Politecnico di Torino

Engineering and Management

MSc Thesis

Emergency humanitarian logistics: models and algorithms for evacuation planning

Supervisor:
Prof. Marco Ghirardi

Candidate:
Salvatore Caristo

Academic year 2018/2019

A Nonna Caterina e Zio Santo

Emergency humanitarian logistics: models and algorithms for evacuation planning

Salvatore Caristo

October 2019

Contents

Chapter 1

Introduction

This thesis deals with an emergency humanitarian problem aiming to plan and control the evacuation time of a building. The main tools used for handling the problem are linear programming models. Our implementation is based on models originally engineered for many problems like routing design and strategical positioning of hospitals, shipping centers, hubs, warehouses and in logistics. In the different proposed configurations, our mathematical model is able to optimize the evacuation time from the building, deciding the routing path for all the evacuees and their assigned emergency exits, with different variants on the evacuation path modelling. But it is also able, in a planning phase, to choose which subset of candidate positions for emergency exits is preferable in terms of expected evacuation time. The real case exploited to test this work is the emptying of Palazzo Camponeschi in L'Aquila during an earthquake. Several solution algorithms have been implemented and tested: both exact (the models themselves and their solution through the constraints generation techniques, often able to solve more efficiently models with a huge number of constraints), and heuristic (a variant of the constraint generation algorithm, designed for solving larger problem instances). The computer that runs the solver, indispensable for solving the mathematical programming models is an Asus core i5, 8th Gen, 1.8 GHZ with 8 Gb of RAM memory under Windows 10 Home 64-bits, and the solver is Xpress IVE with the Mosel language. The first chapter introduces the main components used in the following. Chapter 2 addresses the problem itself in detail, introducing the linear programming models and the parameter setting. Chapter 3 describes in practice the algorithms in support of the model resolution and finally, chapter 4, outlines the results.

1.1 Linear Programming, Combinatorial Optimization and Solver

The study of this thesis falls back in the wide field of Operational Research. To be more specific, the subject lies in a branch of Linear Programming that is Zero-one linear programming where some variables are restricted to be binary. The goal is to interpret a real problem and turning it into a system of inequalities and this process is known as Mathematical Programming (MP). Since the system can be very large and very difficult to solve, NP-Hard to put it in technical terms, some algorithms may be required to solve the problem within a reasonable time. Those used to solve the problem are classified into three groups [1]:

- Exact algorithms (see 1.2): they guarantee to find an optimal solution but may take an exponential number of iterations.
- Heuristic algorithms (see 1.3): they can't guarantee the quality of the solution, in fact, it may be sub-optimal. They don't guarantee to find the solution in a polynomial time, either.
- Approximation algorithms: they do guarantee a sub-optimal solution in a polynomial time through the setting of a bound on the degree of sub-optimality for example by stopping the algorithm after a certain time-limit.

In the first lines of this section, has not been said that, in addition to the inequalities, an objective function must be optimized. This leads to the definition of combinatorial optimization: it deals with problems where the best solution must be chosen among a finite (although large) number of alternatives. The best solution is the one dictated right by the objective function.

All this work is made possible by the use of a Mixed Integer Programming solver which employs technologies, a suite of pre-processing tools, algorithms and frameworks and possibly row and column generation functionality for reaching the target [2].

1.2 Constraint generation in theory

Besides the resolution of the whole problem, i.e. the system of inequalities, that is indeed an exact algorithm, under this family one more approach has been exploited which is Constraint Generation. This one, instead of solving the whole problem with a large number of constraints (and consequently with a run-time that may be large), solves a reduced problem (smaller in terms of number of constraints) and then a particular condition is checked: if the condition is violated, further constraints are added to the reduced problem and the check is repeated (... and so on, as long as the condition is violated), otherwise the solver stops with a solution (if there is one). In other words, constraints are added only if needed. Is this process advantageous? Perhaps, it depends on the number of times that the particular condition is violated and therefore the number of times that a smaller problem must be resolved.

1.3 Matheuristics in theory

As told, an exact approach can be very time consuming when the problem dimensions are large. In those cases, a lot of different heuristic algorithms have been proposed in order to find a "good enough" solution in a limited amount of time (greedy algorithms, local search, metaheuristics such as tabu search, genetic algorithms, etc.). More recently, matheuristics have been proposed. The main idea is to use a solver not with the aim to find the optimal solution, but in order to iteratively solve smaller sub-problems, through a heuristic scheme. A presentation of possible uses of an LP solver inside a heuristic algorithm can be found in [2].

Chapter 2

Problem

2.1 Description

On the night of April 6, 2009 a brutal earthquake rated 6.3 on the MMS shook the whole centre of Italy, devastated thousands of lives causing dead, wounded and refugees, pulverized hundreds of municipalities sparing nothing: houses, hospitals and schools leveled, as well as the components of the rich historical and artistic heritage like churches, basilicas and nobiliary buildings. Among the latter in L'Aquila, Palazzo Camponeschi, under study in this work: pure baroque style facade with eclectic taste and 14-th century interior. It suffered severe damages but not for the first time in fact the building has already suffered in 1703 during another devastating earthquake that deeply damaged the structure elements and therefore has been redesigned, restored and consolidated.

2.2 Fields of application

As made guess in the previous section, this problem can be and it is applied to the tragic event of an earthquake. Here is addressed the problem of evacuating a building as quickly as possible. In addition to this application the problem can be implemented to suit several other problems in humanitarian logistics for different types of disasters like fires for example in which somebody has to reach as soon as possible a shelter, a fire extinguisher etcetera. Other than that it could be exploited for designing routings and in distribution logistics could be applied for location problems where the issue is locating something like a warehouse, a shipping center, a hub or a hospital. In general can be adapted to any situation in which something has to move through a network towards something else in order to quickly meet a certain goal.

2.3 Mathematical models

In this section there will be exhibited three LP problems [5]: the first one and the simplest allows persons to evacuate without taking into account the real rules of a real evacuation. During the manifestation of such critical event in fact it is not possible to fork while leaving a unit cell and furthermore each arc can be travelled only in one and only direction (because it is practically impossible to think to control the path each single individual will follow during evacuation). These considerations are controlled by further constraints in the second model. The third and final model has a different goal: it can be used in a planning phase, in order to choose between a given set E of shelters a subset e that will save more people in a given amount of time. These three LP problems are called respectively B (like 'Basic'), S (like 'Straightforward evacuation') and F (like 'Filtering').

2.3.1 Equipment of problem

Here there will be presented the tools of the problem:

- A graph *G(V,A)* is required to represent the characteristics of the building at hand.
  - V corresponds to set of vertexes in which the building has been embedded. Every node has a maximum load capacity and can be crossed in any direction in a single time slot. V_s is a subset of V which contains vertexes labeled as "safe place".
  - Set of arcs A corresponds to the connections between adjacent nodes not interrupted by walls. Capacity of links is determined by type thereof.
- Decision variables are:
  - x_i^t = number of persons in vertex i at time t. Initial occupation is supposed to be known.
  - y_{ij}^t = number of persons leaving cell i towards cell j from time t to time t+1.
  - z_{ij} = binary variable. Its value is 1 if the flow from i to j is allowed, 0 otherwise.

- Parameters of the graph model are:
  - N = set where each element n_i corresponds to the capacity of cell i.
  - Q_{ij} = max capacity of link from i to j, therefore neglecting congestion.
  - Time is described through the set $T = \{1, 2, \ldots, \tau\}$.
  - δ = percentage that rules congestion. It allows to entry in a unit cell only to a fraction of the effective available seats.
  - Γ_i^{-1} = set of predecessor cell to cell i.
  - Γ_i = set of successor cell to cell i.
  - d_i = initial occupation of cell i.
  - e = dimension of subset of exits.
  - B = budget to comply with.
  - c_i = cost of the exploitation of the i-th exit.
  - V_f = set of vertexes that can actually fork.

  All these parameters are contained in the input data file or passed directly through the solver scene.

2.3.2 Model B

$$O.F. = min \sum_{i \notin N_s} x_i^{\tau}$$

Subject to:

$$x_i^{t=2} = x_i^{t=1} + \sum_{j \in \Gamma^{-1}(i)} y_{ji}^{t=1} - \sum_{j \in \Gamma(i)} y_{ij}^{t=1} + d(i) \qquad \forall i, j \in V \qquad (2.1)$$

$$x_i^t = x_i^{t-1} + \sum_{j \in \Gamma^{-1}(i)} y_{ji}^{t-1} - \sum_{j \in \Gamma(i)} y_{ij}^{t-1} \qquad \forall i, j \in V;\ t \in T \mid t > 1 \qquad (2.2)$$

$$y_{ij}^t + y_{ji}^t \leq Q_{ij} \qquad \forall i, j \in V;\ t \in T \qquad (2.3)$$

$$\sum_{j \in \Gamma(i)} y_{ji}^t \leq x_i^t \qquad \forall i, j \in V;\ t \in T \qquad (2.4)$$

$$\sum_{j \in \Gamma^{-1}(i)} y_{ji}^t \leq (N_i - x_i^t)\delta \qquad \forall i, j \in V;\ t \in T \qquad (2.5)$$

$$x_i^t \geq 0 \qquad \forall i \in V;\ t \in T \qquad (2.6)$$

$$y_{ij}^t \geq 0 \qquad \forall i, j \in V;\ t \in T \qquad (2.7)$$

2.3.3 Model S

In order to avoid forks out a unit cell and bi-directional flows across a link, comes into play the binary variable z_{ij} and few new constraints are added:

$$\sum_j z_{ij} = 0 \qquad \forall i \in V_s;\ j \in \Gamma(i) \qquad (2.8)$$

$$\sum_j z_{ij} = 1 \qquad \forall i \in ((V - V_s)\&(V - V_f));\ j \in \Gamma(i) \qquad (2.9)$$

$$y_{ij}^t \leq Q_{ij}(z_{ij}) \qquad \forall i, j \in V;\ t \in T \qquad (2.10)$$

$$y_{ji}^t \leq Q_{ij}(1 - z_{ij}) \qquad \forall i, j \in V;\ t \in T \qquad (2.11)$$

$$z_{ij} \in \{0, 1\} \qquad \forall i, j \in V \qquad (2.12)$$

2.3.4 Model F

Following constraints are needed for selecting among 'E' exits only a portion 'e' of them.

$$\sum_{j \in \Gamma^{-1}(i)} z_{ji} \leq e \qquad \forall i \in V_s;\ j \in \Gamma^{-1}(i) \qquad (2.13)$$

... and, if using an exit is a matter of money according to the necessity of meeting the budget 'B', also the following:

$$\sum_{j \in \Gamma^{-1}(i)} c_j z_{ji} \leq B \qquad \forall i \in V_s;\ j \in \Gamma^{-1}(i) \qquad (2.14)$$

2.3.5 Interpretation of constraints

Constraint reference	Interpretation
2.1	At time 2 the number of people found in a vertex is given by the previous occupation of the same vertex plus the initial occupation plus the number of persons who arrived minus number of people who left.
2.2	For each time greater than 2, the number of people found in a vertex is given by the previous occupation of the same vertex plus the number of persons who arrived minus the number of people who left.
2.3	Flow between two nodes can never exceed the capacity of the link.
2.4	Temporary occupation on a node must be less than the capacity of the cell.
2.5	The real number of persons who can enter a node is a fraction of the actual available residual capacity. This for taking into account factors like congestion e panic.
2.6	Obviously node occupation can't be negative.
2.7	. . . as well as the flow between two connecting units.
2.8	Persons who have reached the safe place must stay there.
2.9	Out of a cell people must go always in the same direction, forks are not allowed. . . unless the i-th vertex belongs to the set V_f. Actually exists a portion of nodes out of which it is possible to fork, in particular it is possible for cells inside a room.
2.10	The capacity of a link is associated to a binary variable in order to allow the flow one way only.
2.11	The capacity of a link is associated to a binary variable in order to allow the flow one way only.
2.12	The variable that rules flows in one direction is binary, precisely.
2.13	The number of exits could be imposed smaller than the actual maximum number.
2.14	In addition the number of exits have to meet a budget as a matter of cost for installing or keeping open or keep in service the way out.

Table 2.1: Interpretation of constraints

2.3.6 Setting of parameters

In order to reflect reality during the expression of a quake, the model must be parameterized by taking into account walking velocity, door width, human behavior, links and nodes capacity.

As already mentioned in 2.1 the building examined in this study is Palazzo Camponeschi. It all starts with the blueprints. This has been split into 108 unit cells as you can see in figure 2.1 and 2.2 and this grid has been therefore converted in the graph in Figure 2.3.6 [6]. Node zero represents the outside of the building, nothing more than a safe place. Once reached the latter, people are out of the woods and can't leave it by re-entering in the building. For this reason, node 0 has infinite capacity while all the others are able to contain 1.25 persons per square meter according to UK fire safety regulations. As regards link capacity it is determined by type thereof. In particular is adopted a free flow walking speed of 1.20 m/s [4] multiplied by the width of the bond shared: it could be a door or the full side of the cell. The pedestrian free flow velocity is different when people travel over the stairs, in this situation the speed is 0.76 m/s [3].

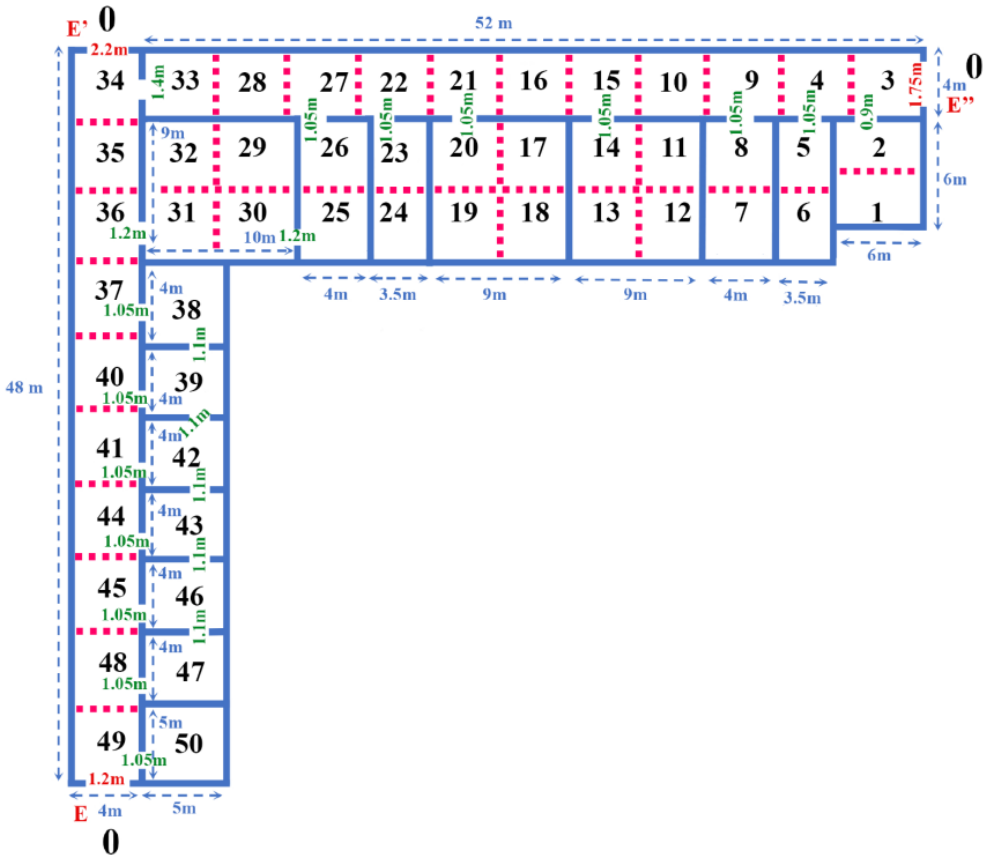

Figure 2.1: Ground floor of the Palazzo

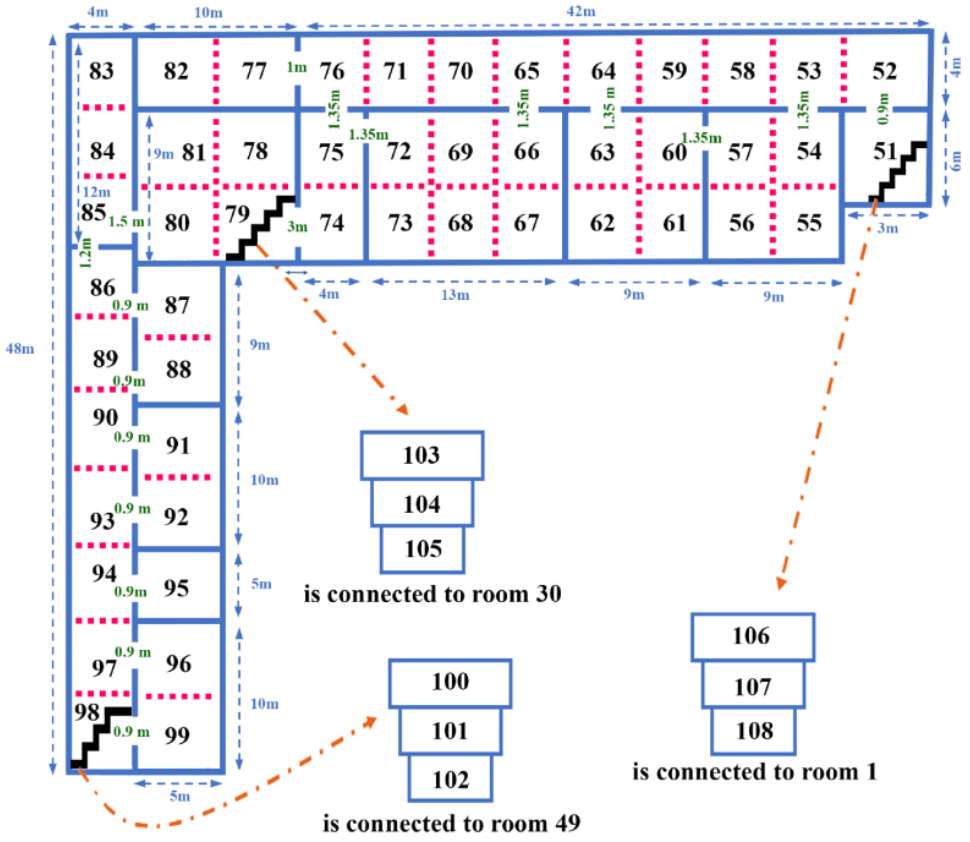

Figure 2.2: Top Floor of the Palazzo

In addition to these experimental data, takes over δ in such a way as taking into account congestion and human behavior like panic. Its value is set to 0.75, this mean that only the 75% of the actual flow is able to move. As regards the initial occupation of the vertexes have been addressed two scenarios. In the first one people are uniformly distributed, in particular there are 5 persons in any node (excluding the node 0 obviously) whereas in the second scenario people are distributed in a more concentrate manner and in less nodes, in particular in nodes {54,56,57,60,61,62,63,66,67,68,69,72,73,77,78,79,80,81, 82,83,84,85} there are {25,24,25,25,25,25,25,24,24,24,24,24,24,25,28,28,28,28, 25,20,20,20} respectively. Let's call these scenarios respectively "Uniform" and "Concentrations". In both cases the total amount of people is equal to 540, a bit more of the peak registered during an event on 29 September 2017, when the simultaneous presence of 528 persons was recorded.

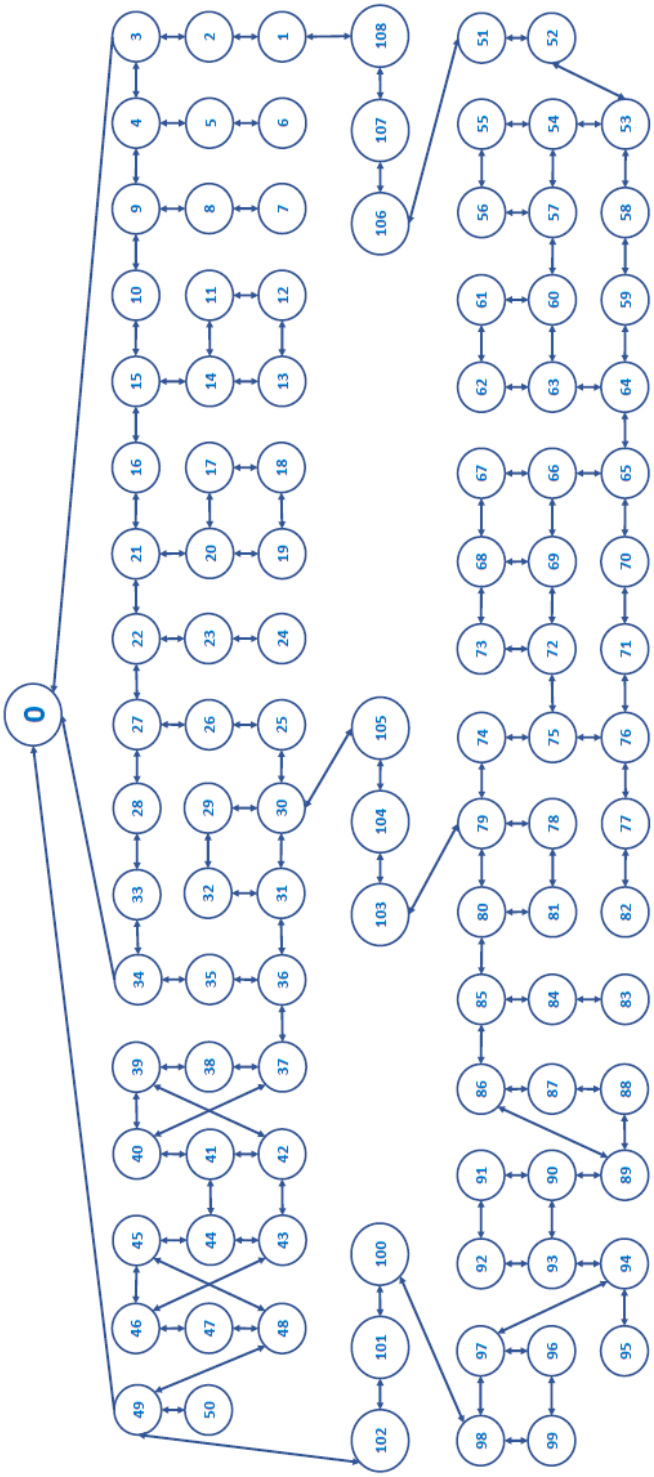

Figure 2.3: Palazzo Camponeschi in the form of graph

Chapter 3

Algorithms

Once defined the "rules" (2.3.6), the first goal has been finding the minimum total evacuation time through a logarithmic search. In other words trying different values of τ until the first instant that had empty the Palazzo, hence everybody was safe, would be found. Let's call it τ_{safe}. Since τ_{safe} could be quite large, the solver could have trouble solving a problem, in particular due to run-time. For this reason had to take measures from an algorithmic standpoint. All the codes can be found in Appendix A.

3.1 Constraint generation in practice

The first approach has been the Constraint Generation in support of Straightforward model which is too large. It is said in advance that the practical example can be found at the end of this section. The point is to let Xpress IVE solve the Basic problem 2.3.2 which takes only a few seconds to find the optimal solution, then it was checked whether there were manifestation of behaviors not tolerated (like forks out of a node or bi-directional flows in a connection) and if so, the constraints of 2.3.3 were added to those targeted vertexes/arcs, the ones that do not act properly... and so on. This is repeated until there were no longer forks or bi-directional flows. In the end, constraints (2.9), (2.10), (2.11), (2.12), are applied only to a portion of nodes, not to all. The key point is to solve one huge problem or several smaller problems. Is it faster? In the next chapter, results will show that, in this specific case, the constraint generation approach is always worse in terms of run-time. For this part of the work, algorithms designed are two: "C" and "G". "C" loops on the dimension T (time) hence it is looked up the first moment in which forks (or bi-directions) occur and then the constraints, thus also the problem, are refreshed by binding the nodes that for first are affected by the undesired behavior... and the next check will start from time

1 because with the new updated model, the undesired behavior may happen before the time found in the previous loop.

Example:

1. Inner loop = $1...\tau$
2. Fork spotted out of node 7 at time 32.
3. Refresh model by adding proper constraints to node 7
4. Inner loop = $1...\tau$
5. Fork spotted out of node 19 at time 4.
6. Refresh model by adding proper constraints to node 19
7. Inner loop = $1...\tau$

$\vdots$

"G" loops on the dimension V (number of vertexes) therefore are looked up all the forking cells, constraints are added, problem is refreshed and a new set of forking nodes is searched.

Example:

1. Inner loop = $1...V_{max}$ (node 0 obviously excluded)
2. Fork spotted out of nodes 7 and 26 respectively at time 32 and 70
3. Refresh model by adding proper constraints to nodes 7 and 26
4. Inner loop = $1...V_{max}$
5. Fork spotted out of node 19 and 67 respectively at time 4 and 13. Observation: if in the first iteration it had been added the constraint in node 7 only, maybe once refreshed the problem, there would be no other forks. The algorithm would be over... but it's not.
6. Refresh model by adding proper constraints to node 19 and 67
7. Inner loop = $1...V_{max}$

$\vdots$

The C algorithm is more 'correct', has the pro of adding less constraints but the con of taking more run-time. Although the G algorithm has the con

of adding more constraints, these generate a smaller domain of solutions for the benefit of run-time.

In both cases the check of forks and bi-directions at every iteration is performed in sequence: first are searched forks only. When there are no more forks, check for bi-directions starts. From here on out the check is executed in parallel, both forks and bi-directions. From various tests, this execution emerged as faster than checking for anomalies in parallel from the beginning.

3.2 Matheuristics in practice

This approach intervenes in case the previous algorithm is not fast enough. It is said in advance that the practical example follows in this section. The aim is to solve the problem several times with an ad-hoc algorithm as mentioned in the previous section (3.1), by accepting on each iteration the current solutions just found and by adding them in the form of new constraints to the next iteration that, being stricter, favors run-time saving. Always? The next chapter will show that Matheuristics is competitive since in some instances run-time is lower (then better) than with an exact algorithm.

Example:

1. First iteration, first resolution.

2. Reading of results: since node 85 forked, constraints (2.9) were added and the corresponding solutions are:

$$z(85, 80) = 1$$

and

$$z(85, 86) = 0$$

3. New constraints are added in order to keep these solutions:

$$z(85, 86) <= 0$$

and

$$z(85, 80) >= 1$$

4. Second iteration, second resolution.

$$\vdots$$

This process is always (in this study, not universally) faster than a Constraint Generation algorithm but has a drawback: the solutions at each iteration are temporary, they are only valid for the optimum of the temporary sub-problem, not for the complete problem that would have been solved with the pure constraint generation algorithm. By forcing those values in fact, the optimum solution of the problem may worsen.

Chapter 4

Results

In this section will be exhibited and compared results related to different launches of the solver in order to draw conclusions on the behavior of the evacuation of Palazzo Camponeschi. The dimensions in question will be 'number of persons', 'run-time', 'algorithms' and 'scenarios'. Some graphs will follow in order to better view the results. The first section compares the straightforward model (B+S) in both scenarios in terms of time of evacuation. After this, scenarios will be taken separately (Uniform first in section 4.2 and Concentrations then in section 4.3) and analyzed by showing for each one the relevant comparisons. All measure of time are expressed is seconds. For any detail of data used, check in appendix Figures B.1, B.2.

4.1 Scenarios comparison in B+S model

Due to much accentuated congestion issues in the Concentrations scenario, the full emptying of the building is delayed compared to the one of the Uniform scenario. See Table 4.1 and Figure 4.1.

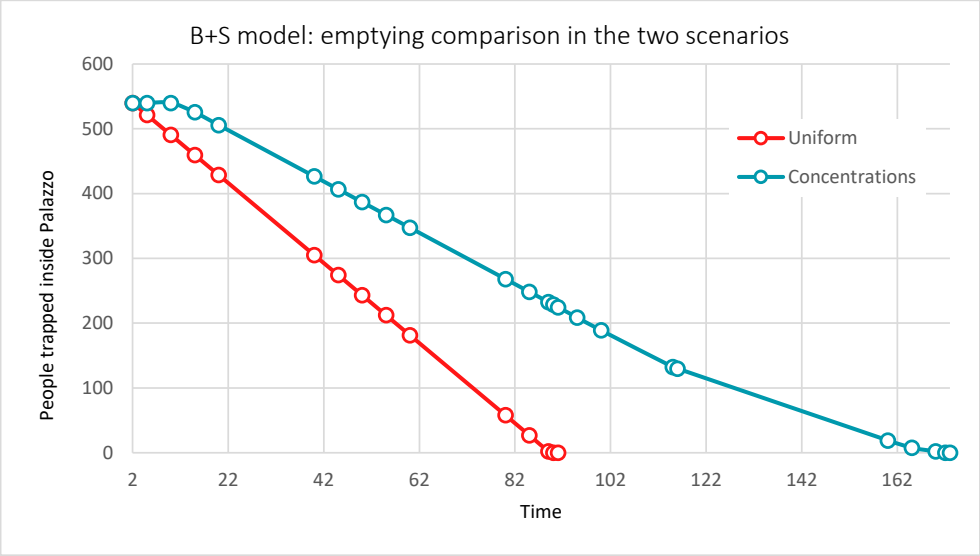

Figure 4.1: Emptying in both scenarios

τ_{safe} $[s]$	Scenario
91	Uniform
173	Concentrations

Table 4.1: B+S model: comparison of τ_{safe} in both scenarios

4.2 Uniform scenario

In this section will be shown some differences under Uniform scenario in terms of emptying time and in terms of run-time. The first one is between B and B+S model, in the second subsection will be compared the algorithms used and in the third subsection model B+S vs B+S+F.

4.2.1 B vs B+S

Under Uniform scenario, in this subsection, B and B+S models are compared.

The τ_{safe} in the model B+S is delayed by 1 second, see Table 4.2 and Figure 4.2. Note that model B is a lower bound of model B+S and therefore τ_{safe} of the former will always be less than or equal to τ_{safe} of the latter.

If this comparison is played on the run-time dimension, emerges the graph in Figure 4.3. Whereas run-time of model B increases very slowly, the other one undergoes significant surges. In this graph as in all the following graphs of run-times comparison, the exponential trend proves the NP-hardness of the problem in question and oscillations can be mitigated, in order to obtain a smoother function, by an approximation algorithm as described in section 1.1. The instruction to achieve that is:

setparam("XPRS_MAXTIME", timelimit)

where "timelimit" must be set to the desired value.

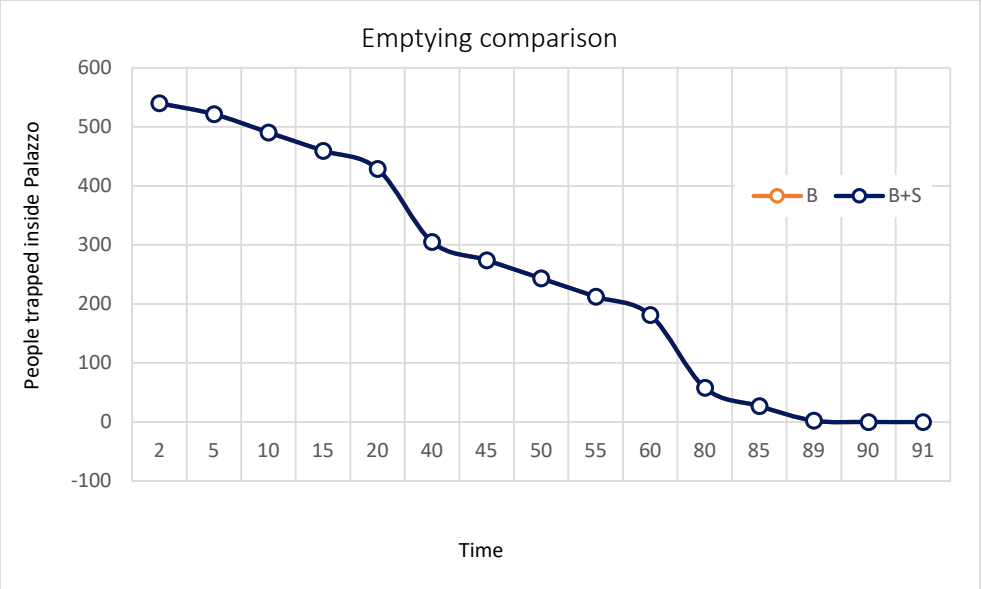

Figure 4.2: Uniform scenario, emptying: Basic model vs Straightforward evacuation model

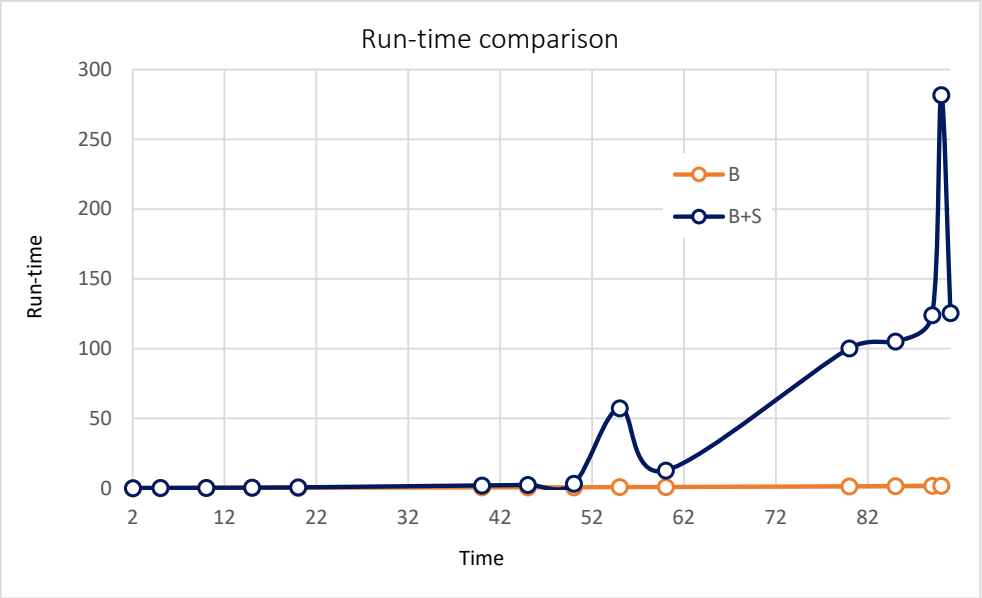

Figure 4.3: Uniform scenario: run-time: Basic model vs Straightforward evacuation model

τ_{safe} $[s]$	Model
90	B
91	B+S

Table 4.2: Uniform scenario: comparison of τ_{safe} in the two primary models

4.2.2 Comparison of algorithms

B+S, B+S Constraint Generation and B+S Matheuristics are compared in this subsection under Uniform scenario.

The functions of emptying in the three cases coincide (Figure 4.4). It is no surprise that function B+S coincides with B+S Constraint Generation because both are exact methods and therefore they lead to the same results, but so is the fact that B+S Matheuristics function almost coincides (except for $tmax = 90$, see Figure B in appendix) because it cannot be taken for granted. Although τ_{safe} (Table 4.3) are the same, run-times are quite different, above all for B+S Constraint Generation which is one that takes longer to solve the problem (Figure 4.4). The other two act differently: while B+S is faster than B+S Matheuristics (in terms of run-time) in the first half of the graph, things are reversed in the second half.

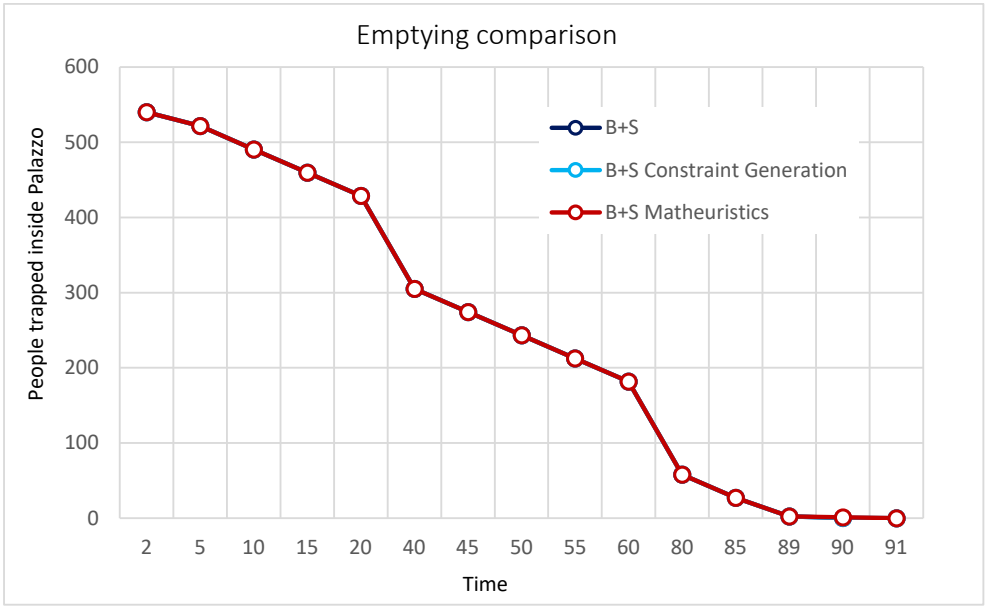

Figure 4.4: Uniform scenario, emptying: algorithms in comparison

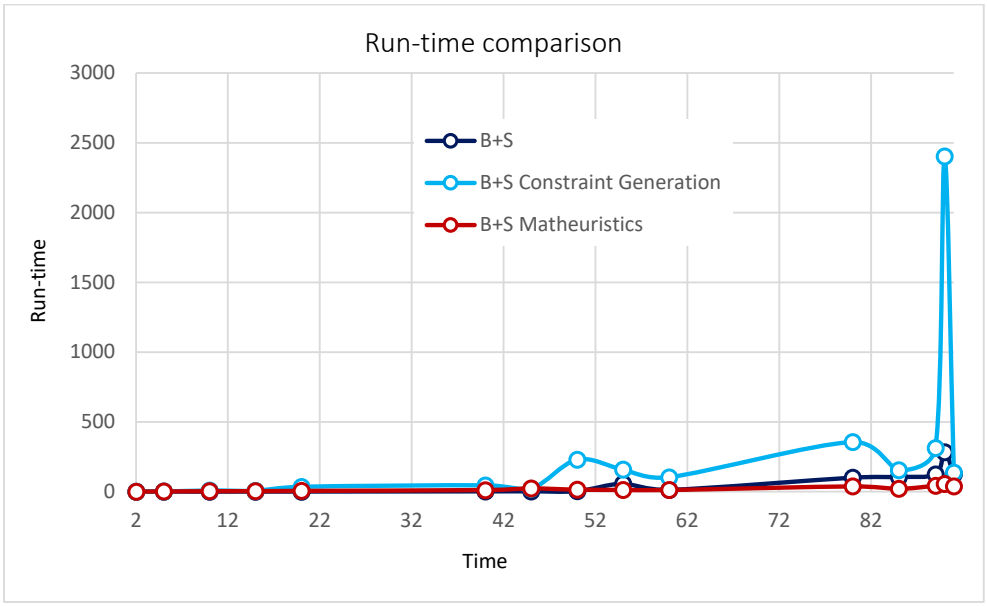

Figure 4.5: Uniform scenario: run-time: algorithms in comparison

τ_{safe} $[s]$	Algorithm
91	B+S
91	B+S Constraint Generation
91	B+S Matheuristics

Table 4.3: Uniform scenario: comparison of τ_{safe} in the three algorithms

4.2.3 B+S vs B+S+F in Matheuristics environment

B+S is compared to B+S+F model, both performed with matheuristics under Uniform scenario.

As can be seen in Figure 4.6 and as well as can be expected, exploiting two exits instead of three, therefore setting the "e" parameter of constraint 2.13 equal to 2, delays the τ_{safe} (see Table 4.4). With reference to Figure 2.3.6, the exits chosen for the way out are the 34-th and the 3-rd, thus the 49-th was discarded. With regard to run-time, Figure 4.7 shows that the curves take turns a bunch of times. The latest data are collected obviously for B+S+F model only because Palazzo in B+S model was already empty in those times. Noteworthy the last reading in which the run-time rises abruptly.

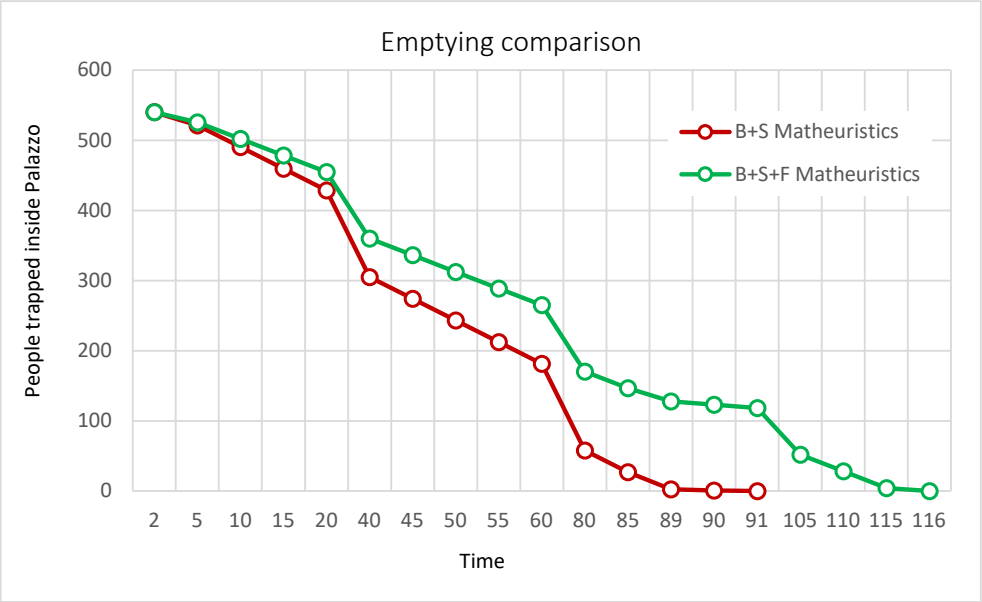

Figure 4.6: Uniform scenario: emptying with Matheuristics: B+S vs B+S+F

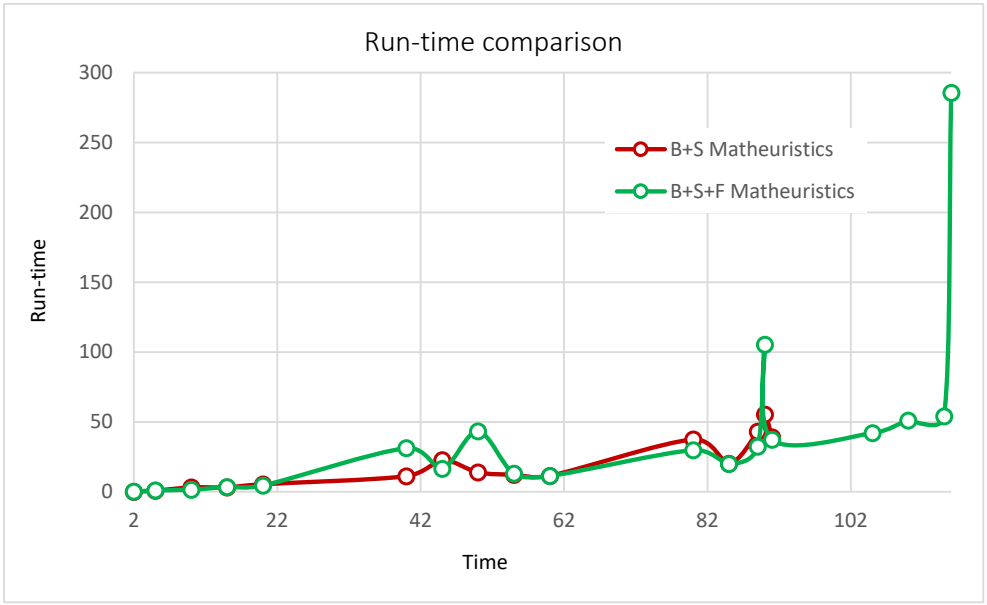

Figure 4.7: Uniform scenario: run-time: Matheuristics for B+S and B+S+F in comparison

τ_{safe} $[s]$	Model
91	B+S Matheuristics
116	B+S+F Matheuristics

Table 4.4: Uniform scenario: comparison of τ_{safe} obtained with Matheuristics in B+S and in B+S+F

4.3 Concentrations scenario

In this section will be shown some differences under Concentrations scenario in terms of emptying time and in terms of run-time.

The first one is between B and B+S model, in the second subsection will be compared the algorithms used and in the third subsection model B+S vs B+S+F.

4.3.1 B vs B+S

Under Concentrations scenario, in this subsection, B and B+S models are compared.

By just looking at the graph in Figure 4.8 it is possible to better notice the differences between the two functions compared to the similar case of the uniform scenario (Figure 4.2). In the present case in fact the τ_{safe} (Table 4.5) of the straightforward model is a lot farther.

Run-time of the Basic model slowly increases whereas in the B+S model it grows with an exponential trend (Figure 4.9).

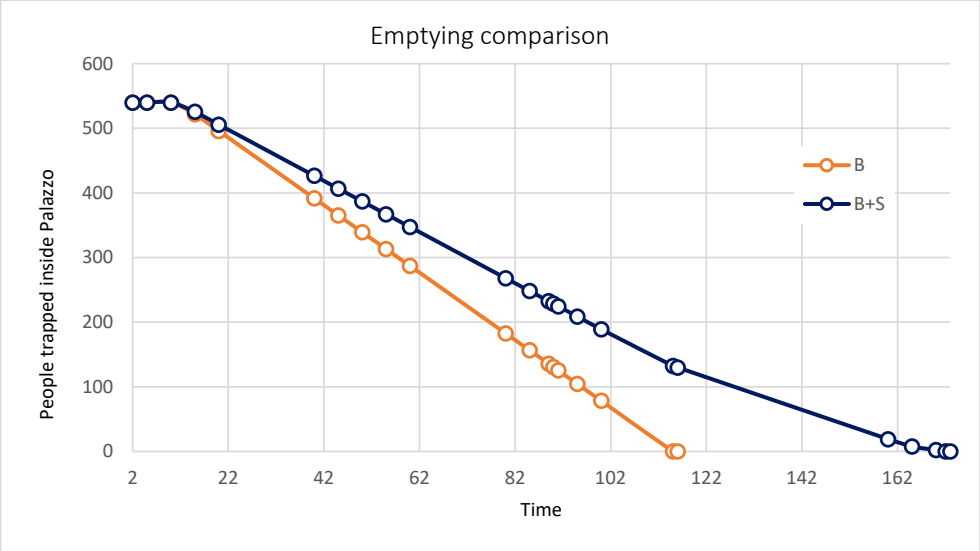

Figure 4.8: Concentrations scenario, emptying: Basic model vs Straightforward evacuation model

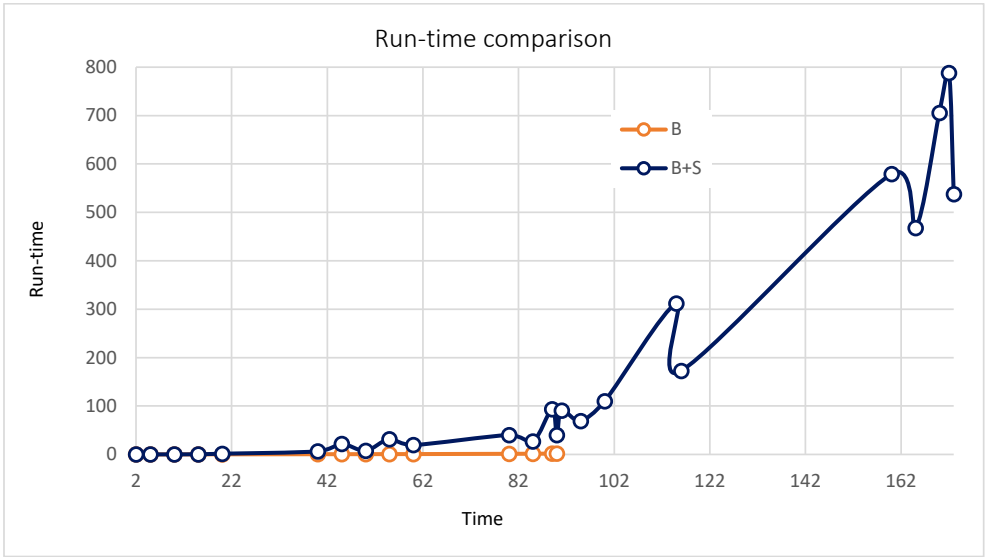

Figure 4.9: Concentrations scenario: run-time: Basic model vs Straightforward evacuation model

τ_{safe} $[s]$	Model
116	B
173	B+S

Table 4.5: Concentrations scenario: comparison of τ_{safe} in the two primary models

4.3.2 Comparison of algorithms

B+S, B+S Constraint Generation and B+S Matheuristics are compared in this subsection under Concentrations scenario.

The functions of emptying are the same (Figure 4.10) as also the τ_{safe} (Table 4.6) even if should be stressed that there are less measurements due to run-time issues (Figure 4.10) for the Constraint Generation algorithm.

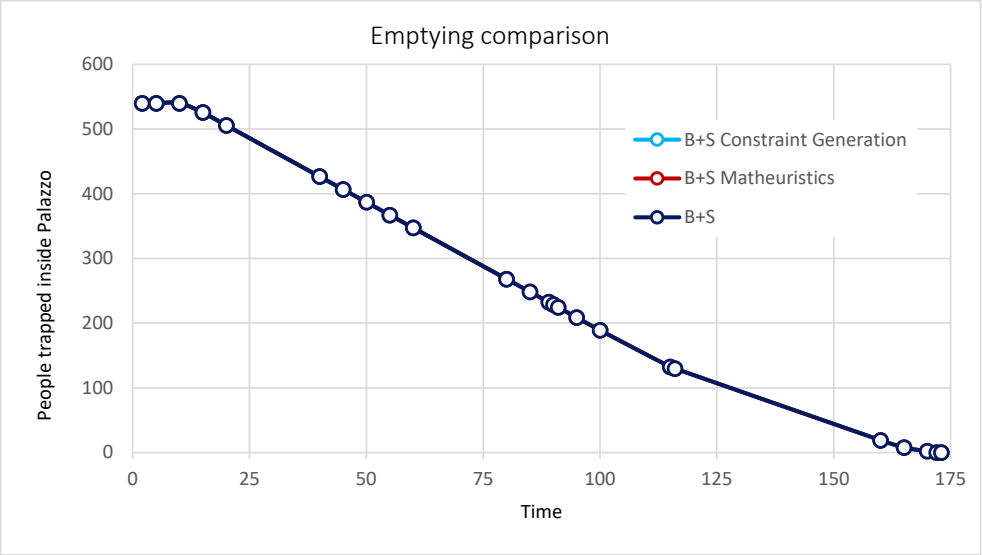

Figure 4.10: Concentrations scenario, emptying: algorithms in comparison

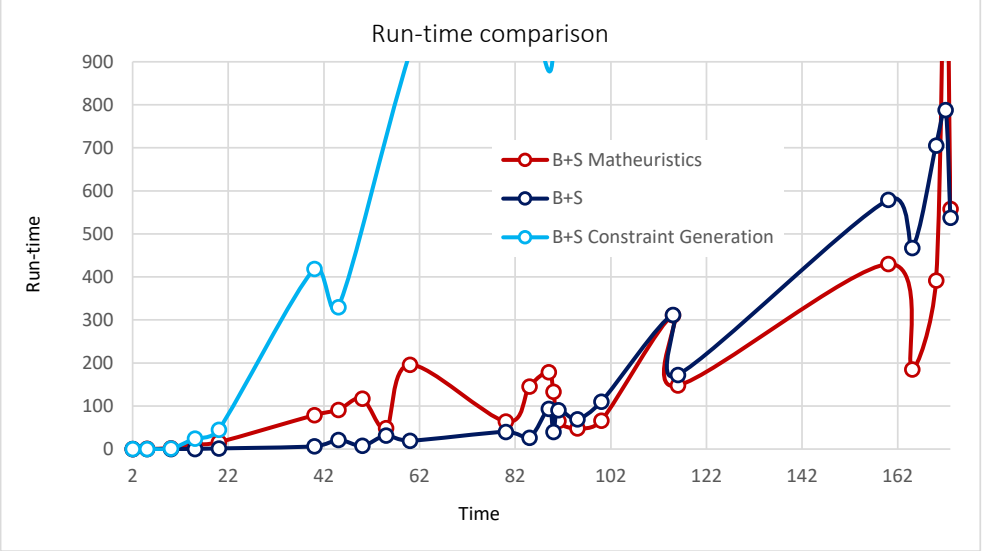

Figure 4.11: Concentrations scenario: run-time: algorithms in comparison

τ_{safe} $[s]$	Algorithm
173	B+S
173	B+S Constraint Generation
173	B+S Matheuristics

Table 4.6: Concentrations scenario: comparison of τ_{safe} in the three algorithms

4.3.3 B+S vs B+S+F in Matheuristics environment

B+S is compared to B+S+F model, both performed with matheuristics under Concentrations scenario.

Although B+S+F Matheuristics empties a little slower (see Figure 4.12), the final result τ_{safe} is the same (Table 4.7), contrary to Figure 4.6; this also means that in this specific case there is no difference between using two doors (the 3-rd and the 34-th way out) and three doors, the outcome is the same. As regard run-time, as before in Figure 4.7, the curves take turns a bunch of times (see Figure 4.13).

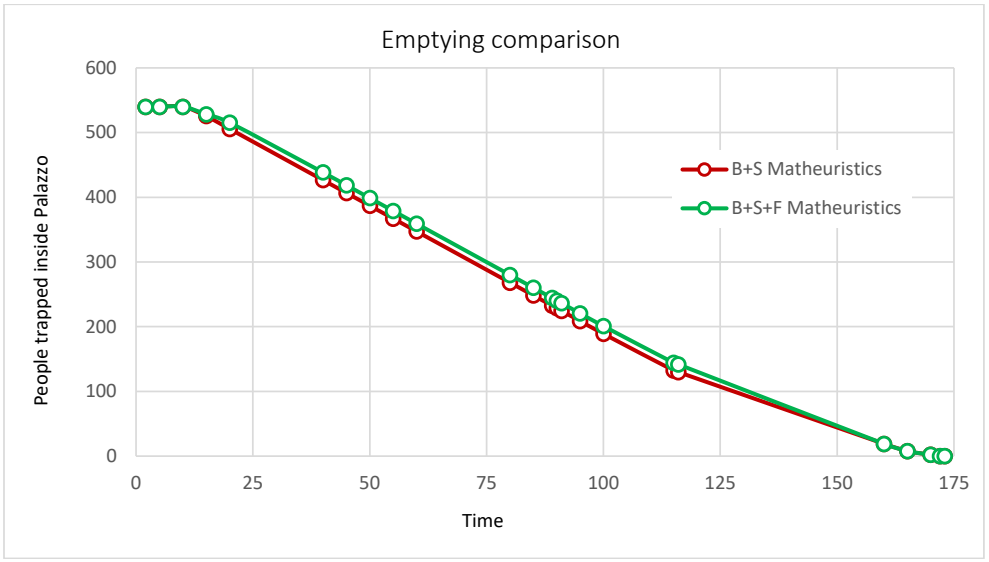

Figure 4.12: Concentrations scenario: emptying with Matheuristics: B+S vs B+S+F

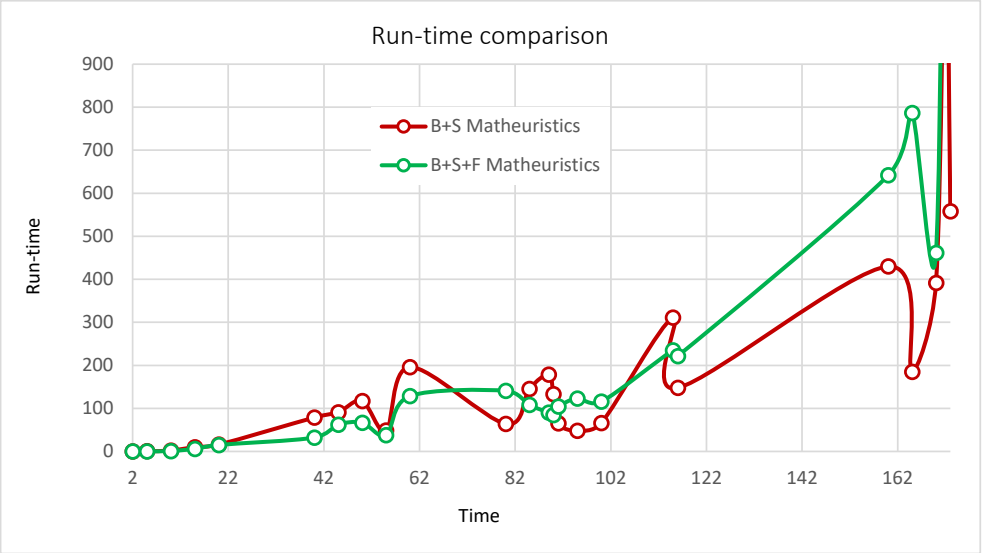

Figure 4.13: Concentrations scenario: run-time: Matheuristics for B+S and B+S+F in comparison

τ_{safe} $[s]$	Model
173	B+S Matheuristics
173	B+S+F Matheuristics

Table 4.7: Concentrations scenario: comparison of τ_{safe} obtained with Matheuristics in B+S and B+S+F

Chapter 5

Conclusion

This work was born with the aim of crafting and delivering a framework that consists in few algorithmic options for any problem in section 2.2, particularly for Humanitarian emergency problems. Computational tests have proven that there is not an obvious best algorithm choice, since the outcomes depend by the intrinsic characteristics of the problem in question e this leads to a competitiveness of the tools adopted. Although the framework lacks of a predominant procedure, there are two discriminating factors. The first one is the phase in which the analysis shall be applied for a matter of available time: in a planning phase, in general this time is large and hence it is possible and favorable to exploit an exact algorithm; in case of short of time, it is more likely to consider using a heuristic approach. The second factor is the real case itself: in case of a large enough available time, it is undoubtedly preferable an exact algorithm since in this case, with a heuristic algorithm, people could get hurt because of potential sub-optimal results.

Appendix A

Xpress IVE codes

All the codes of all the models follow in this section in sequence: Model B then Model B+S and finally Model B+S ConGen. The latter can optionally be used to perform Model B+S+F with Matheuristics.

```
model Caristo_Model_B
uses "mmxprs"
uses "mmsystem"

!setparam("XPRS_verbose", true)
setparam("REALFMT", "%1.15f")

declarations
    Nmax = 108
    Tmax = 90  change this
    delta= 0.75
    TIMESLOTS= 1..Tmax
    NODES= 0..Nmax

    SourceNode: array(NODES) of integer     ! =1 The node is a source node
    SinkNode: array(NODES) of integer       ! =1 The node is a safe place
    Graph: array(NODES, NODES) of integer   ! Graph matrix
    Q: array(NODES, NODES) of real ! Flow that can exit/enter max
    N: array(NODES) of integer ! Node Capacity
    d: array(NODES) of integer ! Initial occupation
    x: array(NODES, TIMESLOTS) of mpvar                    !occupation
    y: array(NODES, NODES, TIMESLOTS) of mpvar             !flow
    z: array(NODES, NODES) of mpvar                ! =1 --> fork allowed
end-declarations

initializations from "InputDatasConcentrationsScenario.txt" !Switch between:
!InputDatasConcentrationsScenario.txt and InputDatasUniformScenario.txt
    Graph
    SourceNode
    SinkNode
    d
    N
    Q
end-initializations

! (1)
forall(i in NODES) do
    x(i,2) = x(i,1) + sum(j in NODES | Graph(j,i) = 1) y(j,i,1) - sum(j in NODES
| Graph(i,j) = 1) y(i,j,1) + d(i)
end-do

! (2)
forall(i in NODES, t in TIMESLOTS | t>2) do
    x(i,t) = x(i,t-1) + sum(j in NODES | Graph(j,i) = 1) y(j,i,t-1) - sum(j in
NODES | Graph(i,j) = 1) y(i,j,t-1)
end-do

! (4)
forall(i in NODES, t in TIMESLOTS | t>0) do
    sum(j in NODES | Graph(i,j) = 1) y(i,j,t) <= x(i,t)
end-do

! (3)
forall(i, j in NODES | Graph(j,i) = 1, t in TIMESLOTS | t>0) do
    y(j,i,t)+y(i,j,t) <= Q(i,j)
end-do

! (5)
forall(i in NODES, t in TIMESLOTS | t>1) do
    sum(j in NODES | Graph(j,i) = 1) y(j,i,t-1) <= (N(i)-x(i,t-1))*delta
end-do

ObjSum := sum(i in NODES | SinkNode(i) = 0) x(i,Tmax)
```

```
start_time := time(SYS_NOW)
minimize(ObjSum)
end_time := time(SYS_NOW)
writeln("OF = ", getobjval, " - time = ", end_time-start_time, " ms.")
end-model
```

```
model Caristo_Model_B_S
uses "mmxprs"
uses "mmsystem"

!setparam("XPRS_verbose", true)
setparam("REALFMT", "%1.15f")

declarations
    Nmax = 108
    Tmax = 90  change this
    delta= 0.75
    TIMESLOTS= 1..Tmax
    NODES= 0..Nmax

    SourceNode: array(NODES) of integer     ! =1 The node is a source node
    SinkNode: array(NODES) of integer       ! =1 The node is a safe place
    Graph: array(NODES, NODES) of integer   ! Graph matrix
    Q: array(NODES, NODES) of real ! Flow that can exit/enter max
    N: array(NODES) of integer ! Node Capacity
    d: array(NODES) of integer ! Initial occupation
    x: array(NODES, TIMESLOTS) of mpvar                    !occupation
    y: array(NODES, NODES, TIMESLOTS) of mpvar             !flow
    z: array(NODES, NODES) of mpvar                ! =1 --> fork allowed
end-declarations

initializations from "InputDatasConcentrationsScenario.txt" !Switch between:
!InputDatasConcentrationsScenario.txt and InputDatasUniformScenario.txt
    Graph
    SourceNode
    SinkNode
    d
    N
    Q
end-initializations

! (1)
forall(i in NODES) do
    x(i,2) = x(i,1) + sum(j in NODES | Graph(j,i) = 1) y(j,i,1) - sum(j in NODES
| Graph(i,j) = 1) y(i,j,1) + d(i)
end-do

! (2)
forall(i in NODES, t in TIMESLOTS | t>2) do
    x(i,t) = x(i,t-1) + sum(j in NODES | Graph(j,i) = 1) y(j,i,t-1) - sum(j in
NODES | Graph(i,j) = 1) y(i,j,t-1)
end-do

! (4)
forall(i in NODES, t in TIMESLOTS | t>0) do
    sum(j in NODES | Graph(i,j) = 1) y(i,j,t) <= x(i,t)
end-do

! (3)
forall(i, j in NODES | Graph(j,i) = 1, t in TIMESLOTS | t>0) do
    y(j,i,t)+y(i,j,t) <= Q(i,j)
end-do

! (5)
forall(i in NODES, t in TIMESLOTS | t>1) do
    sum(j in NODES | Graph(j,i) = 1) y(j,i,t-1) <= (N(i)-x(i,t-1))*delta
end-do

! (8)
```

```
forall(i in NODES | SinkNode(i)=1) do
    sum(j in NODES | Graph(i,j) = 1) z(i,j) = 0
end-do

forall(i in NODES) do
    forall(j in NODES | (Graph(i,j)=1)) do
        forall(t in TIMESLOTS | t > 0) do
            ! (10)
            y(i,j,t) <= Q(i,j)*z(i,j)
            !(11)
            y(j,i,t) <= Q(i,j)*(1-z(i,j))
        end-do
        !(12)
        z(i,j) is_binary
    end-do
end-do

forall(i in NODES | SinkNode(i)=0 and ForkFree(i)=0) do
    !(9)
    sum(k in NODES | Graph(i,k) = 1) z(i,k) = 1
end-do

ObjSum := sum(i in NODES | SinkNode(i) = 0) x(i,Tmax)

start_time := time(SYS_NOW)
minimize(ObjSum)
end_time := time(SYS_NOW)
writeln("OF = ", getobjval, " - time = ", end_time-start_time, " ms.")
end-model
```

```
model Caristo_B_S_ConGen_and_Matheur_and_B_S_F
uses "mmxprs"
uses "mmsystem"

!setparam("XPRS_verbose", true)
setparam("REALFMT", "%1.15f")

declarations
    Nmax = 108
    Tmax = 90  change this
    delta= 0.75
    TIMESLOTS= 1..Tmax
    NODES= 0..Nmax

    SourceNode: array(NODES) of integer     ! =1 The node is a source node
    SinkNode: array(NODES) of integer       ! =1 The node is a safe place
    Graph: array(NODES, NODES) of integer   ! Graph matrix
    Q: array(NODES, NODES) of real ! Flow that can exit/enter max
    N: array(NODES) of integer ! Node Capacity
    d: array(NODES) of integer ! Initial occupation
    x: array(NODES, TIMESLOTS) of mpvar                    !occupation
    y: array(NODES, NODES, TIMESLOTS) of mpvar             !flow
    z: array(NODES, NODES) of mpvar                ! =1 --> fork allowed
end-declarations

initializations from "InputDatasConcentrationsScenario.txt" !Switch between:
!InputDatasConcentrationsScenario.txt and InputDatasUniformScenario.txt
    Graph
    SourceNode
    SinkNode
    d
    N
    Q
end-initializations

! (1)
forall(i in NODES) do
    x(i,2) = x(i,1) + sum(j in NODES | Graph(j,i) = 1) y(j,i,1) - sum(j in NODES
| Graph(i,j) = 1) y(i,j,1) + d(i)
end-do

! (2)
forall(i in NODES, t in TIMESLOTS | t>2) do
    x(i,t) = x(i,t-1) + sum(j in NODES | Graph(j,i) = 1) y(j,i,t-1) - sum(j in
NODES | Graph(i,j) = 1) y(i,j,t-1)
end-do

! (4)
forall(i in NODES, t in TIMESLOTS | t>0) do
    sum(j in NODES | Graph(i,j) = 1) y(i,j,t) <= x(i,t)
end-do

! (3), !(10)
forall(i, j in NODES | Graph(j,i) = 1, t in TIMESLOTS | t>0) do
    y(j,i,t)+y(i,j,t) <= Q(i,j)
    y(i,j,t) <= Q(i,j)*z(i,j)
end-do

! (8)
forall(i in NODES | SinkNode(i)=1) do
    sum(j in NODES | Graph(i,j) = 1) z(i,j) = 0
end-do
```

```
! (12)
forall(i, j in NODES | Graph(j,i) = 1) do
    z(i,j) is_binary
end-do

! (5)
forall(i in NODES, t in TIMESLOTS | t>1) do
    sum(j in NODES | Graph(j,i) = 1) y(j,i,t-1) <= (N(i)-x(i,t-1))*delta
end-do

(!      Uncomment the following lines to run model B+S+F with Matheuristics

! (2.12)
forall(i in NODES | SinkNode(i)=1) do
    sum(j in NODES | Graph(j,i) = 1) z(j,i) <= 2
end-do
!)

ObjSum := sum(i in NODES | SinkNode(i) = 0) x(i,Tmax)

start_time := time(SYS_NOW)
flag:=1
while(flag=1) do
    flag:=0
    minimize(ObjSum)

    (!  Uncomment the following lines to run with Matheuristics

    forall(i in NODES | (i in ITERATION)) do
        forall(j in NODES | (Graph(i,j)=1)) do
            if(getsol(z(i,j)) = 1) then
                z(i,j)>=1
                flag:=1
            end-if
            if(getsol(z(i,j)) = 0) then
                z(i,j)<=0
                flag:=1
            end-if
            ITERATION-={i}
        end-do
    end-do
    !)

    forall(i in NODES | SinkNode(i)=0 and ForkFree(i)=0 ) do
        i_biforca := 0
        cont:=0
        forall(j in NODES | (Graph(i,j)=1) ) do
            forall(t in TIMESLOTS | t>0 and (getsol(y(i,j,t)) > 0.001)) do
                cont := cont + 1
                break
            end-do
            if(cont>1) then
                i_biforca:=1
                break
            end-if
        end-do
        if (i_biforca = 1) then
            flag := 1
            writeln("ALERT FORK: node ", i)
            ITERATION+={i}
            !(9)
            sum(k in NODES | Graph(i,k) = 1) z(i,k) = 1
        end-if
```

```
    end-do

    if(flag=0)then
        forall(i in NODES) do
            forall(j in NODES | (Graph(i,j)=1)) do
                vij:=0
                vji:=0
                forall(t in TIMESLOTS | t > 0) do
                    if(getsol(y(i,j,t)) > 0) then
                        vij:=1
                    end-if
                    if(getsol(y(j,i,t)) > 0) then
                        vji:=1
                    end-if
                    if(vij=1 and vji=1)then
                        writeln("WARNING: bi-directonal flow between ", i, " and
", j)
                        !(11)
                        y(j,i,t) <= Q(i,j)*(1-z(i,j))
                        flag:=1
                        break
                    end-if
                end-do
            end-do
        end-do
    end-if
end-do
end_time := time(SYS_NOW)
writeln("OF = ", getobjval, " - time = ", end_time-start_time, " ms.")
end-model
```

Appendix B

Excel data

	B		B+S		B+S ConGen		B+S Matheuristics		B+S+F Matheuristics	
tmax	OF	run-time [s]	OF	run-time [s]	OF	run-time [s]	OF	run-time [s]	OF	run-time [s]
2	540	0,032	540	0,046	540	0,054	540	0,053	540	0,1
5	521,46	0,053	521,46	0,116	521,46	0,733	521,46	0,834	525,78	1
10	490,56	0,116	490,56	0,216	490,56	8,1	490,56	3,1	502,08	1,5
15	459,66	0,162	459,66	0,347	459,66	6,5	459,66	3,2	478,38	3,4
20	428,76	0,223	428,76	0,524	428,76	34,7	428,76	5,4	454,68	4,5
40	305,16	0,488	305,16	1,9	305,16	45,2	305,16	11,2	359,88	31,2
45	274,26	0,547	274,26	2,3	274,26	21,9	274,26	22,7	336,18	16,4
50	243,36	0,58	243,36	3,2	243,36	229,5	243,36	13,9	312,48	43,2
55	212,46	0,709	212,46	57,27	212,46	157,6	212,46	12,2	288,78	13
60	181,56	0,766	181,56	12,6	181,56	103,1	181,56	11,4	265,08	11,3
80	57,96	1,3	57,96	100,2	57,96	355,1	57,96	37,4	170,28	29,8
85	27,06	1,5	27,06	105	27,06	153,6	27,06	20,1	146,58	20
89	2,34	1,7	2,34	123,9	2,34	313,721	2,34	43,1	127,62	32,3
90	0	1,73	0,2	281,7	0,2	2402,2	0,96	55,3	122,88	105,3
91			0	125,5	0	137,4	0	39,1	118,14	37,3
105									51,78	42
110									28,08	51
115									4,38	54,1
116									0	285,6

Figure B.1: Data collected under Uniform scenario

	B		B+S		B+S ConGen		B+S Matheuristics		B+S+F Matheuristics	
tmax	OF	run-time [s]	OF	run-time [s]	OF	run-time [s]	OF	run-time [s]	OF	run-time [s]
2	540	0,015	540	0,053	540	0,053	540	0,084	540	0,054
5	540	0,053	540	0,115	540	0,185	540	0,5	540	0,254
10	540	0,1	540	0,2	540	1,4	540	2,2	540	1
15	522	0,179	525,6	0,417	525,6	24,6	525,6	10	528,48	6,1
20	496,02	0,221	505,8	1,5	505,8	45,2	505,8	15,9	515,52	15,1
40	391,62	0,618	426,6	6,5	426,6	418,7	426,6	78,8	438,48	32,2
45	365,52	0,818	406,78	21,5	406,8	329,6	406,8	91	418,68	62,1
50	339,42	0,844	387	7,8			387	117,3	398,88	66,8
55	313,32	1,1	367,2	31,7			367,2	48,6	379,08	38,1
60	287,22	1,2	347,4	19,6	347,4	922,2	347,4	196,1	359,28	128,7
80	182,82	1,8	268,2	40	268,2	1589,2	268,2	64,1	280,08	141
85	156,72	2,2	248,4	26,7			248,4	145,7	260,28	108
89	135,84	2,4	232,56	93,3			232,56	178,7	244,44	90,2
90	130,62	2,4	228,6	40	228,6	944,6	228,6	133,3	240,48	84,9
91	125,4	2,3	224,64	90,5			224,64	65,4	236,52	104,3
95	104,52	2,8	208,79	69,1			208,8	48,2	220,68	123,2
100	78,42	3,6	189	110			189	65,9	200,88	115,7
115	0,12	5,6	132,48	311,9			132,48	311	144,36	235
116	0	5,7	129,96	172,6			129,96	147,7	141,84	221,8
160			19,08	579,4		32400	19,08	429,9	19,08	642
165			7,68	467,6			7,68	185,3	7,68	786,8
170			2,28	705,5			2,28	391,5	2,28	461,7
172			0,12	788,1			0,12	1310,6	0,12	2056,1
173			0	537,9			0	558,1	0	2347,9

Figure B.2: Data collected under Concentrations scenario

Bibliography

[1] Laura Galli: Computational Complexity Theory
`http://www.di.unipi.it/optimize/Courses/RO2IG/aa1718/Complexity\_theory.pdf`

[2] Michael O. Ball. Heuristics based on Mathematical programming, *Surveys in Operations Research and Management Science* **16**,1 (2011) 21-38

[3] Jiang CS, Deng YF, Hu C, Ding H, Chow WK. Crowding in platform staircases of a subway station in China during rush hours, *Safety Science* **47**,7 (2009) 931-938

[4] Ye J, Chen X, Yang C, Wu J. Walking behavior and pedestrian flow characteristics for different types of walking facilities, *Transportation Research Record: Journal of the Transportation Research Board* **2048** (2008) 43-51

[5] Marco Ghirardi. Evacuation Models, *Polito*

[6] Claudio Arbib, Henry Muccini, Mahyar Tourchi Moghaddam. Applying a network flow model to quick and safe evacuation of people from a building: a real case, *University of L'Aquila*

Ringraziamenti

Al traguardo di questo percorso è doveroso spendere delle sincere parole di ringraziamento destinate a delle persone in particolare. Grazie al Prof. Marco Ghirardi, una di quelle persone che controbilancia le esperienze negative provocate dagli ostacoli dell'università, estremamente umile e disponibile, e che con queste sue qualità ha decorato il finale della mia carriera accademica. Grazie alla pazienza, alla generosità e al sostegno della mia famiglia che non ha mai smesso di partecipare positivamente ai miei progetti, fondamentale. Grazie a tutti quei miei amici e quelle persone che mi hanno dedicato un qualsiasi momento attraverso una qualsiasi azione volta a farmi sorridere, a confortarmi, a motivarmi e a migliorarmi. Grazie dunque a chi ha remato a mio favore perché mi ha reso forte ma grazie anche a chi non ha avuto fiducia in me perchè mi ha reso brillante.