_time_queue
```

```
# Calculates the maximum waiting time for all the stations. Recalled by the
    def calc_max_waiting(self,t):
        self.max_waiting_time = []
        for j in range(number_stations):
            self.max_waiting_time.append(0)
            for k in range(number_users):
                if (self.users[k].state == 2)&(self.users[k].queue ==
self.stations[j].name)&(self.users[k].queue < 99999):
                    provv = t - self.users[k].time_entrance_queue
                    if self.max_waiting_time[j] < provv:
                        self.max_waiting_time[j] = provv
        globale = []
        for j in range(number_stations):
            self.stations[j].max_waiting_time = self.max_waiting_time[j]
            globale.append(self.stations[j].max_waiting_time)
        # print 'max waiting time at stations',globale




#--------------- BEGIN DEFINITION OF CLASSES ----------------



class user:
    name = 0   #name
    distance = 0 # distance and slope are related to the user's trip and are transferred to the
picav unit.
    slope = 0
    travel_time = 0
    t_arrival = 999999 #arrival time
    time_waiting = 0 #waiting time  # FREE VARIABLE USE IT!!!
    time_entrance_queue = 999999 #time of entering queue
    destination = 0 #destination
    activity_pattern = 0
    provenience = 0 #provenience
    travel_time = 0 #travel time
    activity_time = 0 #activity time
    parking_arrival = 999999 #parking of arrival
    state = 0  # 0 = initialized, still has to be generated, 1 = in the vehicle, 2 = in the queue, 3
= dead, 4 = already assigned a vehicle
    parking = 999999 #parking
    pikav = 999999
    prob = 0
    typology = 0 #1 = trip from a station on the border of the intervention area ; 2 = return
trip to a station on the border of the intervention area; 3 = trip among units
    queue = 999999
    time_queue = 0
    time_provv_queue = 0
    memory_pl = 999999
    final_time = 999999
    reapparance_place = 999999
    unit = 99999

##    def case_study_management(self,sim):
##        if sim.case_study == 3:
##            for j in range(sim.number_users):
##                if self.typology == 2:
##                    self.typology_origin = 2
##                else:
##                    self.typology_origin = 1

    def istant_arrival (self,k,sim):  # defines the exact instant in which the user arrives at the
origin parking lot
                                  # the parking lot origin and the users hourly arrival period are given
        if k < nor_morning[0]:  # this is for the users arrived in the morning
            self.t_arrival = int(uniform(TI1_min,TI1_min+60))
        for j in range(1,len(nor_morning)):
            if (k >= nor_morning[j-1])&(k < nor_morning[j]):
                self.t_arrival = int(uniform((TI1_min+60*j),(TI1_min+60*(j+1))))
        if (k < nor_afternoon[0])&(k >= number_users_morning):  # this is for the users
arrived in the afternoon
            self.t_arrival = int(uniform(TI2_min,TI2_min+60))
        for j in range(1,len(nor_afternoon)):
            if (k >= nor_afternoon[j-1])&(k < nor_afternoon[j]):
                self.t_arrival = int(uniform((TI2_min+60*j),(TI2_min+60*(j+1))))
        # in the evening no users arrive, they only finish their activities
        self.parking_arrival = self.origin # arrival pl of user is the same as origin
```

```
   def mission_generation (self,t,sim): # defines if the user will have an activity travel
pattern or not
        # defines also the distances and the slopes that the user will encounter in its trip
        # calculates the user travel time and the activity time as a function of travel time
        if (sim.case_study == 2)&(self.activity_pattern == 1): # case study of
supervisor/flexible users; all users have a trip chain.
            self.calculate_reapparance_place(sim) # if the user has a trip chain and the case of
study is with the supervisor
            self.destination = 999999            # it calculates the last stop in the trip chain in
which the supervisor is called
            self.calculate_activity_time(sim,t)     # recalls the method which calculates the
activity time duration. In this case of
            self.activity_time = 9999999           # study, the time duration calculated is the one
before the supervisor call
            # print 'the user will reappear at the unit',self.reapparance_place,'after a
time',self.final_time
            if self.reapparance_place > 1000:
                print 'error'
                STOP
            if self.final_time > 10000:
                print 'error'
                stop
        elif (sim.case_study == 2)&(self.typology == 2): # case study of supervisor/flexible
users.
            self.destination = 999999                  # users are performing single mission trip from
inside the city centre to outside
        elif (sim.case_study == 3)&((self.activity_pattern == 2)|(self.typology == 3)):
            self.calculate_activity_time(sim,t) # calculates the duration of the travel time and
the activity time. This is for case of study 3
                                                # and users having origin or destination at a unit
        else: # all case study 1, case study 2 single mission trip from outside to inside the
intervention area, case study 3 with both
                # origin and destination at some parking lots.
            for j in range(len(dist_pls_pls)): # here calculates the travel time between the origin
and destination station
                if (dist_pls_pls[j].origin == self.origin)&(dist_pls_pls[j].destination ==
self.destination):
                    self.distance = dist_pls_pls[j].value
                    self.slope = pend_pls_pls[j].value
                    if (sim.case_study == 1):
```

```
                # print 'origin',self.origin,'; destination',self.destination,'; the parking lot is
distant mt.',self.distance
             # distance: in metres; speed: in metres/seconds.
             # travel time: in minutes
                if (t < TI2_min):
                   self.travel_time = round(self.distance / (sim.vel_morning*60))
                elif (t <= TI2_max):
                   self.travel_time = round(self.distance / (sim.vel_aftern*60))
                else:
                   self.travel_time = round(self.distance / (sim.vel_evening*60))
                # print 'travel time',self.travel_time
                if self.travel_time == 0:
                   self.travel_time = 1
             if ((sim.case_study == 2)&(self.typology == 1)&(self.activity_pattern == 2)):
                # print 'origin',self.origin,'; destination',self.destination,'; the parking lot is
distant mt.',self.distance
             # distance: in metres; speed: in metres/seconds.
             # travel time: in minutes
                if (t < TI2_min):
                   self.travel_time = round(self.distance / (sim.vel_morning*60))
                elif (t <= TI2_max):
                   self.travel_time = round(self.distance / (sim.vel_aftern*60))
                else:
                   self.travel_time = round(self.distance / (sim.vel_evening*60))
                # print 'travel time',self.travel_time
                if self.travel_time == 0:
                   self.travel_time = 1
             if ((sim.case_study == 3)&(self.activity_pattern == 1)):
                # print 'origin',self.origin,'; destination',self.destination,'; the parking lot is
distant mt.',self.distance
             # distance: in metres; speed: in metres/seconds.
             # travel time: in minutes
                if (t < TI2_min):
                   self.travel_time = round(self.distance / (sim.vel_morning*60))
                elif (t <= TI2_max):
                   self.travel_time = round(self.distance / (sim.vel_aftern*60))
                else:
                   self.travel_time = round(self.distance / (sim.vel_evening*60))
                # print 'travel time',self.travel_time
                if self.travel_time == 0:
                   self.travel_time = 1
```

```
    # calculates the activity time and the final time. Final time is the time in which the user
performs the last stop of
    # the trip chain and calls the supervisor.
                if (sim.case_study == 2)&(self.activity_pattern == 1):
                    self.final_time = self.travel_time * sim.coeff_activity_pattern
                    # print 'supervisor call time',self.final_time
                if (sim.case_study != 2)&(self.activity_pattern == 1):
                    self.activity_time = self.travel_time * sim.coeff_activity_pattern
                    # print 'activity time',self.activity_time
        if self.activity_pattern == 2:
            self.activity_time = 9999999
            self.final_time = 99999999
        if self.travel_time == 0:
            self.travel_time = 1
        if self.activity_time == 0:
            self.activity_time = 1
        if self.final_time == 0:
            self.final_time = 1

    def calculate_reapparance_place(self,sim):
        # defines the reapparance place, i.e. the unit in which the user performs the last stop of
his trip chain and recalls the supervisor.
        prob = uniform (0,100)
        zzz = 999999
        if (prob <= sim.unit[0].cumul_att):
            self.reapparance_place = sim.unit[0].name
            zzz = 0
        else:
            for j in range (1,number_units):
                if (prob > sim.unit[j-1].cumul_att)&(prob <= sim.unit[j].cumul_att):
                    self.reapparance_place = sim.unit[j].name
                    zzz = j

    def exits(self,sim):
        self.state = 3
        # print 'the user',self.name,'exits the system'

    def calculate_activity_time(self,sim,t):
        # trip with origin at a station and destination at a unit
        if (sim.case_study == 2)|((sim.case_study == 3)&(self.typology == 1)):
            for j in range(len(dist_pls_points)):
```

```
            if (dist_pls_points[j].origin == self.origin)&((dist_pls_points[j].destination ==
self.reapparance_place)|(dist_pls_points[j].destination == self.destination)):
                self.distance = dist_pls_points[j].value
                self.slope = pend_pls_points[j].value
                # distance: in metres; speed: in metres/seconds.
                # travel time: in minutes

        # trip with origin at a unit and destination at a station
        if (sim.case_study == 3)&(self.typology == 2):
            for j in range(len(dist_points_pls)):
                if (dist_points_pls[j].origin == self.origin)&(dist_points_pls[j].destination ==
self.destination):
                    self.distance = dist_points_pls[j].value
                    self.slope = pend_points_pls[j].value

        # trip with both origin and destination at a unit
        if (sim.case_study == 3)&(self.typology == 3):
            for j in range(len(dist_points_points)):
                if (dist_points_points[j].origin == self.origin)&(dist_points_points[j].destination
== self.destination):
                    self.distance = dist_points_points[j].value
                    self.slope = pend_points_points[j].value
                    # distance: in metres; speed: in metres/seconds.
                    # travel time: in minutes

        # calculates the travel time and the activity time
        if (t < TI2_min):
            self.travel_time = round(self.distance / (sim.vel_morning*60))
        elif (t <= TI2_max):
            self.travel_time = round(self.distance / (sim.vel_aftern*60))
        else:
            self.travel_time = round(self.distance / (sim.vel_evening*60))
        if self.travel_time == 0:
            self.travel_time = 1
##          print 'travel time',self.travel_time
        if sim.case_study == 2:
            self.final_time = self.travel_time * sim.coeff_activity_pattern # final time is always
the instant of the supervisor call.
        else:
            self.activity_time = self.travel_time * sim.coeff_activity_pattern
```

```
class picav:
    name = 999999
    parking = 999999  #parking
    provenience = 999999 # parking origin
    destination = 999999 # destination parking, or destination perypheral area
    state = 0 # 0 free, 1 occupied by user, 2 in charge, 3 in relocation, 4 available only to
users but not for relocation (too low battery)
    activity_time = 999999  #activity time
    distance = 999999 # distance travelled
    travel_time = 999999 #travel time
    activity_pattern = 999999
    time_discharge = 999999 # remaining time for using the battery
    minimal_charge = 0.1 # important: remember to update!
        # It is the minimum charging level, i.e. for performing the longest relocation
    state_charge = 999999 # Actual level of battery charge
    final_time = 999999 # time of the last stop of the activity chain
    user = 999999 # name of PICAV user
    parking_arrival = 100000
    level_charge = 1
    problem_relocation = 0
    relocation_time = 9999999
    redirection_time = 9999999
    battery_discharge_time = 1
    slope = 0.0
    distances = 999999
    reapparance_place = 999999 # unit in which the picav user calls the supervisor
    final_time = 999999
    unit = 999999
    typology_trip = 0


# ----- ATTRIBUTES OF THE BATTERY, NOT SUBJECTED TO MODIFICATION -----
--------
# Battery is composed of 15 blocks connected in serial, each of them is composed of 27
cells connected in parallel
    time_considered = battery_discharge_time # TIME OF DISCHARGE IN MINUTES
    height = 1.698 # m
    width = 0.8 # m
    length = 1.62 # m
    weight = 250 # kg
    Cx = 0.7 # aerodynamic coefficient
    frontal_area = 1.3584 # mq
    mechanical_efficiency = 0.93
    motor_voltage = 48 # V DC
    battery_voltage = 48 # V = Vblock * n.blocks
    gravity_constant = 9.81 # m/s^2
    air_density = 1.239 # km / mc
    electrical_efficiency = 0.92
    battery_capacity = 202 # Ampere hour (energy provided by the battery in 1 hour)
    accelleration = 0
    balancing_time = 0.1
    added_weight = weight # weight of a person
    rolling_coefficient = 0.015


    def presence_picav (self,k,sim):  # the picav is generated and its initial station is defined
        station_vehicle = []
        counter = 0
        for j in range(number_stations):
            counter = counter + sim.NPP[j]
            station_vehicle.append(counter)
        if self.name < (station_vehicle[0]):
            self.parking = 1
        else:
            for j in range(1,number_stations):
                if (self.name >= station_vehicle[j-1])&(self.name < station_vehicle[j]):
                    self.parking = j+1
        self.level_charge = 1
        self.unit = 999999


    def occupation (self,user,t,sim):  # occupation of the vehicle from the user
##        print 'the user',user.name,'enters the vehicle',self.name
        user.mission_generation(t,sim) # determines the destination and the end of the trip or
trip chain of the occupying user
        user.memory_pl = user.origin
        self.origin = user.origin # assigns to the picav all the attributes that have belonged to
the user.
        self.activity_pattern = user.activity_pattern # the attributes of the picav
        self.destination = user.destination # from the attributes of the user
        self.parking_arrival = user.parking_arrival
        self.provenience = user.provenience
        self.reapparance_place = user.reapparance_place
        self.typology_destination = user.typology_destination
        self.typology_trip = user.typology
        if (self.reapparance_place > 1000)&(sim.case_study == 2)&(self.activity_pattern == 
1):
            print 'vehicle',self.name,'user',user.name
            print 'error',self.reapparance_place,user.reapparance_place
            stop
        self.parking = 9999999
        self.unit = 999999
        self.state = 1
        user.state = 1
        user.pikav = self.name
        self.user = user.name
        if user.travel_time == 0:
            user.travel_time = 1
        if user.activity_time == 0:
            user.activity_time = 1
        if user.final_time == 0:
            user.final_time = 1
        user.travel_time = user.travel_time + t  # updates the travel time, activity time and 
final time with the current time instant
        user.activity_time = user.activity_time + t # in order to determine the instant of end of 
the user's trip
        user.final_time = user.final_time + t
        self.final_time = user.final_time
##      if (sim.case_study == 2)&(user.activity_pattern == 1):
##          # print 'supervisor call time',user.final_time
        self.travel_time = user.travel_time
        self.activity_time = user.activity_time
        self.slope = user.slope
##      if sim.case_study != 2:
##          if self.activity_pattern == 1:
##              # print 'and will end the trip chain at the instant',self.activity_time
##          else:
##              # print 'and will end the single mission trip at the instant',self.travel_time
##          # print 'user name',user.name,'type of user', user.typology,'vehicle 
occupied',self.name
##      if sim.case_study == 2:
```

```
##          if self.activity_pattern == 1:
##              # print 'and will end the trip chain at the instant',self.final_time
##          elif user.typology == 1:
##              # print 'and will end the single mission trip at the instant',self.travel_time
##          # print 'user name',user.name,'type of user', user.typology,'vehicle
occupied',self.name
##
##      if (sim.case_study == 1):
##          # print 'origin pl',self.origin,'destination pl',self.destination
##      if sim.case_study == 2:
##          if (self.activity_pattern == 2)&(user.typology == 1):
##              # print 'origin pl',self.origin,'destination pl',self.destination
##              if self.destination > 99999:
##                  print 'error: destination',self.destination
##                  stop
##          else:
##              print 'origin pl',self.origin,'destination area',self.provenience
##              if self.provenience > 99999:
##                  print 'error: destination area',self.provenience
##                  stop
        if (sim.case_study == 2)&(self.activity_pattern == 2)&(user.typology == 2):
            self.reapparance_place = self.origin
            sim.supervis[0].choice_final_destination(self,sim,t)
##      if (sim.case_study == 3):
##          print 'origin pl/unit',self.origin,'destination pl/unit',self.destination

# the picav begins to approach the user's position
    def begin_approach(self,sim,section):
        self.state = 5 # 5 = approaching the unit
        self.travel_time = section.travel_time
        if self.travel_time > 5*Tmax:
            print 'error: travel time too high. Travel time =',self.travel_time
            stop
        self.destination = section.destination
        self.unit = 999999
        self.parking = 999999
        if self.destination > 999:
            print 'error'
            stop
##      print '-----'
##       print 'the call or the offer of vehicle is begun from the destination
unit',self.destination
##       print 'the arriving vehicle is the',self.name,'after a time',self.travel_time

# end of the picav occupation
    def end_occupation (self,t,sim):  # picav vehicles begins being recharged
        # state of Picav vehicle: 0 = available/free, 1 = occupied by a user, 2 = in charge, 3 =
relocating, 4 = in charge(2), 5 = moving towards a unit
        # re-initializes all the picav parameters
        self.activity_pattern = 999999
        if (sim.case_study == 3)&(self.typology_destination == 2):
            self.unit = self.destination
            self.state = 0
        else:
            self.parking = self.destination
        self.final_time = 99999999
        self.activity_time = 99999999
        self.travel_time = 99999999
        self.destination = 99999999
        self.origin = 9999999
        self.provenience = 999999
        self.typology_trip = 0
        self.reapparance_place = 999999
    # recalls the method "arrival vehicle" at the destination unit or at the destination station
        if ((sim.case_study == 3)&(self.typology_destination == 2)):
            for j in range(sim.number_unit):
                if (sim.unit[j].name == self.unit)&(self.unit < 99999):
                    # print 'the vehicle is arrived at the unit',self.unit
                    sim.unit[j].arrival_vehicle(self,t,sim)
                    break
        else:
            for j in range(sim.number_stations):
                if (sim.stations[j].name == self.parking)&(self.parking < 99999):
                    sim.stations[j].n_veh_relocation = sim.stations[j].n_veh_relocation + 1
                    # print 'the vehicle',self.name,'is arrived at the station',self.parking
                    sim.stations[j].arrival_vehicle(self,t,sim)
                    break

# end of the picav redirection trip: the redirection trip occurs when the destination station is
full. In this case, the
# vehicle needs to be redirected to another station
```

```
    def end_redirection(self,t,sim):
        self.redirection_time = 999999
        self.parking = self.destination
        self.unit = 999999
        for j in range(number_stations):
            if (self.destination == sim.stations[j].name)&(self.destination < 99999):
                # print 'the vehicle',self.name,'has finished its redirection and is arrived at the
station',self.parking
                sim.stations[j].n_veh_relocation = sim.stations[j].n_veh_relocation + 1
                self.destination = 999999
                sim.stations[j].arrival_vehicle(self,t,sim)
                break

# implementation of the law of the battery discharge process
    def discharge(self,t,sim):
    # Remember: here the speed is expressed in mt/sec
        if (self.state == 1)|(self.state == 4):
            if (t < TI2_min):
                self.speed = sim.vel_morning
            elif (t <= TI2_max):
                self.speed = sim.vel_aftern
            else:
                self.speed = sim.vel_evening
        elif (self.state == 3)|(self.state == 5):
            if (t < TI2_min):
                self.speed = sim.vel_autom_morn
            elif (t <= TI2_max):
                self.speed = sim.vel_autom_aft
            else:
                self.speed = sim.vel_autom_even
        self.slope = self.slope / 100
        if self.slope < 0:
            self.slope = 0.0
        self.angulation  = atan(self.slope)
        self.first_contribution = (self.weight + self.added_weight)* self.accelleration
        self.second_contribution = (self.weight + self.added_weight)* self.gravity_constant *
self.rolling_coefficient * cos(self.angulation)
        self.third_contribution = ((1/2)* self.frontal_area * self.air_density * self.Cx *
((self.speed)*(self.speed)))
        self.fourt_contribution = (self.weight + self.added_weight)* self.gravity_constant *
sin(self.angulation)
```

```
        self.resistenz = self.speed * (self.first_contribution + self.second_contribution + 
self.third_contribution + self.fourt_contribution)
        self.power = 
(1/self.electrical_efficiency)*((1/self.mechanical_efficiency)*self.resistenz)/10
        self.ampere_consumed = (self.power * 1000)/self.motor_voltage
        # follows the output of the method
        self.battery_consumed = 0.01*(self.ampere_consumed * 
self.time_considered)/(60*self.battery_capacity)
        if self.activity_pattern == 1:
            self.battery_consumed = self.battery_consumed/sim.coeff_activity_pattern
#        print 'battery consumed by the vehcile',self.name,'=',self.battery_consumed
        self.level_charge = self.level_charge - self.battery_consumed
        if self.level_charge > 1:
            self.level_charge = 1
#        print 'level of charge of the vehicle',self.name,'=',self.level_charge


# implementation of the law of the battery recharge process. The normal recharge technique 
has been considered. The fast recharge indeed damages the battery
    def recharge(self,t,sim):
        self.recharge_time = 1
        self.initial_level_recharge = self.level_charge
        if (self.initial_level_recharge >= 0.8):
            self.final_level_recharge = self.initial_level_recharge + 0.1/(60*(1 + 
self.balancing_time))
        if (self.initial_level_recharge < 0.8):
            self.final_level_recharge = self.initial_level_recharge + 0.3/(60*(1 + 
self.balancing_time))
            if (self.final_level_recharge >= 0.8):
                self.final_level_recharge = 0.8 + 0.1 *(((self.initial_level_recharge - 
0.8)/0.3)+(1/(60*(1 + self.balancing_time))))
        if (self.final_level_recharge < 0)|(self.initial_level_recharge 
<0)|(self.initial_level_recharge > self.final_level_recharge):
            print ' mistake on the charge level'
            STOP
        self.level_charge = self.final_level_recharge
        if self.level_charge > 1:
            self.level_charge = 1

    def calculate_travel_time(self,sim,t):  # travel time between a unit and a station. It 
defines this travel time which is needed for the supervisor call
```

```
        for j in range(len(dist_points_pls)):
            if (dist_points_pls[j].origin == self.origin)&(dist_points_pls[j].destination ==
self.destination):
                self.distance = dist_points_pls[j].value
                self.slope = pend_points_pls[j].value
##                  if self.activity_pattern == 2:
##                      print 'origin unit',self.origin,', destination pl',self.destination,'. The parking
lot is distant',self.distance
                # distance: in metres; speed: in metres/seconds.
                # travel time: in minutes
                if (t < TI2_min):
                    self.travel_time = round(self.distance / (sim.vel_morning*60))
                elif (t <= TI2_max):
                    self.travel_time = round(self.distance / (sim.vel_aftern*60))
                else:
                    self.travel_time = round(self.distance / (sim.vel_evening*60))
                if self.travel_time == 0:
                    self.travel_time = 1
##                  print 'travel time',self.travel_time
                break

    def travel_time_pls(self,sim,t):  # travel time among parking lots
        for j in range(len(dist_pls_pls)):
            if (dist_pls_pls[j].origin == self.origin)&(dist_pls_pls[j].destination ==
self.destination):
                self.distance = dist_pls_pls[j].value
                self.slope = pend_pls_pls[j].value
##                  print 'origin pl',self.origin,', destination pl',self.destination,'. The parking lot is
distant',self.distance
                # distance: in metres; speed: in metres/seconds.
                # travel time: in minutes
                if (t < TI2_min):
                    self.travel_time = round(self.distance / (sim.vel_morning*60))
                elif (t <= TI2_max):
                    self.travel_time = round(self.distance / (sim.vel_aftern*60))
                else:
                    self.travel_time = round(self.distance / (sim.vel_evening*60))
                if self.travel_time == 0:
                    self.travel_time = 1
                if self.origin == self.destination:
                    self.travel_time = 999999
```

```
##              print 'travel time',self.travel_time
            break

    def calculate_parking_lot(self,sim,t):  # gives the supervisor all the travel times of all the
possible user's destinations stations
        for j in range(number_stations):
            self.destination = sim.stations[j].name
##          print 'destination =',self.destination
            if self.activity_pattern == 1:
                self.calculate_travel_time(sim,t)  # recalls the method to determine the
destination station.
            elif self.activity_pattern == 2:
                self.travel_time_pls(sim,t)
            if self.travel_time == 0:
                self.travel_time = 999999
            sim.stations[j].travel_time = self.travel_time
        self.destination = 999999




class parking:
    max_waiting_time = 0  # maximum waiting time
    n_vehicles = 0
    name = 999999
    user = 999999
    travel_time = 999999
    rt = 0
    rl = 0
    problem = 999999  # 1 = zero vehicle time ; 2 = full port time

    def arrival_user(self,user,t,sim):  # arrival of user
        # print 'the user',user.name,'is now arrived at the station',self.name
        if (self.n_vehicles > 0):
            for j in range(sim.number_picav):
                if (sim.vehicles[j].state == 0)&(sim.vehicles[j].parking ==
self.name)&(sim.vehicles[j].parking < 99999):
                    self.n_vehicles = self.n_vehicles - 1
                    self.n_veh_relocation = self.n_veh_relocation - 1
                    # print 'subtraction of a vehicle',sim.vehicles[j].name,'at the station',self.name
                    sim.vehicles[j].occupation(user,t,sim)
                    break
        else:
            nnn = self.name - 1
            user.queue = self.name
            user.state = 2
            user.parking = self.name
            user.time_entrance_queue = t
            sim.queue[nnn].append(user)
            # print 'the user',user.name,'is now in queue at the station',self.name


    def arrival_vehicle(self,picav,t,sim):  # the vehicles arrives at the parking
        picav.parking = self.name
        picav.unit = 999999
        if self.n_vehicles < self.capacity:
            picav.state = 0
            picav.parking = self.name
            picav.travel_time = 99999999
            picav.relocation_time = 9999999
            picav.user = 9999999
            if picav.level_charge >= picav.minimal_charge:
                picav.state = 0 # available in toto
                # print 'the picav',picav.name,'is now recharging'
                # print 'I call the event ready vehicle'
                self.ready_vehicle(picav,t,sim)
            else:
                picav.state = 2
                self.n_veh_relocation = self.n_veh_relocation - 1
                # print 'the picav',picav.name,'is now recharging'

        # if the station is full, the vehicle must be redirected.
        else:
            # print 'the station',self.name,'is full and the vehicle must be redirected'
            self.n_veh_relocation = self.n_veh_relocation - 1
            picav.parking = 999999
            picav.unit = 999999
            index = -1
            distance = 999999
        # among the stations, chooses the nearest one which is able to accept the vehicle
            for i in range(number_stations):
                if sim.stations[i].n_vehicles < high_buffer_thresholds[i]:
                    for j in range(len(dist_pls_pls)):
                        if (dist_pls_pls[j].origin == self.name)&(dist_pls_pls[j].destination ==
    sim.stations[i].name):
                            if dist_pls_pls[j].value < distance:
                                distance = dist_pls_pls[j].value
                                index = sim.stations[i].name
            if index == -1:
                # print 'error, no receiving station. The picav will push in the destination station
    hoping for some space soon'
                picav.state = 0
                picav.parking = self.name
                picav.travel_time = 99999999
                picav.relocation_time = 9999999
                picav.user = 9999999
                if picav.level_charge >= picav.minimal_charge:
                    picav.state = 0 # available in toto
                    # print 'the picav',picav.name,'is now recharging'
                    # print 'I call the event ready vehicle'
                    self.ready_vehicle(picav,t,sim)
                else:
                    picav.state = 2
                    self.n_veh_relocation = self.n_veh_relocation - 1
                    # print 'the picav',picav.name,'is now recharging'

            else:
                # if there is a receiving station, the picav is sent to it
                # print 'supporting station',index
                self.origin = self.name
                self.destination = index
                picav.destination = index
                picav.origin = self.name
                picav.travel_time_pls(sim,t) # calculation of the travel time to the receiving
    station
    ##              if picav.travel_time > Tmax:
    ##                  print 'ERRORE'
    ##                  STOP
                picav.travel_time = picav.travel_time + t
                picav.redirection_time = picav.travel_time
                if picav.state == 1: # updates the instant of end of the trips also to the occupying
    user
                    picav.typology = 1
                    for j in range(number_users):
                        if (sim.users[j].name == picav.user)&(picav.user <
    99999)&(sim.users[j].state == 1):
                            sim.users[j].activity_time = picav.redirection_time
                            sim.users[j].travel_time = picav.redirection_time
                            break
                picav.state = 4
                picav.parking = 999999
                picav.unit = 999999
                # print 'the picav will arrive at the destination at the
    instant',picav.redirection_time


        # the vehicle has enough charging level and therefore it is free for being occupied.
        def ready_vehicle(self,picav,t,sim):
            picav.parking = self.name
            nnn = self.name - 1
            if (len(sim.queue[nnn])>0)&(picav.level_charge > picav.minimal_charge):
            # print 'length of queue',nnn,'=',len(sim.queue[nnn])
            uuu = sim.queue[nnn].pop(0)
            # print 'I estract the user',uuu.name,'from the queue at the station',self.name
            for j in range(number_users):
                if (sim.users[j].name == uuu.name)&(uuu.name < 99999):
                    sim.users[j] = uuu
                    sim.users[j].state = 2
                    sim.users[j].time_queue = t - uuu.time_entrance_queue
                    self.n_veh_relocation = self.n_veh_relocation - 1
                    if (uuu.time_entrance_queue == 0)|(uuu.time_entrance_queue > 99999):
                        print 'errore grave'
                        stop
                    picav.state = 1
                    picav.occupation(sim.users[j],t,sim)
            else:
            if picav.level_charge > picav.minimal_charge:
                picav.state = 0 # available in toto
                self.n_vehicles = self.n_vehicles + 1
            # print 'the vehicle',picav.name,'is now free at the station',self.name
            # print 'addiction of a vehicle to station',self.name


        # Beginning of relocation procedure
        def begin_relocation(self,t,sim):
            minimal = 100000000
            misser = -1
            pppp = -1
            receiver = -1
            if sim.problem == 1: # zero vehicle time
            receiver = self.name
            if sim.problem == 2: # full port time
            misser = self.name
            for i in range(len(dist_pls_pls)):

            # it checkes which can be the supporting station
            if ((dist_pls_pls[i].destination == self.name)&(sim.problem ==
    1))|((dist_pls_pls[i].origin == self.name)&(sim.problem == 2)):
                for j in range(sim.number_stations):
                    pppp = sim.stations[j].name - 1
                    if (sim.stations[j].n_vehicles >=
    sim.low_buffer_thresholds[pppp])&(sim.problem == 1)&(sim.stations[j].name !=
    self.name):
                        if sim.stations[j].name == dist_pls_pls[i].origin:
                            # print 'possible misser =',sim.stations[j].name,'; distance
    =',dist_pls_pls[i].value
                            if (dist_pls_pls[i].value < minimal):
                                minimal = dist_pls_pls[i].value
                                misser = dist_pls_pls[i].origin
                    if (sim.stations[j].n_vehicles <=
    sim.high_buffer_thresholds[pppp])&(sim.problem == 2)&(sim.stations[j].name !=
    self.name):
                        if sim.stations[j].name == dist_pls_pls[i].destination:
                            # print 'possible receiver =',sim.stations[j].name,'; distance
    =',dist_pls_pls[i].value
                            if (dist_pls_pls[i].value < minimal):
                                minimal = dist_pls_pls[i].value
                                receiver = dist_pls_pls[i].destination

            # if there is a supporting station, updates all the picav and stations' attributes and
    counters
            if ((misser != -1)&(receiver != -1)):
            self.rt = self.rt + 1
            # print 'The relocation is now beginning. A vehicle is send from the
    station',misser,'to the station',receiver
```

```
            # print 'total relocations',self.rt
            # print 'impossible relocations',self.rl
            # print 'and they are distant metres',minimal
            for k in range(sim.number_stations):
                # updates the counter of the vehicles available at the current station and at the
supporting station
                if sim.stations[k].name == misser:
                    if sim.problem == 1:
                        sim.stations[k].n_vehicles = sim.stations[k].n_vehicles - 1
                        sim.stations[k].n_veh_relocation = sim.stations[k].n_veh_relocation - 1
                        self.n_veh_relocation = self.n_veh_relocation + 1
                        # print 'the misser is the station',misser,' whereas I decurt a vehicle from the
station',sim.stations[k].name
                if sim.stations[k].name == receiver:
                    if sim.problem == 2:
                        self.n_vehicles = self.n_vehicles - 1
                        self.n_veh_relocation = self.n_veh_relocation - 1
                        # print 'the receiver is the station',receiver,' whereas I add a vehicle to the
station',sim.stations[k].name

            # determines the vehicle which is relocating and assigns to it the relocation time
            for i in range(sim.number_picav):
                if (sim.vehicles[i].state == 0)&(sim.vehicles[i].parking == misser):
                    sim.vehicles[i].state = 3
                    sim.vehicles[i].parking = 9999999
                    sim.vehicles[i].unit = 9999999
                    sim.vehicles[i].distance = minimal
                    if sim.problem == 1:
                        sim.vehicles[i].destination = self.name
                    elif sim.problem == 2:
                        sim.vehicles[i].destination = receiver
                    if (t < TI2_min):
                        sim.vehicles[i].relocation_time =
round(((sim.vehicles[i].distance)/(sim.vel_autom_morn*60))+t)
                    elif (t < TI2_max):
                        sim.vehicles[i].relocation_time =
round(((sim.vehicles[i].distance)/(sim.vel_autom_aft*60))+t)
                    else:
                        sim.vehicles[i].relocation_time =
round(((sim.vehicles[i].distance)/(sim.vel_autom_even*60))+t)
                    if sim.vehicles[i].relocation_time == t:
```

```
                    sim.vehicles[i].relocation_time = t+1
                sim.total_relocation_time = sim.total_relocation_time +
sim.vehicles[i].relocation_time - t
                sim.number_relocations = sim.number_relocations + 1
                # print 'the vehicle',sim.vehicles[i].name,'relocates now between the
stations',misser,'and',receiver
                sim.vehicles[i].problem_relocation = sim.problem
                # print 'will finish relocating at time instant',sim.vehicles[i].relocation_time
                break

        # if the case of study is the 1 the relocation is impossible
        elif (misser == -1)&(sim.problem == 1)&(case_study == 1):
            # print 'impossible relocation'
            self.rt = self.rt + 1
            self.rl = self.rl + 1

        # if the case study is 3 checks if it is possible to get some vehicle from a unit
        elif (misser == -1)&(sim.problem == 1)&(case_study == 3):
            # print 'impossible relocation. Try if some vehicle is available from the units'
            self.rt = self.rt + 1
            self.rl = self.rl + 1
            distdist = 999999
            index = -1

            # checks if some vehicle is at a unit and takes the nearest one.
            for i in range(sim.number_picav):
                if (sim.vehicles[i].state == 0)&(sim.vehicles[i].unit < 99999):
                    originprovv = sim.vehicles[i].unit
                    destinationprovv = self.name
                    for j in range(len(dist_points_pls)):
                        if (dist_points_pls[j].origin == originprovv)&(dist_points_pls[j].destination
== destinationprovv):
                            distprovv = dist_points_pls[j].value
                            if distprovv < distdist:
                                distdist = distprovv
                                index = sim.vehicles[i].unit
                    originprovv = 999999
                    destinationprovv = 999999
            # print 'origin unit',index
        # If there is a supporting unit, updates the number of vehicles at the supporting unit
and at the current station
            for j in range(number_unit):
                if (sim.unit[j].name == index)&(index > -1):
                    sim.unit[j].n_vehicles = sim.unit[j].n_vehicles - 1
            if (index > -1)&(index < 99999):
                self.n_veh_relocation = self.n_veh_relocation + 1

        # calculate the distance from the parking lot and the unit. It doesnt use the method
"calculate travel time" because here the velocity considered is the one
        # for the automatic movement of the picav.
            for i in range(sim.number_picav):
                if (index == sim.vehicles[i].unit)&(index != -1):
                    sim.vehicles[i].origin = sim.vehicles[i].unit
                    sim.vehicles[i].destination = self.name
                    for j in range(len(dist_points_pls)):
                        if (dist_points_pls[j].origin ==
sim.vehicles[i].origin)&(dist_points_pls[j].destination == sim.vehicles[i].destination):
                            sim.vehicles[i].distance = dist_points_pls[j].value
                            sim.vehicles[i].slope = pend_points_pls[j].value
                            break
                    # print 'origin unit',sim.vehicles[i].origin,', destination
pl',sim.vehicles[i].destination,'. The unit is distant',sim.vehicles[i].distance
                        # distance: in metres; speed: in metres/seconds.
                        # travel time: in minutes
                    if (t < TI2_min):
                        sim.vehicles[i].travel_time = round(sim.vehicles[i].distance /
(sim.vel_autom_morn*60))
                    elif (t <= TI2_max):
                        sim.vehicles[i].travel_time = round(sim.vehicles[i].distance /
(sim.vel_autom_aft*60))
                    else:
                        sim.vehicles[i].travel_time = round(sim.vehicles[i].distance /
(sim.vel_autom_even*60))
                    if sim.vehicles[i].travel_time == 0:
                        sim.vehicles[i].travel_time = 1
                    sim.vehicles[i].travel_time = sim.vehicles[i].travel_time + t
                    # print 'travel time',sim.vehicles[i].travel_time
                    if sim.vehicles[i].travel_time > 99999:
                        print 'error'
                        stop
                sim.vehicles[i].unit = 999999
                sim.vehicles[i].state = 3
                sim.vehicles[i].relocation_time = sim.vehicles[i].travel_time
                break


    #end of relocation. Reinitialization of all the picav parameters and recall of the method
"arrival vehicle" of the picav
    def end_relocation(self,picav,t,sim):
        # print 'the picav',picav.name,'has now finished relocating and currently is at the
station',self.name
        if self.name != picav.destination:
            print 'error: the picav has gone to the wrong station'
        picav.parking = self.name
        picav.unit = 999999
        self.final_time = 99999999
        picav.activity_time = 99999999
        picav.destination = 99999999
        picav.origin = 99999999
        picav.user = 99999999
        picav.provenience = 9999999
        picav.typology_trip = 0
        picav.reapparance_place = 9999999
        if picav.problem_relocation == 2:
            self.n_veh_relocation = self.n_veh_relocation + 1
        self.arrival_vehicle(picav,t,sim)




class supervisor:

    def choice_final_destination(self,picav,sim,t):

        ppl = -1
        # print '--'
        if picav.activity_pattern == 1: # call of the supervisor from the last stop of the trip
chain
            picav.origin = picav.reapparance_place
            # print 'vehicle',picav.name,'calls supervisor from the unit',picav.reapparance_place
            picav.calculate_parking_lot(sim,t) # calculates the distance from the current unit to
all stations in the user's choice set
```

```
            if picav.reapparance_place > 1000:
                print 'error'
                error
        elif picav.activity_pattern == 2: # call of the supervisor from the origin parking lot
            picav.reapparance_place = picav.origin
            # print 'vehicle',picav.name,'calls supervisor from the parking lot',picav.origin
            picav.calculate_parking_lot(sim,t) # calculates the distance from the current station
to all the other stations in the user's choice set
            if picav.reapparance_place > 1000:
                print 'error'
                error
        # print 'provenience of the vehicle',picav.provenience
        sim.calc_max_waiting(t) # calculates the maximum waiting time at each station
        flag = 0
        idt = picav.provenience - 1
        stations_provv = sim.choice_sets[idt]
        ttp_waiting = 0
        # takes the best station, i.e. the one for which the ratio: maximum waiting time / travel
time , is the greatest
        for k in range(len(stations_provv)):
            for j in range(number_stations):
                if (sim.stations[j].name == stations_provv[k])&(sim.stations[j].name !=
picav.origin):
                    # print 'origin pl or unit',picav.origin,', destination pl',sim.stations[j].name,'.
Distance mins',sim.stations[j].travel_time,'wait
time',sim.stations[j].max_waiting_time,'vehicles',sim.stations[j].n_vehicles
                    if sim.stations[j].max_waiting_time / sim.stations[j].travel_time > ttp_waiting:
                        ppl = sim.stations[j].name
                        ttp_waiting = sim.stations[j].max_waiting_time/sim.stations[j].travel_time
                        flag = 1
        # otherwise takes the station for which the product: number vehicles available * travel
time, is the minimum
        if ppl == -1:
            nveicc = 100000
            for k in range(len(stations_provv)):
                for j in range(number_stations):
                    if (sim.stations[j].name == stations_provv[k])&(sim.stations[j].name !=
picav.origin):
                        if sim.stations[j].n_vehicles*sim.stations[j].travel_time < nveicc:
                            nveicc = sim.stations[j].n_vehicles * sim.stations[j].travel_time
                            flag = 2
                    ppl = sim.stations[j].name
        picav.destination = ppl
        if picav.destination == -1:
            error
        if flag == 0:
            error
        picav.origin = picav.reapparance_place
        # CALCULATES THE TRAVEL TIME from the current user's position and the
chosen destination station
        if picav.activity_pattern == 1:
            picav.calculate_travel_time(sim,t)
        elif picav.activity_pattern == 2:
            picav.travel_time_pls(sim,t)
        picav.travel_time = picav.travel_time + t
        picav.activity_time = picav.travel_time
        if picav.travel_time > 10000:
            error
        # print 'the picav vehicle has been assigned to station',picav.destination,'and it will
take',picav.travel_time
        if (picav.travel_time > 10000)|(picav.activity_time > 10000):
            # print picav.travel_time, picav.activity_time, 'error'
            stop
        for j in range(number_users):
            if (sim.users[j].name == picav.user)&(picav.user < 99999):
                sim.users[j].destination = picav.destination
                sim.users[j].travel_time = picav.travel_time
                sim.users[j].activity_time = picav.activity_time
# Now it updates the state of the user who has been assigned the picav vehicle
        max_waiting_time = 0
        index = -1
        if flag == 1:
            for k in range(number_users):
                if (sim.users[k].queue == picav.destination)&(sim.users[k].state ==
2)&(picav.destination < 99999):
                    provv = t - sim.users[k].time_entrance_queue
                    if max_waiting_time < provv:
                        max_waiting_time = provv
                        index = sim.users[k].name
            sim.users[index].state = 4
```

```
class sections:
    n_vehicles = 0
    destination = 999999
    name = 999999
    travel_time = 999999
    origin = 999999
    typology_misser = 999999
    picav = 999999

    # calculates the distances among the units
    def calculate_distance_unit(self,sim,t):
        for j in range(len(dist_points_points)):
            if (dist_points_points[j].origin == self.origin)&(dist_points_points[j].destination ==
self.destination)&(self.origin < 99999)&(self.destination < 99999):
                self.distance = dist_points_points[j].value
                self.slope = pend_points_points[j].value
                if (t < TI2_min):
                    self.travel_time = round(self.distance / (sim.vel_morning*60))
                elif (t <= TI2_max):
                    self.travel_time = round(self.distance / (sim.vel_aftern*60))
                else:
                    self.travel_time = round(self.distance / (sim.vel_evening*60))
                if self.travel_time == 0:
                    self.travel_time = 1
                # print 'travel time between the units',self.travel_time
                break

    # calculates the distances from a station to an unit
    def calculate_distance_stations(self,sim,t):
        # print 'origin',self.origin,'destination',self.destination
        for j in range(len(dist_pls_points)):
            if (dist_pls_points[j].origin == self.origin)&(dist_pls_points[j].destination ==
self.destination)&(self.origin < 99999)&(self.destination < 99999):
                self.distance = dist_pls_points[j].value
                self.slope = pend_pls_points[j].value
                if (t < TI2_min):
                    self.travel_time = round(self.distance / (sim.vel_morning*60))
                elif (t <= TI2_max):
                    self.travel_time = round(self.distance / (sim.vel_aftern*60))
            else:
                self.travel_time = round(self.distance / (sim.vel_evening*60))
            if self.travel_time == 0:
                self.travel_time = 1
            # print 'travel time between the units',self.travel_time
            break

    # calculates the distance from a unit to a station. Similar to calculate travel time but here
it involves the units and not the vehicles
    def calculate_distance_stations_reverse(self,sim,t):
        flag = 0
        for j in range(len(dist_points_pls)):
            if (dist_points_pls[j].origin == self.origin)&(dist_points_pls[j].destination ==
self.destination)&(self.origin < 99999)&(self.destination < 99999):
                self.distance = dist_points_pls[j].value
                self.slope = pend_points_pls[j].value
                if (t < TI2_min):
                    self.travel_time = round(self.distance / (sim.vel_morning*60))
                elif (t <= TI2_max):
                    self.travel_time = round(self.distance / (sim.vel_aftern*60))
                else:
                    self.travel_time = round(self.distance / (sim.vel_evening*60))
                if self.travel_time == 0:
                    self.travel_time = 1
                # print 'travel time between the units',self.travel_time
                flag = 1
                break
        if flag == 0:
            print 'impossible to calculate travel time. Information missing'
            stop

    # arrival of the user at the unit
    def arrival_user(self,user,t,sim):
        # print 'the user',user.name,'is now arrived at the unit',self.name
        if (self.n_vehicles > 0):
            for j in range(sim.number_picav):
                if (sim.vehicles[j].state == 0)&(sim.vehicles[j].unit ==
self.name)&(sim.vehicles[j].unit < 99999):
                    self.n_vehicles = self.n_vehicles - 1
                    # print '--'
                    # print 'subtraction of a vehicle at the unit',self.name
                sim.vehicles[j].occupation(user,t,sim)
                break
        # no vehicles available at the current unit
        else:
            # print '--'
            # print 'no vehicles available, the request of a vehicle is generated from the
unit',self.name
            user.unit = self.name
            user.state = 2
            user.time_entrance_queue = t
            product = 0
            timetravel = 999999
            index = -1
        # checks if there are some vehicles at the units
            for j in range(sim.number_unit):
                if sim.unit[j].n_vehicles > 0:
                    self.origin = sim.unit[j].name
                    self.destination = self.name
                    # print 'unit of origin',self.origin,'number vehicles',sim.unit[j].n_vehicles
                    self.calculate_distance_unit(sim,t)
                    if (1/self.travel_time) > product:
                        timetravel = self.travel_time
                        index = sim.unit[j].name
                        product = 1/self.travel_time
            timetravel_unit = timetravel
            unit_misser = index
            if index == -1:
            # checks if there are some vehicles at the stations
                for j in range(sim.number_stations):
                    if sim.stations[j].n_vehicles > 0:
                        self.origin = sim.stations[j].name
                        self.destination = self.name
                        # print 'station of origin',self.origin,'number
vehicles',sim.stations[j].n_vehicles
                        self.calculate_distance_stations(sim,t)
                        if (1/self.travel_time) > product:
                            timetravel = self.travel_time
                            index = sim.stations[j].name
                            product = 1/self.travel_time
                timetravel_stations = timetravel
                station_misser = index
            # updates the user's attributes
                if index != -1:
                    self.travel_time = timetravel_stations
                    self.origin = station_misser
                    self.typology_misser = 2
                    user.state = 4
                    # print 'station misser',station_misser
                else:
                    # print 'the user must wait, no vehicles available in the whole area in this
moment'
                    user.state = 2
            else:
                self.travel_time = timetravel_unit
                self.origin = unit_misser
                self.typology_misser = 1
                user.state = 4
                # print 'unit misser',unit_misser

            # the user begins waiting at the unit
            nnn = self.name
            sim.queue_unit[nnn].append(user)
            # print 'the user',user.name,'is now in queue at the unit',self.name
            if index != -1:
                self.travel_time = self.travel_time + t
                # if the misser is a unit, it updates the attributes of the unit misser and of the
current unit (receiver).
                # Determines the vehicle which will reach the user and updates its attributes
                if self.typology_misser == 1:
                    for j in range (sim.number_unit):
                        if (sim.unit[j].name == self.origin)&(self.origin < 99999):
                            sim.unit[j].n_vehicles = sim.unit[j].n_vehicles - 1
                            # print 'a vehicle will arrive at the user place at time',self.travel_time,'from
the unit',self.origin
                            user.state = 4
                            chck = -1
                            for i in range(sim.number_picav):
                                if (sim.vehicles[i].unit == self.origin)&(self.origin <
99999)&(sim.vehicles[i].state == 0):
                                self.destination = self.name
                                if self.destination == -1:
                                    print 'error on destination; method arrival user'
                        stop
                      # print 'recall from arrival user'
                      sim.vehicles[i].parking = 999999
                      sim.vehicles[i].unit = 999999
                      sim.vehicles[i].typology_destination = 2
                      sim.vehicles[i].begin_approach(sim,self) # the vehicle is now
approaching
                      chck = sim.vehicles[i].name
                      break
                  if chck == -1:
                    print 'wrrore1!!!'
                    print 'unit',self.origin
                    stop
            # if the misser is a station, it updates the attributes of the station misser and of the
current unit (receiver).
            # Determines the vehicle which will reach the user and updates its attributes
            elif self.typology_misser == 2:
              for j in range (sim.number_stations):
                if (sim.stations[j].name == self.origin)&(self.origin < 99999):
                  sim.stations[j].n_vehicles = sim.stations[j].n_vehicles - 1
                  sim.stations[j].n_veh_relocation = sim.stations[j].n_veh_relocation - 1
                  if sim.stations[j].n_vehicles < 0:
                    print 'error severe',sim.stations[j].n_vehicles
                  # print 'a vehicle will arrive at the user place at time',self.travel_time,'from
the station',self.origin
                  user.state = 4
                  chck = -1
                  for i in range(sim.number_picav):
                    if (sim.vehicles[i].parking == self.origin)&(self.origin <
99999)&(sim.vehicles[i].state == 0):
                      sim.vehicles[i].parking = 999999
                      sim.vehicles[i].unit = 9999999
                      self.destination = self.name
                      if self.destination == -1:
                        print 'error on destination; method arrival user'
                        stop
                      # print 'recall from arrival user'
                      sim.vehicles[i].typology_destination = 2
                      sim.vehicles[i].begin_approach(sim,self) # the vehicle is now
approaching
                      chck = sim.vehicles[i].name
```

```
                                break
                            if chck == -1:
                                print 'wrrore2!!!'
                                print 'station',self.origin
                                stop
                            break
            elif index == -1: # if there is no vehicle available anywhere, the user simply waits
                self.origin = 999999
                self.destination = 999999
                # print 'the user stays in the unit',self.name,'waiting'


    def arrival_vehicle(self,picav,t,sim):  # the vehicle arrives at the unit
        picav.user = 999999
        picav.unit = self.name
        picav.parking = 999999
        # print '---'
        # print 'the picav',picav.name,'is now arrived at unit',self.name
        www = self.name
        if picav.level_charge >= picav.minimal_charge:
            self.n_vehicles = self.n_vehicles + 1
            if len(sim.queue_unit[www]) == 0:
                time_waiting = 0
                index = 999999
                name_provv = 999999
    # if there are users waiting at a unit, takes the user who has been in queue for longer and
a vehicle is sent to him
                for j in range(sim.number_users):
                    if (sim.users[j].unit < 99999)&(sim.users[j].parking >
99999)&(sim.users[j].state == 2):
                        timewaiting = t - sim.users[j].time_entrance_queue
                        if timewaiting > time_waiting:
                            time_waiting = timewaiting
                            index = sim.users[j].unit
                            name_provv = sim.users[j].name
    # the receiving unit has been determined above. If a receiving unit exists, the attribute of
the current unit are updated,
    # as well as the attributes of the picav vehicle
                if (index != 999999)&(index != self.name):
                    self.origin = self.name
                    self.destination = index
                    self.calculate_distance_unit(sim,t)
                    self.travel_time = self.travel_time + t
                    for k in range(sim.number_users):
                        if (name_provv == sim.users[k].name)&(name_provv < 99999):
                            sim.users[k].state = 4
                    # print 'recall from arrival vehicle'
                    # print 'origin unit',self.name,'destination unit',self.destination
                    self.n_vehicles = self.n_vehicles - 1
                    picav.unit = 999999
                    picav.parking = 999999
                    picav.typology_destination = 2
                    picav.begin_approach(sim,self)
    # if there are users waiting at a station, takes the user who has been in queue for longer
and a vehicle is sent to him
                else:
                    index = 999999
                    time_waiting = 0
                    for k in range(sim.number_users):
                        if (sim.users[k].unit > 99999)&(sim.users[k].parking <
99999)&(sim.users[k].state == 2):
                            timewaiting = t - sim.users[k].time_entrance_queue
                            if timewaiting > time_waiting:
                                time_waiting = timewaiting
                                index = sim.users[k].parking
                                name_provv = sim.users[k].name
    # the receiving station has been determined above. If a receiving station exists, the
attribute of the current unit are updated,
    # as well as the attributes of the picav vehicle
                    if (index != 999999)&(index != self.name):
                        self.origin = self.name
                        self.destination = index
                        self.calculate_distance_stations_reverse(sim,t)
                        self.travel_time = self.travel_time + t
                        for f in range(sim.number_users):
                            if (name_provv == sim.users[f].name)&(name_provv < 99999):
                                sim.users[f].state = 4
                        # print 'recall from arrival vehicle'
                        # print 'origin unit',self.name,'destination station',self.destination
                        self.n_vehicles = self.n_vehicles - 1
                        picav.unit = 999999
                        picav.parking = 999999
                    picav.typology_destination = 1
                    picav.begin_approach(sim,self)
    # if there are users in the current unit, no need is to send the vehicle and the picav is
occupied again
            www = self.name
            if len(sim.queue_unit[www])>0:
                nnn = self.name
                u = sim.queue_unit[nnn].pop(0)
                for j in range(sim.number_users):
                    if (sim.users[j].name == u.name)&(u.name < 99999):
                        sim.users[j] = u
                        sim.users[j].time_queue = t - u.time_entrance_queue
                        if (u.time_entrance_queue == 0)|(u.time_entrance_queue > 99999):
                            print 'errore grave'
                            print u.name,u.time_entrance_queue,u.unit
                            stop
                        # print 'the picav is now occupied again'
                        picav.unit = 9999999
                        picav.parking = 9999999
                        self.n_vehicles = self.n_vehicles - 1
                        picav.occupation(sim.users[j],t,sim)
                        break
    # if the state of charge of the vehicle is not enough, the vehicle is redirected to the nearest
station
        else:
            self.origin = self.name
            print 'not enough charge. The picav must be redirected to a station and put in charge'
            distmin = 9999999
            self.destination = -1
            for j in range(sim.number_stations):
                index = sim.stations[j].name
                for k in range(len(dist_points_pls)):
                    if (dist_points_pls[k].origin == self.origin)&(dist_points_pls[k].destination ==
index):
                        distprovv = dist_points_pls[k].value
                        break
                if distprovv < distmin:
                    self.destination = index
                    distmin = distprovv
            if self.destination == -1:
                print 'error no charging station'
            else:
                self.calculate_distance_stations_reverse(sim,t) #calculates the travel time to the
supporting station
                picav.origin = picav.unit
                picav.unit = 9999999
                picav.state = 4
                picav.destination = self.destination
                self.travel_time = self.travel_time + t
                picav.redirection_time = self.travel_time
                picav.distance = self.distance
                picav.slope = self.slope
                # print 'the picav is redirected to recharge at the station',picav.destination
                # print 'the picav will arrive there at time',picav.redirection_time
```

The file chiamasimulatore.py

```
# -*- coding: cp1252 -*-
from numpy import *
from numpy.random import *
from numpy.oldnumeric.random_array import *
from simulatore import *

ss=simulator()
ss.simulation()

import csv


parking = csv.writer(open('PP.csv','wb'),
delimiter=',',quotechar='|', quoting=csv.QUOTE_NONE)
parking.writerow(('station id','time instant','n°veh
available','max_waiting_time','n°users_queue'))
for i in range(ss.number_stations):
    n_veh_available = ss.number_vehicles_available[i]
    max_time_queue = ss.max_time_queue[i]
    number_users_queue = ss.number_users_queue[i]
    for j in range(len(n_veh_available)):

parking.writerow((i+1,'',j+TI1_min,'',n_veh_available[j],'',m
ax_time_queue[j],'',number_users_queue[j]))
```

```
queue = csv.writer(open('Users.csv','wb'),
delimiter=',',quotechar='|', quoting=csv.QUOTE_NONE)
queue.writerow(('identifier of user','time spent in queue'))
for j in range(ss.number_users):

queue.writerow((ss.users[j].name,'',ss.users[j].time_queue))

picav = csv.writer(open('Picav.csv','wb'), delimiter=',')
picav.writerow(('time instant','n°veh available','n°veh
occupied','n°veh in charge','n°veh relocating','n° veh
redirected for full station','n°veh approaching user'))
for j in range(len(ss.number_picav_state0)):

picav.writerow((j+TI1_min,'',ss.number_picav_state0[j],'',ss.
number_picav_state1[j],'',ss.number_picav_state2[j],'',ss.num
ber_picav_state3[j],'',ss.number_picav_state4[j],'',ss.number
_picav_state5[j]))
```

Attachment B. Code of the optimization algorithm

```
# -*- coding: cp1252 -*-
from numpy import *
from numpy.random import *
from numpy.oldnumeric.random_array import *
from simulatorefinale import *

# ---------------- INPUT DATA ------------------



flag = 0
# flag is an indicator which defines whether to act on thresholds or on the fleet in case study
1 or 3.
# flag is set equal to 0 for case of study 2 as thresholds are not relevant
# flag is set equal to 1 if we are going to modify the fleet dimension and 2 if we are going
to modify thresholds

NPP = [11,11,11,11,11,11,11] # Number of vehicles in each station

low_critical_thresholds = [1,1,1,1,1,1,1]  # critical thresholds
low_buffer_thresholds = [7,7,7,7,7,7,7] # buffer thresholds
print 'low critical thresholds =',low_critical_thresholds
print 'low buffer thresholds =',low_buffer_thresholds
print 'fleet =',NPP

cost_waiting = 0.10  # euro per minute
cost_vehicle = 4.117  # euro per vehicle
cost_relocation = 0.01 # euro per minute
constant = 20.82 # euro, due to energy consumption and management
mu = 1000.0 # penalty coefficient
memory = []

print 'MU =',mu

number_stations = len(NPP)

# ------------ INITIALIZATION ---------------

number_picav = 0
for j in range(number_stations):
    number_picav = number_picav + NPP[j]

count = 0
lists1 = []
percentile_95 = 0
```

```
percentile_90 = 0
percentile_50 = 0
objective_function = 0
total_wait_cost = 0
total_relocation_cost = 0

# Calculation of the objective function and percentiles.
# this process is iterated 30 times in order to eliminate random effects

for w in range(30):
    print w
    ss=simulator()
    ss.simulation(NPP,low_critical_thresholds,low_buffer_thresholds)
    lists1.append(ss.lista)
    obj_fun = constant + cost_waiting*ss.total_queuetime + cost_vehicle*number_picav +
cost_relocation*ss.total_relocation_time
    c_wait = cost_waiting*ss.total_queuetime
    c_rel = cost_relocation*ss.total_relocation_time
    percentile_95 = percentile_95 + ss.percentile_95
    percentile_90 = percentile_90 + ss.percentile_90
    percentile_50 = percentile_50 + ss.percentile_50
    objective_function = objective_function + obj_fun
    total_wait_cost = total_wait_cost + c_wait
    total_relocation_cost = total_relocation_cost + c_rel

percentile_95 = percentile_95 / 30.0
percentile_90 = percentile_90 / 30.0
percentile_50 = percentile_50 / 30.0
objective_function = objective_function / 30.0
total_wait_cost = total_wait_cost / 30.0
total_relocation_cost = total_relocation_cost / 30.0


# Update of the objective function with the penalty contribution (penalty contribution of
constraints)
if percentile_95 > (600.0/60.0):
    objective_function = objective_function + mu*cost_waiting*((percentile_95 -
(600.0/60.0))*(percentile_95 - (600.0/60.0)))
    print 'constraints not satisfied'
print '95th percentile =',percentile_95
if percentile_90 > (480.0/60.0):
    objective_function = objective_function + mu*cost_waiting*((percentile_90 -
(480.0/60.0))*(percentile_90 - (480.0/60.0)))
    print 'constraints not satisfied'
print '90th percentile =',percentile_90
if percentile_50 > (240.0/60.0):
    objective_function = objective_function + mu*cost_waiting*((percentile_50 -
(240.0/60.0))*(percentile_50 - (240.0/60.0)))
    print 'constraints not satisfied'
print '50th percentile =',percentile_50

print 'objective function', objective_function
print 'total_wait_cost',total_wait_cost
print 'total_relocation_cost',total_relocation_cost

f_ob1 = objective_function  # determines the objective function necessary for the cooling
schedule


# --------------- FIRST ITERATION -----------------

print '------------------------'
# recalls the number of vehicles and thresholds determined at the previous iteration
NPP_old = []
old_crit_thresholds = []
old_buffer_thresholds = []
number_picav_old = number_picav
for j in range(number_stations):
    old_crit_thresholds.append(low_critical_thresholds[j])
    old_buffer_thresholds.append(low_buffer_thresholds[j])
    NPP_old.append(NPP[j])

if (flag == 2)&((case_study == 1)|(case_study == 3)):

    # decides whether to modify low critical thresholds or low buffer thresholds
    i = uniform(0,1)
    if i >= 0.5:
        print 'act on critical thresholds'
        j = uniform(0,1)
        if ((j < 0.5)|(low_critical_thresholds[0] >= (NPP[0]-
1)))&(low_critical_thresholds[0]>0):
            for k in range(number_stations):
                low_critical_thresholds[k] = low_critical_thresholds[k]-1
            print 'decrease by one'
        else:
            for k in range(number_stations):
                low_critical_thresholds[k] = low_critical_thresholds[k]+1
            print 'increase by one'

    else:
        print 'act on buffer thresholds'
        j = uniform(0,1)
        if ((j < 0.5)|(low_buffer_thresholds[0] >= (NPP[0]-
1)))&(low_buffer_thresholds[0]>(low_critical_thresholds[0]+1)):
            for w in range(number_stations):
                low_buffer_thresholds[w] = low_buffer_thresholds[w]-1
            print 'decrease by one'
        else:
            for w in range(number_stations):
                low_buffer_thresholds[w] = low_buffer_thresholds[w]+1
            print 'increase by one'


if (flag == 1)&((case_study == 1)|(case_study == 3)):
    print 'act on fleet dimension'
    j = uniform(0,1)
    if j < 0.5:
        number_picav = number_picav - len(NPP)
        component = -1
        print 'decrease by',len(NPP),'the number of vehicles'
    else:
        number_picav = number_picav + len(NPP)
        component = 1
        print 'increase by',len(NPP),'the number of vehicles'

    for j in range(number_stations):
        NPP[j] = NPP[j] + component


if case_study == 2:
    k = int(uniform(0,9))
    if k == 9:
        k = 5
    j = uniform(0,1)
    if j < 0.5:
        NPP[k] = NPP[k]-1
        print 'decrease by 1 in position',k+1
    else:
        NPP[k] = NPP[k]+1
        print 'increase by 1 in position',k+1
    print 'new solution:',NPP
    number_picav = 0
    for j in range(number_stations):
        number_picav = number_picav + NPP[j]




print 'low critical thresholds =',low_critical_thresholds
print 'low buffer thresholds =',low_buffer_thresholds
print 'number of vehicles =',NPP


lists1 = []
percentile_95 = 0
percentile_90 = 0
percentile_50 = 0
objective_function = 0
total_wait_cost = 0
total_relocation_cost = 0


# Calculation of the objective function and percentiles.

for w in range(30):
    print w
    ss=simulator()
    ss.simulation(NPP,low_critical_thresholds,low_buffer_thresholds)
    lists1.append(ss.lista)
    obj_fun = constant + cost_waiting*ss.total_queuetime + cost_vehicle*number_picav +
cost_relocation*ss.total_relocation_time
    c_wait = cost_waiting*ss.total_queuetime
    c_rel = cost_relocation*ss.total_relocation_time
    percentile_95 = percentile_95 + ss.percentile_95
    percentile_90 = percentile_90 + ss.percentile_90
    percentile_50 = percentile_50 + ss.percentile_50
    objective_function = objective_function + obj_fun
    total_wait_cost = total_wait_cost + c_wait
    total_relocation_cost = total_relocation_cost + c_rel

percentile_95 = percentile_95 / 30.0
percentile_90 = percentile_90 / 30.0
percentile_50 = percentile_50 / 30.0
objective_function = objective_function / 30.0
total_wait_cost = total_wait_cost / 30.0
total_relocation_cost = total_relocation_cost / 30.0


# Updates the objective function with the constraints' penalty function
if percentile_95 > (600.0/60.0):
    objective_function = objective_function + mu*cost_waiting*((percentile_95 -
(600.0/60.0))*(percentile_95 - (600.0/60.0)))
    print 'constraints not satisfied'
print '95th percentile =',percentile_95
if percentile_90 > (480.0/60.0):
    objective_function = objective_function + mu*cost_waiting*((percentile_90 -
(480.0/60.0))*(percentile_90 - (480.0/60.0)))
    print 'constraints not satisfied'
print '90th percentile =',percentile_90
if percentile_50 > (240.0/60.0):
    objective_function = objective_function + mu*cost_waiting*((percentile_50 -
(240.0/60.0))*(percentile_50 - (240.0/60.0)))
    print 'constraints not satisfied'
print '50th percentile =',percentile_50

print 'objective function', objective_function
print 'total_wait_cost',total_wait_cost
print 'total_relocation_cost',total_relocation_cost

f_ob2 = objective_function  # determines the second objective function necessary for the
ooling schedule


# ------ DETERMINATION OF THE INITIAL TEMPERATURE AS REQUIRED BY
THE COOLING SCHEDULE ------

acceptance_ratio = 0.5
print 'acceptance_ratio',acceptance_ratio


if f_ob2 < f_ob1:    # Determination of initial temperature
    diffp = f_ob1-f_ob2
else:
    diffp = f_ob2-f_ob1
temperat0 = diffp/log((1/acceptance_ratio))

print 'initial temperature',temperat0

f_ob = []
f_ob.append(f_ob1)
temperat = []
temperat.append(temperat0)

memory_critical = []
memory_buffer = []
memory_fleet = []

if f_ob2 < f_ob1:       # determines which solution will be the best departure point for the
next iteration
    print 'solution 2 is better than solution 1 and therefore will be the departure point for the
next iteration'
    f_ob.append(f_ob2)
    NPP_old = []
    old_crit_thresholds = []
    old_buffer_thresholds = []
    number_picav_old = number_picav
    for j in range(number_stations):
        old_crit_thresholds.append(low_critical_thresholds[j])
        old_buffer_thresholds.append(low_buffer_thresholds[j])
        NPP_old.append(NPP[j])

else:
    nnnnn = uniform(0,1)
    if nnnnn < 0.5:
        print 'accept the new solution; n=',nnnnn
        f_ob.append(f_ob2)
        old_crit_thresholds = []
        old_buffer_thresholds = []
        NPP_old = []
        number_picav_old = number_picav
        for j in range(number_stations):
            old_crit_thresholds.append(low_critical_thresholds[j])
            old_buffer_thresholds.append(low_buffer_thresholds[j])
            NPP_old.append(NPP[j])
    else:
        print 'the previous solution is the good one; n=',nnnnn
        low_critical_thresholds = []
        low_buffer_thresholds = []

        NPP = []
        number_picav = number_picav_old
        for j in range(number_stations):
            low_critical_thresholds.append(old_crit_thresholds[j])
            low_buffer_thresholds.append(old_buffer_thresholds[j])
            NPP.append(NPP_old[j])
for j in range(number_stations):
    memory_critical.append(low_critical_thresholds[j])
    memory_buffer.append(low_buffer_thresholds[j])
    memory_fleet.append(NPP[j])


# -------------------------------------------------------------------------
# ----------------------- BODY OF THE ALGORITHM ----------------------------
# -------------------------------------------------------------------------

local_optimum = 0
for m in range (500):
    print '------------------------'
    print 'iteration =',m
    lists1 = []


    if (flag == 2)&((case_study == 1)|(case_study == 3)):

        # decides whether to modify low critical thresholds or low buffer thresholds
        i = uniform(0,1)
        if i >= 0.5:
            print 'act on critical thresholds'
            j = uniform(0,1)
            if ((j < 0.5)|(low_critical_thresholds[0] >= (NPP[0]-
1)))&(low_critical_thresholds[0]>0):
                for k in range(number_stations):
                    low_critical_thresholds[k] = low_critical_thresholds[k]-1
                print 'decrease by one'
            else:
                for k in range(number_stations):
                    low_critical_thresholds[k] = low_critical_thresholds[k]+1
                print 'increase by one'
            if low_critical_thresholds == memory_critical:
                print 'follows'
                continue

        else:
            print 'act on buffer thresholds'
            j = uniform(0,1)
            if ((j < 0.5)|(low_buffer_thresholds[0] >= (NPP[0]-
1)))&(low_buffer_thresholds[0]>(low_critical_thresholds[0]+1)):
                for w in range(number_stations):
                    low_buffer_thresholds[w] = low_buffer_thresholds[w]-1
                print 'decrease by one'
            else:
                for w in range(number_stations):
                    low_buffer_thresholds[w] = low_buffer_thresholds[w]+1
                print 'increase by one'
            if low_buffer_thresholds == memory_buffer:
                print 'follows'
                continue


    if (flag == 1)&((case_study == 1)|(case_study == 3)):
        print 'act on fleet dimension'
        j = uniform(0,1)
        if (j < 0.5)&(NPP[0]>low_buffer_thresholds[0]):
            number_picav = number_picav - len(NPP)
            component = -1
            print 'decrease by',len(NPP),'the number of vehicles'
        else:
            number_picav = number_picav + len(NPP)
            component = 1
            print 'increase by',len(NPP),'the number of vehicles'

        for j in range(number_stations):
            NPP[j] = NPP[j] + component

        if NPP == memory_fleet:
            print 'follows'
            continue


    if case_study == 2:
        k = int(uniform(0,9))
        if k == 9:
            k = 5
        j = uniform(0,1)
        if (j < 0.5)&(NPP[k]>low_buffer_thresholds[k]):
            NPP[k] = NPP[k]-1
            print 'decrease by 1 in position',k+1
        else:
            NPP[k] = NPP[k]+1
            print 'increase by 1 in position',k+1
        print 'new solution:',NPP
        number_picav = 0
        for j in range(number_stations):
            number_picav = number_picav + NPP[j]

        if NPP == memory_fleet:
            print 'follows'
            continue



    print 'low critical thresholds =',low_critical_thresholds
    print 'low buffer thresholds =',low_buffer_thresholds
    print 'number of vehicles =',NPP


    # Calculates the objective function

    lists1 = []
    percentile_95 = 0
    percentile_90 = 0
    percentile_50 = 0
    objective_function = 0
```

```
    total_wait_cost = 0
    total_relocation_cost = 0

    for w in range(30):
        print w
        ss=simulator()
        ss.simulation(NPP,low_critical_thresholds,low_buffer_thresholds)
        lists1.append(ss.lista)
        obj_fun = constant + cost_waiting*ss.total_queuetime + cost_vehicle*number_picav +
cost_relocation*ss.total_relocation_time
        c_wait = cost_waiting*ss.total_queuetime
        c_rel = cost_relocation*ss.total_relocation_time
        percentile_95 = percentile_95 + ss.percentile_95
        percentile_90 = percentile_90 + ss.percentile_90
        percentile_50 = percentile_50 + ss.percentile_50
        objective_function = objective_function + obj_fun
        total_wait_cost = total_wait_cost + c_wait
        total_relocation_cost = total_relocation_cost + c_rel

    percentile_95 = percentile_95 / 30.0
    percentile_90 = percentile_90 / 30.0
    percentile_50 = percentile_50 / 30.0
    objective_function = objective_function / 30.0
    total_wait_cost = total_wait_cost / 30.0
    total_relocation_cost = total_relocation_cost / 30.0


    # updates the objective function with the penaility function
    if percentile_95 > (600.0/60.0):
        objective_function = objective_function + mu*cost_waiting*((percentile_95 -
(600.0/60.0))*(percentile_95 - (600.0/60.0)))
        print 'constraints not satisfied'
        print '95th percentile =',percentile_95
    if percentile_90 > (480.0/60.0):
        objective_function = objective_function + mu*cost_waiting*((percentile_90 -
(480.0/60.0))*(percentile_90 - (480.0/60.0)))
        print 'constraints not satisfied'
        print '90th percentile =',percentile_90
    if percentile_50 > (240.0/60.0):
        objective_function = objective_function + mu*cost_waiting*((percentile_50 -
(240.0/60.0))*(percentile_50 - (240.0/60.0)))
        print 'constraints not satisfied'
        print '50th percentile =',percentile_50

    print 'objective function', objective_function
    print 'total_wait_cost',total_wait_cost
    print 'total_relocation_cost',total_relocation_cost
    temperature = temperat[m-1]*0.9  # cooling schedule
    temperat.append(temperature)

    ll = size(f_ob)

    if local_optimum > 100:
        print 'Reached a local optimum. End of the algorithm'
        STOP

# ---------------------- Vaglio la funzione obiettivo -------------------------
    if f_ob[ll-1]>objective_function:
        print 'accept the new solution because the objective function is better'
        f_ob.append(objective_function)
        local_optimum = 0
        old_crit_thresholds = []
        old_buffer_thresholds = []
        NPP_old = []
        number_picav_old = number_picav
        for j in range(number_stations):
            old_crit_thresholds.append(low_critical_thresholds[j])
            old_buffer_thresholds.append(low_buffer_thresholds[j])
            NPP_old.append(NPP[j])
    else:
        provv = exp((f_ob[ll-1]-objective_function)/temperature)
        nnnnn = uniform(0,1)
        if nnnnn < exp((f_ob[ll-1]-objective_function)/temperature):
            print 'accept the new solution'
            print 'acceptance_ratio',provv,' ,n:',nnnnn
            f_ob.append(objective_function)
            local_optimum = 0
            old_crit_thresholds = []
            old_buffer_thresholds = []
            number_picav_old = number_picav
            NPP_old = []
            for j in range(number_stations):
                old_crit_thresholds.append(low_critical_thresholds[j])
                old_buffer_thresholds.append(low_buffer_thresholds[j])
                NPP_old.append(NPP[j])
        else:
            print 'the previous solution is the good one; n:',nnnnn,'acceptance ratio:',provv
            low_critical_thresholds = []
            low_buffer_thresholds = []
            NPP = []
            number_picav = number_picav_old
            for j in range(number_stations):
                low_critical_thresholds.append(old_crit_thresholds[j])
                low_buffer_thresholds.append(old_buffer_thresholds[j])
                NPP.append(NPP_old[j])
            local_optimum = local_optimum + 1

    memory_critical = []
    memory_buffer = []
    memory_fleet = []

    for j in range(number_stations):
        memory_critical.append(low_critical_thresholds[j])
        memory_buffer.append(low_buffer_thresholds[j])
        memory_fleet.append(NPP[j])
```