Università
Ca'Foscari
Venezia

Corso di Laurea magistrale in
Economia – Economics

Curriculum *Models and Methods in Economics and Management*

Tesi di Laurea

Ca' Foscari
Dorsoduro 3246
30123 Venezia

Crowdsourcing in real time. Number of workers and Expected Waiting Time: the trade-off in the *Retainer Model.*

Relatori

Ch. Prof. Paola Ferretti

Ch. Prof. Andrea Ellero

Laureando

Giovanni Furlanetto

Matricola 838665

Anno Accademico

2012 / 2013

To my always supportive families

Alle mie famiglie, sempre di sostegno

Title of the thesis:

Crowdsourcing in real time. Number of workers and Expected Waiting Time: the trade-off in the *Retainer Model.*

Summary

Index of Acronyms

EWT	……	Expected Waiting Time
ETC	……	Expected Task Cost
FOC	……	First Order Condition
SOC	……	Second Order Condition

Index of Figures

Index of Tables

Index of Equations

Introduction

Crowdsourcing, as a "*participative online activity in which an individual, an institution, a non-profit organization, or company proposes to a group of individuals of varying knowledge, heterogeneity, and number, via a flexible open call, the voluntary undertaking of a task*" (Estellés-Arolas and González-Ladrón-de-Guevara, 2012), is becoming more and more popular.

The application of crowdsourcing in real time is something that the crowdsourcing world is still studying and trying to solve, knowing its potentialities concerning the future of marketing, security and some phone applications that would simplify everyday life.

If crowdsourcing is the encounter between different intelligences, realtime crowdsourcing represents the desire that this meeting produces a response in the shortest possible time.

The *Retainer Model* is presented in this thesis. This model is the first, and the only one, that mathematically analyzes crowdsourcing in real time.

This thesis wants to study the characteristics of this model, analyzing and studying the two parts of its trade-off: the function of total cost of hiring some workers and the *Expected Waiting Time* for a new incoming task. The model assumes that it is possible to recruit paid crowdworkers and obtain from them a response in few seconds.

This thesis finds the reasons of the existence of a minimum stationary point that represents the minimum number of workers optimizing the *Retainer Model's* trade-off. If on the one hand we want the *Expected Waiting Time* to be the as low as possible (reaching this, increasing the number of workers) on the other hand we

do not want the number of workers in the pool to become too large because it increases the total cost.

After the identification and the study of the reasons and the proofs of the existence of this minimum; an algorithm, to automatically find this optimal number, is shown. This algorithm, using a cyclical process, is able to find the optimal number of workers for given values of the parameters of the system.

In the **first** chapter is described the main theme of the thesis: crowdsourcing. The main characteristics of it and the important features, that are increasing its popularity and importance, are presented.

The **second** chapter is about realtime crowdsourcing. This is a new theme because the application of the real time processes'rules in the crowdsourcing field is something that researches are trying to solve, but it is currently used in many applications.

The *Retainer Model* and its development are presented in chapter **three**. It took two papers to build it, describe its characteristics and find mathematical substance for the finding of the optimal number of workers.

Chapter **four** presents the two parts of the *Retainer Model*'s trade-off. The *Expected Waiting Time* for a new incoming task in one hand and the function of total cost of hiring some workers on the other hand.

In chapter **five** there are my contributions on this theme: I find all the proofs that determine the existence of a stationary point and the evidence which state that this point, representing the optimal value in term of crowdworkers in the pool, is a minimum. This because, as the Clay Mathematics Institute says, "*[...] we require a proof in mathematics. A proof gives certainty, but, just as important, it gives understanding: it helps us understand* why *a result is true*"[1]. After this, a faster and more precise algorithm, usable to find this minimum point, has been created

[1] Riemann's 1859 Manuscript description
[http://www.claymath.org/millennium/Riemann_Hypothesis/1859_manuscript/]

using a cyclical approach. Now it is more easy to find this prophetic number in a more precise manner.

The **conclusions** show the main passages of the thesis, but also some contributions for further studies. One of these is a new possible function of total cost, which has origin in the formula of the *Retainer Model* and could be useful, explaining better the entire cost suffered by the system, in my next studies on this field.

Math and network are the main fields of this thesis. One the definition of concreteness and the other the explosion of the community without limits. This is why I have chosen this thesis: to apply mathematical models to support the use of the network and to use mathematics to improve and optimize systems that rely on the network.

We have to exploit this so good fortune. It is on the web and with the web that we can, smartly, find new solutions to old problems. The web is a large network all the more useful as it is consist of many nodes.
A network is a virtual place where goals, ideas and solutions can come together to produce the best end result.

Chapter 1:
Crowdsourcing: how to develop common knowledge through sharing and participation

Why to write about crowdsourcing and why it is important to know and study is highlighted in the preface to the Italian version of the book by Jeff Howe, "CROWDSOURCING: Why the Power of the Crowd is Driving the Future of Business" (J. Howe), which is, now, my favourite quotation. It states:

"Some people think: I am the best of them all in a given thing. Full stop. They are many. They are often mistaken. Some even know they are not the best, but do not care, no one will find out and never mind if the job done at the end will not be the best. […] And then there are those who know that, outside, somewhere, there is definitely someone better. Or someone who can improve your idea. Or many that can help you focus on it better and in a winning way. It is not an act of faith, or lack of confidence in themselves: it is logic, indeed statistics".

And statistics always wins. It is from this victory that crowdsourcing was born: we know that outside these walls there is a crowd of people that can help us. But this crowd is not the entire rest of the world, but only a small portion. The portion of human living in the planet who use the Internet. Jeff Howe, in his book, names this portion: *the Billion*, due to the fact that this is the number of people online in the world.

So crowdsourcing world can have one billion people – on a total of seven billion – who can contribute in different ways to various crowdsourcing projects. This billion people is "*dispersed among innumerable over-lapping online communities, composed of people whose interests align, however temporarily. These communities aren't so different from those we know from the offline world*" (Howe).

In order to understand better and more deeply this world, we have to ask what crowdsourcing is.

The definition that the *Daily Crowdsource*[2] gives is: "*Crowdsourcing is the process of getting work or funding, usually online, from a crowd of people. The word is a combination of the words 'crowd' and 'outsourcing'. The idea is to take work and outsource it to a crowd of workers*"[3].

The principle is ancient: two brains are better than one. Moreover, this principle belongs to a sociological theory called "*the Wisdom of Crowds*". It express that a group of individuals would be able to provide an adequate and valid answer to a question better than an expert is able. This theory bore in the 1907 from Francis Galton who wrote *Vox Populi*, which is the first article on this theme published in *Nature*. He showed that "*the crowd at a county fair accurately guessed the weight of an ox when their individual guesses were averaged*"[4].

Crowdsourcing "*combines the efforts of crowds of self-identified volunteers or part-time workers, where each one on their own initiative adds a small portion that combines into a greater result. Crowdsourcing is different from an ordinary outsourcing since it is a task or problem that is outsourced to an undefined public rather than to a specific, named group*" (Wikipedia).

The first appearance of this term in the scientific literature has been the first February 2008, in the paper by Braham Daren that defines it as an "*online, distributed problem-solving and production model*" (Daren). The main point of crowdsourcing is the power of making actions available to anyone online, with quick access for a huge number of users, collecting data and rewarding theme.

[2] An "open-format website that aims to educate the public on the topic of crowdsourcing" [http://dailycrowdsource.com/about]
[3] Web site of the Daily Crowdsource with this definition [http://dailycrowdsource.com/crowdsourcing-basics/what-is-crowdsourcing]
[4] Quoted in Wikipedia [http://en.wikipedia.org/wiki/The_Wisdom_of_Crowds]

One of the most beautiful comparison is that between the *Encyclopaedia Britannica*[5] and *Wikipedia*[6]. The first, with data until 2012, used 4,000 experts to write 80,000 articles and spent 200 years to develop. The second has more than 1.6 million articles and more than 100,000 amateurs who contribute to the development of *Wikipedia* in six years. These numbers are incredibly different! Moreover, *Wikipedia* can rely on a continuous real time update while *Encyclopaedia Britannica* updates only on a yearly basis.

What about the quality aspect? This aspect opens more than one parenthesis because a lot of studies tried to check and compare the quality of *Wikipedia*. The prestigious journal *Nature*, in an investigation that caused a stir in 2005, shows that the quality of *Wikipedia* is not far from the quality of *Britannica* (that costs, discounted, $ 1,295). In "*Special Report Internet encyclopaedias go head to head*", by Jim Giles, various experts, kept in the dark about the origin of specific articles, researched errors and inaccuracies on 42 subjects taken from both sources. The result is a substantial balance: four serious mistakes for the *Britannica* and the same for *Wikipedia*. For *Nature*, discover "the quality of Wikipedia in science" was a "nice surprise"[7].

The reason for this alignment is not the fact that the unknown authors of *Wikipedia* are gotten better over time, but that their number has increased: increasing the "*critical mass*" increases the likelihood that, within an article, you will find a much greater number of contributors, thus increasing the final quality.

This is the reason for the (*alas!*) still low quality of *Wikipedia Italia*: the lack of this "*critical mass*" prevents the growth of quality and allows the publications of large number of "rubbish" articles often containing glaring mistakes.

[5] Web site of Encyclopedia Britannica [http://www.britannica.com/]
[6] Web site of Wikipedia [http://www.wikipedia.org/]
[7] Web site of Nature with the article: "Special Report Internet encyclopaedias go head to head" [http://www.nature.com/nature/journal/v438/n7070/full/438900a.html]

Thus, crowdsourcing is an online and distributed model producing solutions for problems where users (the so-called crowd) are typically constituted around a community, based on web sites. They provide solutions or produce contents for the website. The crowd can also select the solution, finding the best among them all. The solution becomes property of the crowdsourcer: the requester who has posed the problem. Individuals in the crowd are usually remunerated, but sometimes they participate only for the intellectual stimulation or to be useful and help someone in producing a service.

Two important revolutions in the field of technology have taken place. The first revolution, the revolution of the PC, was the result of powerful and cheaper microprocessors which started with the *IBM PC*. The second revolution, the revolution of communications, is the result of inexpensive broadband communications starting from the Internet.

Nowadays, the critical mass of connections between people is the central point, no longer the technology or innovation (which still are important points). We have achieved this critical mass thanks to the ease of use, availability, cultural change and, above all, to the effect that the web has brought. The two laws that link these two important revolutions are the *Moore's Law*[8] and the *Metcalfe's Law* [9]. The first states that every two years the processing power doubles at the same cost, and the second states that the utility of a network is a function of the square of the number of potential users.

Every network and every social software have a life that can be divided into phases. This is done by the *Gartner Hype Cycles* that "*provide[s] a graphic representation of the maturity and adoption of technologies and applications, and how they are potentially relevant to solving real business problems and exploiting*

[8] Web site of Computer History with Moore's Law
[http://www.computerhistory.org/semiconductor/timeline/1965-Moore.html]
[9] Web site of Princeton University with Metcalfe's Law
[http://www.princeton.edu/~achaney/tmve/wiki100k/docs/Metcalfe_s_law.html]

new opportunities"[10]. These graphs show how and when a technology or an application will evolve over time. Thus, it is easier to manage and use the technology in a specific business, knowing the phase where it is at that specific moment.

We can check the evolution, in terms of phases, crowdsourcing has undergone, starting with the position occupied in 2008.

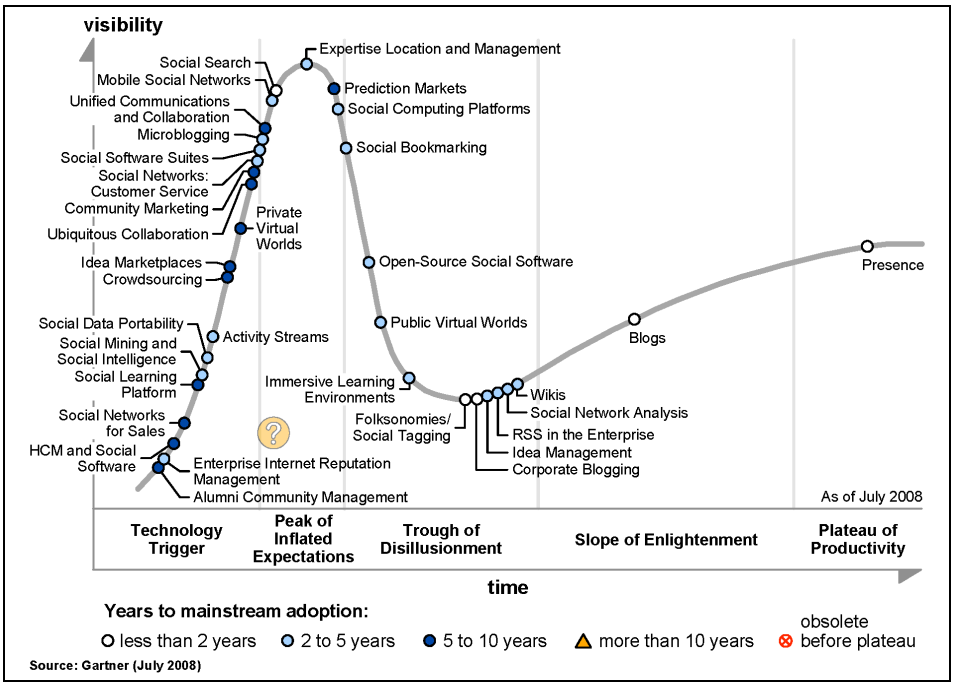

Figure 1: Gartner, Hype Cycle for Social Software in 2008.
Crowdsourcing in the first phase

As we can see, in 2008, the year of the first appearance of this term in scientific literature, crowdsourcing was in its first phase: the *Technology Trigger* phase.

In this phase, a potential technology breakthrough is presented on the market. There is no evidence of profitability and the new products are prototypes.

[10] Gartner's methodologies for the Hype Cycles [http://www.gartner.com/technology/research/methodologies/hype-cycle.jsp]

As we can see from the colour of the point (dark blue) in the graph above, *Gartner*, in 2008, said that mainstream adoption could required between five and ten years.

Nowadays we are in this period: 2013 is the first of these five years. Thus, it is admissible to ask where crowdsourcing is now and in which phase *Gartner* set it.

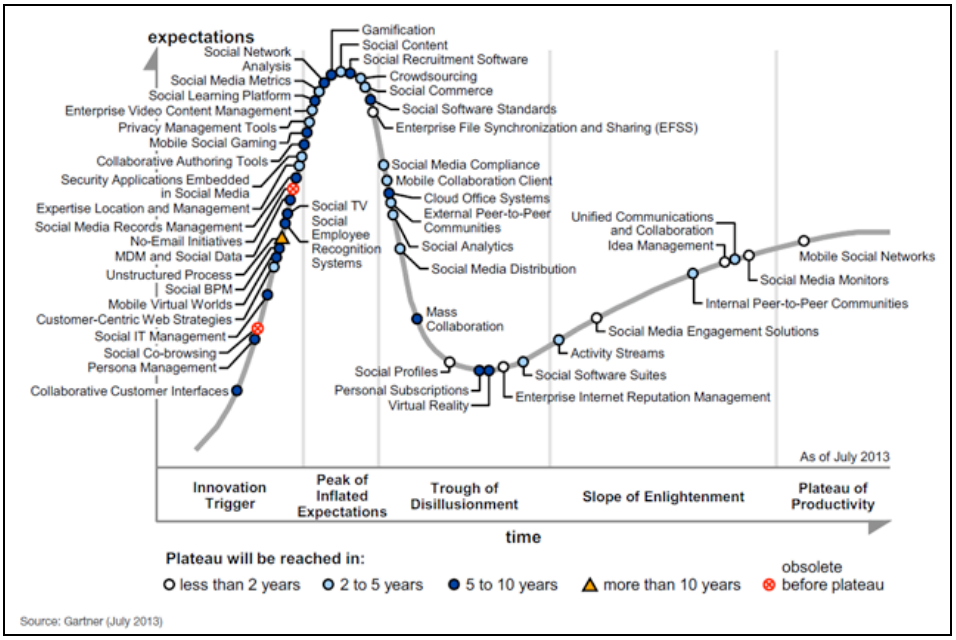

Figure 2: Gartner, Hype Cycle for Social Software in 2013.
Crowdsourcing in the second phase

Gartner sets crowdsourcing in the second phase: the *Peak of Inflated Expectations* phase. Here "*early publicity produces a number of success stories - often accompanied by scores of failures. Some companies take action; many do not*"[11]. We can also see that the time to reach the mainstream adoption is decreased: now it is in only between two and five years.

[11] Web site of Gartner's methodologies [http://www.gartner.com/technology/research/methodologies/hype-cycle.jsp]

So the life cycle of crowdsourcing is rapidly rising and this is due to the benefits that it gives to those who use it. Some of these benefits are, for example, that the problems can be analyzed and compared at low cost, the requester may obtain products, processes and marketing efforts suggested by the users. Some of the key elements of crowdsourcing are:

- *diversity of opinions*: each person must have a different opinion from the others (even an eccentric one);
- *independence*: people's opinions should not be affected by others;
- *decentralization*: people must be able to create *their* opinion from *their* environment;
- *aggregation*: determine mechanisms for aggregating opinion and judgments into a collective decision.

Crowdsourcing is an approach, a method that can be used in very different ways and in a lot of different fields. This represents the multiplicity of opportunities that crowdsourcing can deal with. Some of these are: *microtasks*, *graphic design*, *macrotasks* and *crowdfunding*.

The first application in this list is *microtasks*. The main point is the small size and the repetitiveness of the tasks done by the online crowdworkers. As the *Daily Crowdsource* suggests, microtasking "*is a great place to start when [one is] new to crowdsourcing. Having a crowd of workers complete "small and repetitive" tasks can significantly lower costs and save time*"[12]. Some of this microtasks are, for example, the creation of contents, the description of some products, the collection of information researched on the Internet, the translation of sentences or portion of documents, the organization of photos and documents (tagging, sorting, labelling, digitizing, scanning) and the transcription of audio files.

[12] Daily Crowdsource's article listing the different fields in which crowdsourcing can be useful [http:// dailycrowdsource.com/crowdsourcing-basics/what-can-you-crowdsource]

The second application is *graphic design*. It is the most common task to crowdsource[13]. It is important because it gives to the requester something that is completely original and customized. The most famous, used and important examples are the designs of new logos or their restyles for companies, the creation or redesigning of web pages, web sites or business cards.

The third application is *macrotasking*. Because macrotasking involves more people on a complex project, on its research and on its development, communication and organization are extremely important to manage the different skills of these crowdworkers. The most famous categories for macrotasking application are: the development of eco-green solutions, social projects, medical and public health, productivity innovation projects.

Crowdfunding is the last, but the best-known, application. By definition, given by *Forbes*, it is "*the practice of funding a project or venture by raising many small amounts of money from a large number of people, typically via the Internet*"[14].

A report by *Massolution* about the crowdfunding industry states that the "*crowdfunding platforms raised $2.7 billion and successfully funded more than 1 million campaigns in 2012. Massolution forecasts an 81% increase in global crowdfunding volumes in 2013, to $5.1 billion*"[15]. At the moment, there are more than 500 crowdfunding web sites and more than 9,000 domains related to crowdfunding all over the word: this shows the size of this "*new*" item[16].

Usually, some of the projects that can be crowdfunded are artistic projects, like movies, documentary films, music albums or books. Other kind of projects, financed by crowdfunding, are charitable projects that require resources to travel and assist someone.

[13] Web site of Design Crowd [http://www.designcrowd.com/crowdsourcing]
[14] Web site of Forbes, Tanya Prive, What is crowdfunding [http://www.forbes.com/sites/tanyaprive /2012/11/27/what-is-crowdfunding-and-how-does-it-benefit-the-economy/]
[15] Web site of the Crowdfunding Industry Report by Massolution [http://research.crowdsourcing.org/ 2013cf-crowdfunding-industry-report]
[16] Web site of Forbes, Ryan Caldbeck, "Crowdfunding trends" [http://www.forbes.com/sites/ryancaldbeck/2013/06/23/crowdfunding-trends-which-crowdfunding-sites-will-survive/]

This presentation is not meant to be a strict description of crowdsourcing. It describes the main characteristics of crowdsourcing, the theme of this thesis.

It can be useful to understand the importance of this theme in the present economy and, above all, for the future. It represents the next step in the communication field and in the participation: we need to understand that there is an incredible amount of people (a *billion* and it will increase) ready to help us. Einstein said that "*creativity is born from anguish. Just like the day is born from the dark night*". We have the tools to overcome a lot of problems, sharing ideas, collaborating and joining forces – because the power of sharing and participating is greater than that of a single person.

Chapter 2:
Realtime Crowdsourcing: a new application in a new theme

After the chapter about the illustration of crowdsourcing in general and its importance, now the application of crowdsourcing in real time are presented. This field is very mysterious, not yet studied in depth, because still very recent.

What does real time mean? Only by knowing this definition can we understand when and why it is possible to study its application to crowdsourcing (anticipating its applications in the near future).

By definition, indicated by Professor John Stankovic[17], "*Real time systems are defined as those systems in which the correctness of the system depends not only on the logical result of computation, but also on the time at which the results are produced*". The expression "*real time"* summarizes two fundamental concepts:

1. TIME: the validity of the results of a calculation process depends on the time within which these results are produced;
2. REAL: the system time must be equal to the environment time in which the system operates.

Real time does not mean *fast*. The term *fast* has a relative meaning which makes little sense if it is not concerning the environment in which the system operates. Therefore, the characteristics of a realtime system are closely related to the environment in which the system operates: the objective of a *fast* process is to

[17] Web site of the University of Virginia – Engineering [http://www.cs.virginia.edu/people/faculty/stankovic.html]

minimize the average response time of a set of processes; the goal of a realtime process, instead, is to satisfy the timing requirements of each individual process or task.

The interest in the realtime system is motivated by the many applications that require this type of processing, for example, the regulation of nuclear and chemical sites, the control of complex manufacturing processes (as robotics), military systems or space missions, the flight control systems of airports, the control systems of the traffic and, as in this thesis, the telecommunication systems.

So a realtime process is characterized by a predetermined time expiration, called *deadline*. A result produced after this deadline is not only late, but can be dangerous. Depending on the consequences resulting from a missed deadline, realtime processes are usually divided into three types: *hard*, *firm* and *soft*. These are:

- *hard realtime process*: if the violation of the deadline results in a catastrophic effect on the system;
- *firm realtime process*: if the violation of the deadline does not entail catastrophic effects, but the usefulness of the results decreases with the increasing of the delay of the deadline;
- *soft realtime process*: if the violation of the deadline does not affect the proper functioning of the system.

Responding to an event in real time means responding at a speed that can be predetermined, whatever the operating context of the equipment.

Why could it be important to study crowdsourcing in real time? Because there are a huge number of applications that are trying to decrease the time between the instant that a request (a task) arrives and the instant that this request is solved.

This is well written by Casey Armstrong, the founder of VineStove.com[18], when he writes: "*Real-time crowdsourcing of tasks is something that the crowdsourcing world is trying to solve. It is a difficult problem, and the industry is still learning how to harness the crowd in basic ways*"[19]. So, this theme is something that the industry is trying to solve, but it is currently still using and developing.

An important sentence on this theme is one of the three predictions – that Casey Armstrong "*bets*" – about where crowdsourcing will go in the coming years. It states: "*in the next 5-7 years real time crowdsourcing will be the norm*"[20] and shows the growing importance of this field.

As I wrote above, there are not a lot of papers and models that explain this connection, but as a first example I can mention the paper by Limpaecher, Feltman, Treuille and Cohen, titled "*Real-time drawing assistance through crowdsourcing*".

Their paper presents a method that uses a "*crowdsourced drawing database*" (collected thanks to a mobile app game designed for this purpose) to build a system able to understand when an incoming image was too different from the others. They do so in order to build an automatic "*artistic consensus at the stroke level*".

The algorithm was able to find which strokes were too different from the average of the rest of strokes, comparing thousands of images of the same person drawn by different "*crowdartists*". The authors present a "*simple stroke-correction method"* that used this algorithm for correcting strokes in real time.

[18] A crowdsourcing web site that exchanges microtasks "for donations given to nonprofits" [http://www.vinestove.com/#!about/c1j4r]
[19] Web site of Quora, "best source for knowledge" [http://www.quora.com/Crowdsourcing/Whats-the-best-way-to-generate-real-time-crowdsourced-answers]
[20] Daily Crowdsource's article by Casey Armstrong [http://dailycrowdsource.com/8-resources/index.php?option=com_content&view=article&id=1140:3-crowdsourcing-predictions-that-are-totally-going-to-come-true-by-2020&catid=25:discussion&Itemid=214]

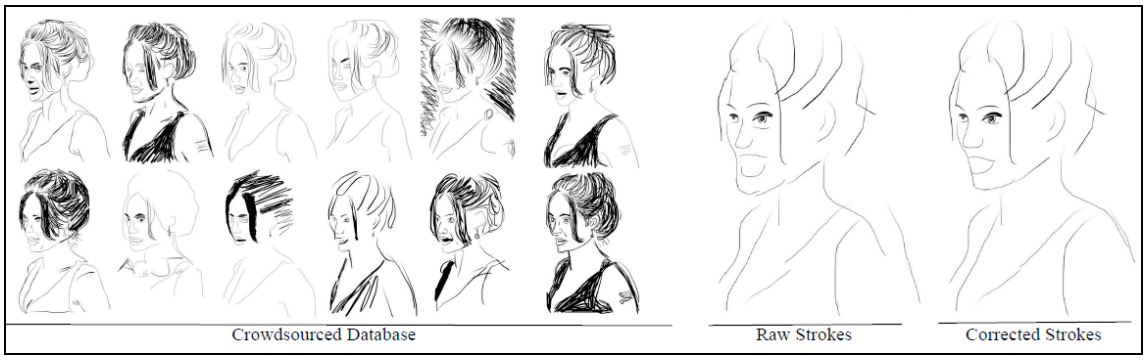

Figure 3: the automatic correction done by the "simple stroke - correction method" in Limpaecher's paper.

As Figure 3 shows, the algorithm is able to correct strokes too different from the original picture (so too different from the crowdsourced database) and it does this operation in real time.

Another application of realtime crowdsourcing has been applied in everyday traffic jams. Drivers are often taken by surprise about traffic jams, congestions and traffic accidents. Realtime crowdsourcing can be (and, actually, it is) a great help to overcome all these problems: through timely informations, it is possible to help many drivers to avoid certain routes and find alternative ways to reach their destinations.

This is what *TrafficTalk*[21] is trying to do: it aims to help drivers with traffic informations in real time, offering them suggestions via mobile phone on the best possible routes to take and those to avoid.

This new free phone app crowdsources information from drivers in a specific area updating in real time all the drivers' iPhones with the information they send about the condition of the traffic.

"*TrafficTalk provides real time, live information on the present condition of the road*"[22] as some radios do, but more slowly and not live (or in real time, as they

[21] Web site of TrafficTalk [http://www.traffictalk.info/]

say). For all these reasons, realtime crowdsourcing is helping drivers simplifying their movements.

Realtime crowdsourcing could be important not only in everyday life, simplifying it with new *phone apps* and new applications, but also in a very important field: Public Security.

A first application of realtime crowdsourcing was implemented after the tragedy of the Boston Marathon on April 15, 2013. The *usual* application (*with which users try to find something or someone in photos or videos)*, honestly, failed: chaos, rivers of adrenaline and confusion exacerbates inaccurate information and accused innocent people of being bombers, ruining their lives forever.

In fact, an *alternative* application of crowdsourcing worked well. The point is that common people cannot substitute the Police and FBI, but what it is possible to do is to use the crowd in simple tasks and leave the Police to do their job. In the future, we will see more and more sophisticated data analysis tools able to analyze the metadata stream (*information describing a set of data*) from iPhones and other devices to map the behaviours of large crowds identifying particular "events" with great precision (monitoring information from passive viewers).

The application used to help the Police in the Boston Marathon bombing, was *CrowdOptic*. This application, originally born for advertisers and commercial use, "*can collect and analyze a wealth of data from smartphones to help determine that – noting where smart phone cameras are pointed and correlating that with social media and other information*"[23]. This tool helped investigators analyze metadata from videos and photos, instead of asking unreliable and excited people to look for some details in pictures. *CrowdOptic*, using an algorithm, can identify "points of focus" of a crowd extracting information from images and using triangulation. The idea is to overcome the "geolocation", with a sort of "focus awareness" found using the individual focus of the members of that crowd.

[22] Web site of Daily Crowdsourc [http://dailycrowdsource.com/20-resources/projects/346-real-time-crowdsourced-road-tips-to-avoid-traffic-jams]
[23] Web site of Security Ledger [https://securityledger.com/2013/04/crowdsourcing-fail-in-boston-but-can-technology-fix-it/]

CrowdOptic's technology played a key role in helping the authorities: it was used to put together all the images that contain the position of the bomb in the area, before and after the explosions and give investigators not only a view of the scene, but also an image of the points of interest that people might have captured. With this type of crowdsourcing, it was possible to help the Police in finding the bombers using the crowd as a single entity and using the information of this entity to collect and analyze evidence.

The first thing that we can note in every one of these examples is the improper use of the term *real time*. It is not used as the definition wants: there is never fixed *deadline* for the answer, but in general this deadline is "*as soon as possible*". The *deadline* is tacitly set within a range of few seconds and this is because we are referring to crowdsourcing and not to computer science. This should not be surprising or worrying: also in telecommunications and computing the term *real time* is used with this meaning. The definition that telecommunications and computing accept is: "*the term [real time] implies that there are no significant delays*"[24].

So, if crowdsourcing is the encounter between different intelligences, *realtime crowdsourcing* represents the desire that this meeting between produces a response in the shortest possible time, or, even better, *without any significant delay*.

As I anticipated above, not many papers discuss this theme, because it is very recent and difficult to analyze.

In this thesis, I will focus on a particular model that was proposed and developed in two recent papers. This model is important because it is the only one (to my knowledge) that gives mathematical substance to this theme. It gives to this theme meaning and rules, using mathematical concepts and formulas to achieve a result, linking *optimization* and realtime crowdsourcing!

[24] Telecommunication engineering category by Wikipedia, page of Near Real Time [http://en.wikipedia. org/wiki/Near_real-time]

The model is called "*the Retainer Model*" and the two papers are: "*Crowds in two seconds: enabling realtime crowd-powered interfaces*" (Bernstein, Karger, Miller and Brandt, 2011)[25] and "*Analytic Methods for Optimizing Realtime Crowdsourcing"* (Bernstein, Karger, Miller and Brandt, 2012)[26].

As we can see, these papers are very recent: one of 2011 and the other of 2012. This expresses the freshness of this theme and suggests how much is possible to do on this field.

In the next chapter, the model is described and in the following one I will optimize it.

[25] Web site of ACM Digital Library [http://dl.acm.org/citation.cfm?id=2047201]
[26] Web site of Cornell University Library [http://arxiv.org/abs/1204.2995]

Chapter 3:

The *Retainer Model*: a mathematical model for realtime crowdsourcing

As I wrote at the end of the previous chapter, the two papers are complementary: the first paper presents the *Retainer Model* and the second optimizes it, introducing some concepts of Queueing Theory.

We know, from the previous chapters, that a crowdsourcing system runs when an incoming task finds some crowdworkers available to work on it. Incoming tasks create a list and this list is emptied, task by task, by the crowdworkers.

The first paper, "*Crowds in two seconds: enabling realtime crowd-powered interfaces*" (Bernstein & al., 2011), wonders whether it is possible to lower significantly the *crowd latency*. *Crowd latency* is the period of time that affects a crowdsourcing system in the search for a new worker to answer a call in order to solve a task.

As I said before, crowdsourcing is a tool useful not only to the requester who has time for a response, but its popularity and importance are increasing for the requesters who operate in short periods of time.

To get an idea of the amount of time and the actual value to be given to this time we can mention: "*VizWiz: Nearly Real-time Answers to Visual Questions*" (Bigham et al., 2010) [27]. This paper describes an iPhone app that allows blind users to get quick answers to questions about their surroundings. A user with some visual impairments takes a picture and records a question with its mobile phone and then sends all these to anonymous crowdworkers. Once this task is received

[27] Web site of the paper [http://hci.cs.rochester.edu/pubs/pdfs/vizwiz.pdf]

from these services (like object recognition software, *Twitter*, *Facebook* or e-mail contacts), the answer is sent to the requester's mobile phone.

This paper is useful in our case, because, through a new mobile application, it promises an answer within 30 seconds.

Thus, the vision of a user pressing a button and having the results (produced by a crowd) in real time starts from new approaches and new concepts. One of the most important concepts is the so called *on-demand synchronous crowd*.

With this type of crowds, built in order to aggregate more than one person per question, we can have several advantages: we can reduce the recruitment time and we can also be able to solve more complex searches in less time. In effect, even the fastest employee cannot get results in real time, and, above all, it is not easy to find workers willing to answer quickly in real time. These are the reasons why multiple users working together (a *synchronous crowd*) could be a good answer to our desire for realtime crowdsourcing.

The model obtained by Bernstein & al. in "*Crowds in two seconds: enabling realtime crowd-powered interfaces*" is the *Retainer Model*. With this model it is possible to engage crowdworkers in advance, leaving them to stay in a so called virtual pool. In this virtual pool they are paid also if no tasks are present and they are free to work on other things; but, when a task arises, they are obliged to put themselves to work. With this pool of *speedy workers*, the authors produced a series of experiments, using not only the concept of this *Retainer Model*, but also the *rapid refinement algorithm*.

This *rapid refinement algorithm* is the process used by the authors to analyze the various responses to the experiments, gathered by the crowdworkers, and find the best answer trough them.

Adrenaline, *A|B* and *Puppeteer*[28] are experiments based on the *Retainer Model*. The *rapid refinement algorithm* is used to guide the search towards the optimal solution through the method that I call "*sieve*". In these experiments, the best answer is the answer with the highest frequency: it is given by a majority of votes calculated from time to time. After x seconds, the system deletes the answer with the lowest frequency and, using this "*sieve process",* the "bad" answers are removed gradually in order to get the final (optimal) answer.

What it is important for the purpose of this thesis is to understand the construction of the *Retainer Model*. The *Retainer Model* addresses two questions: *how to optimize the response time minimizing the cost of the pool* (because the number of workers in the pool and the cost of having this pool are directly proportional) and *how to maintain a high response time.*

The worker in the pool has to focus on the task as soon as it is asked. To do this, the best incentive is to pay an extra amount of money, allowing these workers to continue with other activities while waiting.

The model must, therefore, *ensure quick responses*, *be cost-efficient* (not wasting money unnecessarily hiring too many workers) and *maintain the ability to respond after some time*. A way that the authors found is to introduce an alert sound needed to awake the workers and put them to work immediately in front of the screen. The authors even mention in the paper a new payment as a reward for the fastest reaction (but they do not describe it).

The structure of the cost is very innovative and interesting, because the requester pays workers even when they are not working, compared with the pay to repeat old tasks at the top of the list of all tasks (which is the usual way to sort the tasks on websites such as Amazon Mechanical Turk[29]).

The cost, in this first paper, depends on three factors: the *right arrival time* τ, the *empirical distribution arrival* P and the *desired number of workers* w.

[28] These are the names of the three experiments built by the authors
[29] Web site of Amazon Mechanical Turk [https://www.mturk.com/mturk/]

The *number of retainer workers* r the requester needs, then, is defined as:

$$r = \frac{w}{P(arrive \leq \tau)}$$

Equation 1: number of retainer workers in the first paper

So, the total hour cost C_{tot}, is r multiplied by the cost per minute (the so-called *retainer wage*) and the base cost for the task C_{base}, that is:

$$C_{tot} = r\left(\frac{60}{\tau} \cdot C_{base} + 60 \cdot C_{ret.wage}\right)$$

Equation 2: Total Cost in the first paper

After describing the general characteristics of the *Retainer Model* and defining this first cost structure, the authors describe the *rapid refinement approach*, showing an algorithm using an experiment called "*Adrenaline*". This algorithm is a process by which instead of taking a single photo, an entire video is recorded. The video is divided in f parts called frames. In order to find the best photo frame, the less interesting parts of a video are skimmed obtaining frames of the video that are more interesting and intense. The final frame is declared as the best and most interesting photo frame of the whole video.

The *Retainer Model* in this paper is not optimized, it is only presented and described. This is the reason why, after one year, the same authors wrote the paper: "Analytic Methods for Optimizing Realtime Crowdsourcing".

This second paper analyzes, in a more formal manner – but again descriptive – the *Retainer Model* looking mathematically for the optimal number of workers to be employed in the virtual retainer pool.

This paper wants to show mathematically that is possible to recruit crowds and obtain a response from them in few seconds (so, as I wrote before, in *real time!*) as the previous paper established only by words.

The experiments and the studies of the previous paper did not optimize the *trade-off* between *costs* and *performance*. Through the use of the Queueing Theory, Bernstein, Brandt, Miller and Karger analyzed the relationship between the *Expected Waiting Time* (the time that a requester has to wait when he commits a task to the crowd) and the *Total Cost that the requester must sustain* (maintaining a pool of n workers) in the *Retainer Model* for realtime crowdsourcing.

No-one had ever studied before the relationship that exists between the pool size in the number of workers (and so the Total Cost) and the time of reply of this pool of workers (and so if and how long a requester has to wait for an answer).

The paper analyzes, in the *first* part, "*a simple algorithm that allows requesters to minimize their cost*"[30]. This "*algorithm*" seems not easy to use because it carries out its assignment only graphically and so it does not give a precise result, being also time consuming.

In the *second* part of the paper, the authors described some improvements to the *Retainer Model*, to reduce the *Expected Waiting Time* – these are: the *Retainer Subscriptions*, *Combining Retainer Pools* and the *Precruitment*. With these improvements, the median response is of only 500 milliseconds (half a second).

The *first* part of the paper is the most interesting part for the purposes of this thesis. The description and the application of the Queueing Theory in the *Retainer Model*, in order to optimize the trade-off between the *Expected Waiting Time* and the *Total Cost* of the retainer pool.

The *Retainer Model* pays workers (put together in a common virtual pool) a small wage to respond on demand: the workers accept the task ahead and they are paid to keep the computer on.

[30] Page 1 of the paper "*Analytic Methods for Optimizing Realtime Crowdsourcing*" (Bernstein & al., 2012)

The mathematical model wants to predict how long a task must wait, or rather, how many workers should be employed to minimize this waiting time (while avoiding unnecessary costs).

How does this model work? Upon the arrival of a task a worker leaves the pool, where it was plugged in, and works on the task. At this point, the system looks for another crowdworker to replace the one who is now busy with the task.

The goal is to create a pool large enough to cope with the number of incoming tasks. So, increasing the number of crowdworkers in the pool minimizes the probability that the retainer pool becomes empty, avoiding the case where zero workers are available on new incoming task (and so avoiding the case of a possible non realtime answer).

How to evaluate the arrival of the tasks and the time to process them?

An answer may be provided by the Queueing Theory. The Queueing Theory aims to develop models for the study of the waiting phenomena that may occur in the presence of a request for a service – the so-called queues[31]. It is common to find results obtained by the Queueing Theory, for example, in flexible manufacturing systems, processing systems, transport systems or communication systems/data transmission.

Customers who arrive and cannot find a worker free on arrival are arranged in an orderly manner – the queue. They are served according to certain service disciplines. The queue is formed by the customers waiting to be served and it is usually assumed that each customer leaves the queue immediately after his service is completed. The number of workers is known and constant and it is set at the project level. Usually they have identical characteristics, they can always work in parallel and they can never remain inactive in the presence of customers in queue.

[31] Web site of University of Trieste – Engineering Department [ftp://docenti.ing.units.it/arc_stud/Pesenti/Nettuno/CodeDispense.pdf]

From a dynamic point of view, the queue is basically made of two stochastic processes: the *arrival process* of the customers and the *service process*.

The *arrival process*, which describes the manner in which customers arrive, in general, is a stochastic process. It is defined in terms of the distribution of the intertemporal arrival rate, that is, the interval of time between the arrival of two subsequent customers. To obtain models capable of being analyzed it is usually assumed that the *arrival process* of the service is stationary, or that its statistical properties do not vary over time.

The *service process* describes the manner in which each worker provides the service, in particular, it defines the duration of this action and it is usually a stochastic process. It is defined in terms of the distribution of the different workers' service times.

The *service process* is fuelled by the *arrival process* of the customers. Consequently, the *arrival process* conditions the general *service process*. A client can only be served if he has already arrived. Moreover, when it is not explicitly stated to the contrary, the *arrival process* is considered independent of the *service process*.

All the elements that define a queue are highlighted in the notation A/B/c/K/m/Z called the *Kendall's notation*[32], where the letters indicate respectively:

- A: the distribution of the intertemporal times of arrival *(if arrivals occur according to a Poisson process, than we will find* M*)*;
- B: the distribution of times of service (*if service times have an exponential distribution, than we will find* M);
- c : the number of workers;
- K: the maximum size of the system in number of clients (default: infinite value);
- m : the size of the population (default: infinite value);

[32] Web site of Andrew Ferrier, IT consultant [http://www.andrewferrier.com/oldpages/queueing_theory/Andy/kendall.html]

- Z: the discipline of service (it specifies which will be the next client served among those waiting at the moment as soon as a worker is free; the default is the FIFO[33] discipline).

For a queue system with m workers in parallel, the *Load Coefficient* ρ_c is defined as:

$$\rho_c = \frac{E[t_s]}{E[t_a] \cdot m}$$

Equation 3: Load Coefficient, ρ_c, with the two inter-arrival times

Where $[t_a]$ represents the inter-arrival time between the customers and $[t_s]$ represents the service time – both random variables. Defining the *average frequency of arrivals* λ as:

$$\lambda = 1 / E[t_a]$$

Equation 4: average frequency of arrivals, λ

and the *maximum frequency of the service* μ as:

$$\mu = 1 / E[t_s]$$

Equation 5: maximum frequency of the service, μ

we obtain ρ_c:

$$\rho_c = \frac{\lambda}{\mu \cdot m}$$

Equation 6: load coefficient, ρ_c, with λ and μ

[33] The service disciplines usually considered are: FIFO (first-in first-out), service in order of arrival; LIFO (last-in first-out), service in the reverse order of arrival; SIRO service in random order; class-based service of priority (like, for example, in the First Aid).

For systems in which the customers can never be refused, like in our case, the stability condition is that the *Load Coefficient* must be positive and less than one, that is:

$$0 \leq \rho_c < 1$$

Equation 7: stability condition for ρ_c

When intertemporal times are exponential, the number of events $N(t)$ that occur in a given time t is a Poisson process:

$$P\{N(t) = n\}|_t = e^{-\lambda \cdot t} \cdot \frac{(\lambda \cdot t)^n}{n!}$$

Equation 8: probability mass function of the Poisson distribution

The Poisson process $N(t)$ has the expected value $E\{N(t)\} = \lambda \cdot t$, where λ expresses the average number of events per unit of time, that is the average frequency.

Birth-death processes are stochastic processes representing the number of elements $N(t)$ of a population that may increase, due to a *birth*, or decrease, due to a *death*, of a unit per time. Formally a *birth-death process* assumes that, in any generic instant t, there can only be a single event (birth or death), and also that, given a population of size $N(t) = n$, the time interval until the *next birth* is an exponential random variable with parameter λ_n, while the time interval until the *next death* is an exponential with parameter μ_n.

In this context, the parameters λ_n and μ_n, respectively, can be interpreted as the average rate of birth and death of individuals in a population. The *birth-death process* allows studying the Poisson's queues. In all cases, the arrival of a client can be considered as a *birth* and the completion of a service (thus the abandonment of a client from the system) as a *death*.

In the *Retainer Model* the tasks arrive at the Poisson rate λ and the retainers workers arrive at the rate μ, after they are called. This μ is thought by

the authors as the time it takes a worker to process a task. The system then takes μ to re-fill a space left blank. In this model, the authors assume that λ and μ are known by the requester and that they do not change over time (or, if they change, this happens slowly).

Thus, the *trade-off* is between the probability that an incoming task should rest (the *Expected Waiting Time*) and the Total Cost of having a pool of n workers.

So, to optimize this process and solve this trade-off minimizing the losses, it is necessary to find the unlucky probability that all the crowdworkers are working on the tasks and an incoming task should rest and wait (the so-called *Probability of an Empty Pool*): this is what should not happen in our realtime case.

Knowing this probability, we have to describe the Total Cost of having a pool of n workers, who cost even when they are on stand-by.

If on the one hand we want *the probability of having an Empty Pool* to be the as low as possible (by increasing the number of workers), on the other hand we do not want the number of workers in the pool to become too large because it could cost too much. This trade-off will be studied in the next chapters.

Chapter 4:

The EWT versus Total Cost: the trade-off in the *Retainer Model*

As anticipated above, in this chapter, I will describe the two components of the *Retainers Model's trade-off* expressed formally, but again descriptive, in the paper "*Analytic Methods for Optimizing Real time Crowdsourcing*" (Bernstein & al., 2012).

4.1: The Expected Waiting Time: description of the first part of this trade-off

The *probability of an Empty Pool* (so the fact that a request is not satisfied in real time) is given by the Erlang's B formula.

In 1917, Agner Krarup Erlang[34] found a formula to answer to this problem: how to calculate the probability of a busy signal when a person in a village phoned another person.

He studied the behaviour of the inhabitants of a hypothetical village who used phones to call each other within the confinements of the village, as well as to reach people outside of it. Erlang wanted to calculate how many twisted pairs directed outside might be needed (for the inhabitants) to avoid congestion problems as much as possible.

[34] Agner Krarup Erlang was a pioneer in the studies of telephone traffic. He was born in 1878 in Denmark and he worked for twenty years, until his death in 1929, in the *Copenhagen Telephone Company*. In 1909 he published his first important work, "The theory of probabilities and telephone conversations" earning prestigious awards worldwide and the use of its mathematical models at the British General Post Office. Ten years after his death, in 1940, the Erlang became, in his memory, the unit of measurement of telephone traffic more known in the world of telecommunications.

For first, he supposed that N was the number of telephone twisted pairs directed towards the outside village. For second, that there was an average m of calls done per minute according to the stochastic model of the Poisson process (this was a supposition because he could not know when and for how long the inhabitants had called the outside world).

His ambitious result was to establish the amount of callers who would find the lines busy when trying to call outside the village: he found a formula that answers to his question, which is:

$$B = \frac{\frac{m^N}{N!}}{\sum_{x=0}^{N} \frac{m^x}{x!}}$$

Equation 9: Erlang's original formula of 1917

This equation, nowadays known as *Erlang's B function* is the formula Erlang wrote in 1917. In this formula B is representing the percentage of blocked calls, N the number of telephone channels available and m the estimate volume of *traffic offered.*

So it is possible to calculate $\hat{N}$, the number of telephone channels to be made available, knowing the volume of traffic in a particular period of time, $\hat{m}$ and the percentage of blocked calls that we are willing to accept, $\hat{B}$.

The sizing of the resources of telecommunications systems, for example the number of lines or radio links, is made using the Erlang's B function: the complex mathematics of telephone networks today is still based on these studies. "*Its formula and subsequently his researches are a very important contribution to the telephony and in particular the theories of telephone queuing*" (Carbone)[35].

[35] Web site of the paper [http://www.cirocarbone.it/MyJobs/TrafficTheory.pdf]

The authors, using the concepts of the Queueing Theory and the Erlang's B function, define the *traffic intensity*, ρ, as the percentage of system resources that are used to respond to new tasks incoming:

$$\rho = \frac{\lambda}{\mu}$$

Equation 10: definition of traffic intensity, ρ

They, as said above on page 40, define λ as the *arrival rate of the tasks* and μ as the *arrival rate of the crowdworkers* after they are requested by the system and we can think this μ is the processing time for the tasks.

In $M/M/c/c$ system $\rho < n$ is a necessary condition for the system (as I showed in Equation 7: stability condition for ρ_c): this condition means, by replacing using Equation 10, that the rate of recruitment is higher than the rate of arrival of the task:

$$\rho < n \Leftrightarrow \frac{\lambda}{\mu} < n \Rightarrow \lambda < \mu \cdot n$$

Equation 11: necessary condition in terms of ρ

Only with this hypothesis the system is able to grant an answer to the incoming tasks. After the *shock*, given by the exit of a worker from the pool, the system restores the number only in the case the formula is respected. If, instead, $\rho > n$ then the system will fail to find new workers fast enough to fill the pool.

The probability of an Empty Pool, $G(n)$, is given by the Erlang's B formula, rewriting the Equation 9 with ρ as the *traffic intensity*:

$$G(n) = \frac{\rho^n / n!}{\sum_{i=0}^{n} \rho^i / i!}$$

Equation 12: Erlang's B formula applied to the *Retainer Model*

The probability of an Empty Pool $G(n)$ expresses the probability that the virtual pool containing the crowdworkers waiting for the tasks will be empty (so probability equal 1) or a percentage that can happen within certain values of n and ρ.

If we multiply this probability $G(n)$ per $\frac{1}{\mu}$, representing the waiting time when the pool is empty, we obtain the Expected Waiting Time (EWT):

$$EWT = \frac{1}{\mu} G(n) = \frac{1}{\mu} \cdot \frac{\rho^n / n!}{\sum_{i=0}^{n} \rho^i / i!}$$

Equation 13: Expected Waiting Time formula

The Expected Waiting Time (EWT) is the very first relation between our variables: n (the number of crowdworkers), λ (the *arrival rate of the tasks*), ρ (the traffic intensity, ratio between λ and μ).

Below is the chart of the EWT calculated with $\mu = 2$ and $\rho = 0.5$.

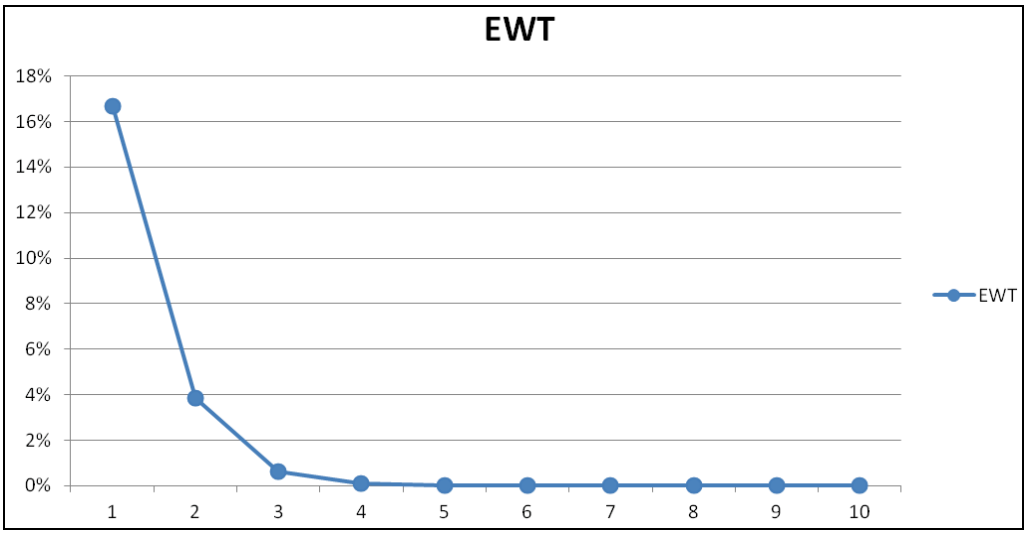

Figure 4: EWT **with** $\mu = 2$ **and** $\rho = 0.5$

From the graph above we can see that the probability that a requester has to face is the 17%, with this value of λ and μ, when in the virtual pool of the system there is only a crowdworker. This probability falls down to 4% when there are two crowdworkers in the pool. We can also see that the $\lim_{n \to +\infty} EWT = 0$.

The EWT is one of the most important relations of the *Retainer Model*. And its component given by the probability of an Empty Pool, $G(n)$, is the first part that has to be better analyzed.

4.2: The Total Cost function C_{tot}: description of the second part of this trade-off

The second part of the trade off of the *Retainer Model* is the cost of having a pool with n crowdworkers in it and, for the authors, it is represented by two main parts.

The first of these parts is the number of employed workers. As we know, we have to pay all the workers in the pool who are waiting for an incoming task, even

those who not engaged in a specific moment. This number is $n - n_{busy}$ (with n_{busy} representing the number of workers active in that specific moment). We can define $E(i)$ as the expected number of busy workers, obtainable from the formula of the probability $G(n)$, as:

$$E(i) = \rho[1 - G(n)]$$

Equation 14: Expected number of busy workers formula

So while the portion of workers who are not waiting does not represent a leak, workers on hold (which still must be paid) represent a cost. The number of this portion of inactive crowdworkers who are still waiting for a task is:

$$n - E(i) \Rightarrow n - \rho[1 - G(n)]$$

Equation 15: number of crowdworkers, in the pool, waiting for a task

If the retainer salary is s (the salary to be paid because a crowdworker previously agree to keep the screen open and waiting for a task, thus staying in this pool), we have to pay, per unit of time:

$$s\{n - \rho[1 - G(n)]\}$$

Equation 16: retainer salary per unit of time formula

The function of Total Cost is also composed by a second part. This second part links the number of workers to a kind of penalty, given by the fact that some tasks cannot be served in real time. This gives a monetary value to the incapacity of the system to serve the requester/s in real time. Because we want to respond in real time, the inability of the system to do so represents a cost, a penalty; this can occur either because of an underestimation of the incoming number of tasks, or because there are not enough inactive crowdworkers in the pool.

So we need to minimize this penalty, this inability, which the authors called the Expected Task Cost (ETC), C_{task}: the cost for a missed task, which result is zero if the system is able to serve all the tasks in real time.

The Total Cost, C_{tot}, putting together the two parts, is then:

$$C_{tot} = C_{task} G(n) + s\{n - \rho[1 - G(n)]\}$$

Equation 17: Total Cost of the Retainer Model equation, C_{tot}

The problem is to find the *optimal value of* n (the number of workers that have to be hired in the pool), n^*, with these parameters:

1. ρ, the *traffic intensity,* calculated using λ and μ;
2. $G(n)$, the probability of an Empty Pool;
3. s, C_{task}, the retainer salary and the ETC.

According to the authors, there are two ways to find this optimum n^*: an *easy way* and a *difficult* (but more truthful) *way*.

The easy one. The first approach (found by the authors) in order to find n^*, is to start by establishing a *maximum accepted probability that an incoming task can wait* $p_{\max}$ not being served in real time. In this case the cost of having n crowdworkers is not minimized, but we are looking for compromise. We are fixing a *maximum accepted probability* and we are calculating, with this, the number of optimal workers, n^*.

We want that the probability of an Empty Pool to be less than this *maximum accepted probability*, $G(n) < p_{\max}$, thinking about $p_{\max}$ as a bound. The authors graphically find n^* looking at the below Figure 5:

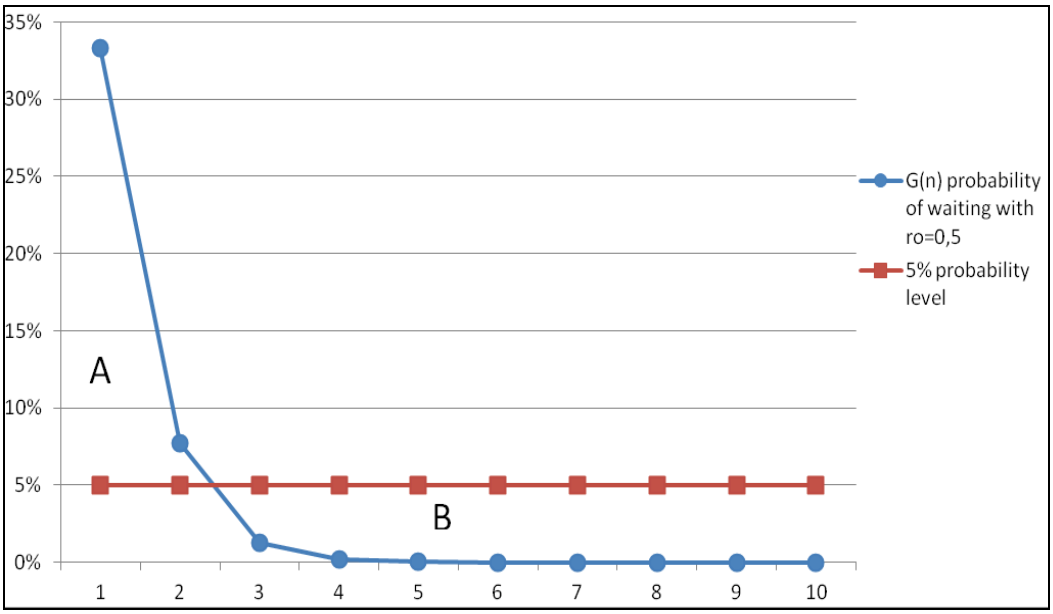

Figure 5: $G(n)$ **with** $\rho = 0.5$ **-vs.-** $p_{\max} = 5\%$**, finding** $n^* = 3$

We can divide Figure 5 in two parts: the part before the intersection (called A) and the part after the intersection (called B). We can see that if $p_{\max} = 5\%$ then $n^* = 3$, because $G(1)$ and $G(2)$ are bigger then $p_{\max} = 5\%$.

Thus, calculating ρ having λ and μ, we can find n^* for every given probability $p_{\max}$. But we are not minimizing the cost, we are arbitrarily fixing a probability without caring whether this amount of probability is good or not. We cannot know if the $p_{\max}$ we have fixed is minimizing the entire cost or not, we do not know if $p_{\max}$ is giving us the best value in the number of workers for given ρ and we are not thinking about the retainer salary s, or about the Expected Task Cost (ETC) C_{task} either.

So, to overcome this inefficiency, *the second and more difficult way* is to think about the entire cost as in Equation 17 and to figure out how it is possible to find n^*, from that equation.

The authors don't say anything about the existence of a minimum, neither they calculate this prophetic number of n^*. They draw the graph of the curve of Total Cost, with different value of λ and μ (so changing ρ), looking at the minimum point in the graph, intersecting this minimum point in the ordinate axis with the *abscissa* axis they find n^*.

We can see this "*algorithm*" in the figure below:

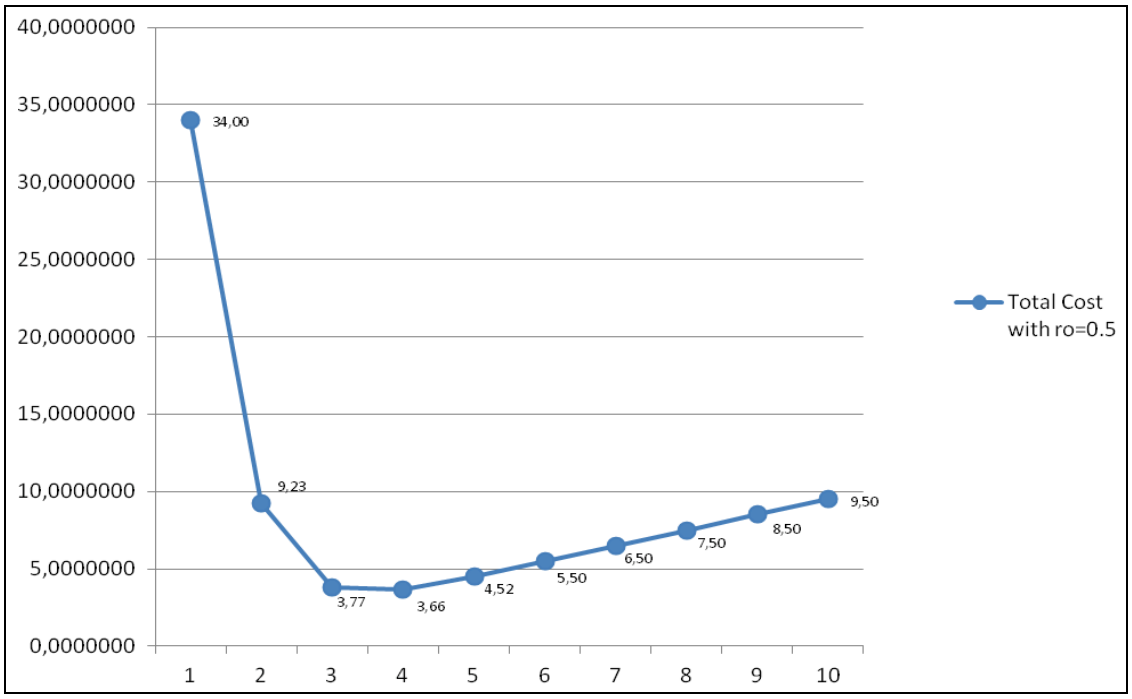

Figure 6: C_{tot} curve with $\rho = 0.5$

In this figure we have plotted the shape of the curve of Total Cost, using Equation 17, with these values of our variables: $\rho = 0.5, s = 1, C_{task} = 100$.

With $\rho = 0.5$, the probability of an Empty Pool $G(n)$ and C_{tot}, becomes:

$$G(n) = \frac{0.5^n / n!}{\sum_{i=0}^{n} 0.5^i / i!}$$

Equation 18: $G(n)$ **with** $\rho = 0.5$

$$C_{tot} = C_{task} G(n) + s\{n - \rho[1 - G(n)]\} \Rightarrow 100 \cdot \frac{0.5^n / n!}{\sum_{i=0}^{n} 0.5^i / i!} + 1\left\{n - 0.5\left[1 - \frac{0.5^n / n!}{\sum_{i=0}^{n} 0.5^i / i!}\right]\right\}$$

Equation 19: C_{tot} **with** $\rho = 0.5$, $s = 1$, $C_{task} = 100$

Given Equation 19, it is easy to find a table that lists values of C_{tot} for the different value of n. This is shown in the table below:

Table 1: values of C_{tot} **with fixed values of our variables, finding** $n^* = 4$

C_{task}	100
s	1
ρ	**0.5**

n	C_{tot}
0	$\rho > n$
1	34.0000000
2	9.2307692
3	3.7721519
4	3.6587678
5	4.5158743

6	5.5013228
7	6.5000945
8	7.5000059
9	8.5000003
10	9.5000000

In this table, representing the different value of C_{tot} with $\rho = 0.5$, for n going from 1 to 10, we can read that the optimal number of crowdworkers in the pool is four, $n^* = 4$, because with $n = 4$ we have the minimum value of C_{tot}.

The authors follow this process using different values of the variables ρ and C_{task} and they show the graphs of these curves.

An important observation follows. As you can see I used the same value of ρ and s for the two ways that the authors used in their paper: the number of workers that come out from the two computations are different: in the first we found $n^* = 3$ and in the second we have $n^* = 4$. The reason is that the two ways are too different and they solve the problem using two different perspectives. The best one between the two is, of course, the second one because it minimizes the function of Total Cost, without fixing any arbitrary value of $G(n)$.

So, to conclude, the minimization process proposed by the authors consists only of a graphic approach: they use a chart in order to find the minimum.

As I wrote above, nothing written by the authors confirms the existence of this minimum (a minimum in a chart is not a mathematical proof) and nothing has been done to find this point in a computational, more efficient and less time-consuming way differing from this "chart process". And these will be the topic of this thesis in the next chapter.

Chapter 5:

Analytical analysis of the trade-off in the *Retainer Model* and the optimized number of workers

The term "*analysis*" derives from the Old Greek term "*αναλύω*" ("*analyo*") that means "decompose, break into different parts". And that is what I will try to do in this chapter, as I said above, in order to *analyze* the different parts of the *Retainer Model* on which the *trade-off* has been defined.

The parts that I will study are:

a. the probability of an Empty Pool $G(n)$;

b. the function of Total Cost C_{tot}.

In this analysis I will use the derivatives of the two components of the trade off I am studying. As we know, the necessary and sufficient conditions, that permits to a function to be differentiable in a specific point, are that the two limits, left and right, of the first derivative for x tending to x_0 are finite and that these two limits coincide with the evaluation of the first derivative at the point x_0. Necessary, but not sufficient, condition for differentiability in a specific point x_0 is that the function is continuous at the point x_0.

In this thesis I will apply the derivatives for functions that admit only positive and integer number, using the derivatives to discover the minimum (*integer*) number of workers in the retainer pool that minimize the Total Cost function. The use of the derivatives will not be used to find this number, but only for the analysis and demonstration of the existence of this minimum.

5.1: The probability of an Empty Pool $G(n)$

In order to study analytically the *probability of an Empty Pool*, $G(n)$, we recall the definition of $G(n)$, which is given by the Equation 12 (the Erlang's B formula) inserting the definition of the *traffic intensity* in it (that is: $\rho = \frac{\lambda}{\mu}$). So, $G(n)$ becomes:

$$G(n) = \frac{\left(\frac{\lambda}{\mu}\right)^n}{n!} \Bigg/ \sum_{i=0}^{n} \left[\frac{\left(\frac{\lambda}{\mu}\right)^i}{i!} \right]$$

Equation 20: $G(n)$ **with** $\rho = \frac{\lambda}{\mu}$

We know that $\rho < n$ is a necessary condition for the system, because otherwise the system is not able to refill the pool with new crowdworkers (as I wrote in Equation 11).

The domain and the limits of $G(n)$.

In order to study this function, the domain of $G(n)$ is the first aspect to analyze. A fraction like $G(n)$ exists if and only if the *denominator* is different from zero. In $G(n)$ the *denominator* of Equation 12, that is $\sum_{i=0}^{n} \frac{\rho^i}{i!}$, is not only different from zero, but always greater than zero.

This because n is always greater than zero – because this number represents the crowdworkers in the virtual pool – and $\rho > 0$ – because λ and μ are always greater than zero.

Second aspects to analyze are the limits of $G(n)$ as n goes to zero or to infinity.

The limit of the probability of an Empty Pool $G(n)$ is one for the number of workers that decrease until zero:

$$\lim_{n\to 0} G(n) = \lim_{n\to 0} \frac{\rho^n / n!}{\sum_{i=0}^{n} \rho^i / i!} = 1$$

Equation 21: limit of $G(n)$ as n goes to zero

This because the numerator becomes $\frac{\rho^0}{0!} = \frac{1}{1}$ and the limit of the denominator of $G(n)$ as n goes to zero goes to 1, being equal to the numerator.

So, we have, as we suspected, the 100% probability of missing a request when there are not any crowdworkers.

The limit of the probability of an Empty Pool, $G(n)$, for the number of workers that goes to infinity is zero:

$$\lim_{n\to +\infty} G(n) = \lim_{n\to +\infty} \frac{\rho^n / n!}{\sum_{i=0}^{n} \rho^i / i!} = 0$$

Equation 22: limit of $G(n)$ as n goes to infinity

The proof of this is that the limit of the numerator of $G(n)$ as n goes to infinity is: $\lim_{n\to +\infty} \rho^n / n! = 0$ and the limit of the denominator of $G(n)$ as n goes to

infinity is: $\lim_{n\to+\infty} \sum_{i=0}^{n} \rho^i / i! = e^{\rho}$. So the ratio between 0 and e^{ρ} is not an *indeterminate form*, but the result is still zero.

Having a pool with a large number of crowdworkers reduces the *probability of an Empty Pool* to zero if the number of crowdworkers goes to infinity.

This is why we want to find the optimal value of n: because this value is closely linked to the values of the probability of missing a request (as they are *directly proportional*) and we want to minimize it, driving down the cost.

The first order derivative of $G(n)$ with respect to n

In order to evaluate the first order condition to optimize $G(n)$, the paper "Computing Erlang–b function derivatives in the number of servers" (Jorge Sá Esteves, 1995)[36] by Sá Esteves, Craveirinha and Cardoso uses a different way to write the Erlang's B function.

Referring to the same notation of this paper, $G(n)$ can be written as:

$$G(n) = \frac{\rho^n / n!}{\sum_{i=0}^{n} \rho^i / i!} = \frac{1}{\rho \cdot \int_0^{+\infty} e^{-\rho \cdot z} \cdot (1+z)^n \, dz}$$

Equation 23: $G(n)$ with the notation of Sá Esteves, 1995.

In this notation the formula is written using the integral operator instead the summation operator.

This notation allows us to describe the first order derivative of the probability of an Empty Pool, with respect to n, which is:

[36] Web site of the paper [http://www.tandfonline.com/doi/abs/10.1080/15326349508807347?journalCode=lstm19#.UhN1Dj9A0_g]

$$G'_n(n) = -\frac{\int_0^{+\infty} e^{-\rho z} \cdot (1+z)^n \cdot \ln(1+z)\,dz}{\rho \cdot \left[\int_0^{+\infty} e^{-\rho z} \cdot (1+z)^n\,dz\right]^2}$$

Equation 24: first order derivative of $G(n)$ with respect to n

The first order derivative of $G(n)$ with respect to the number of workers hired in the virtual pool is negative because it is the ratio of two positive quantities with the minus operator in front. So, $G(n)$ is a strictly decreasing function of n, from 1 to 0, for all ρ.

The second order derivative of $G(n)$ with respect to n

The second order derivative of the probability of an Empty Pool with respect to n is computed in the paper "On a bicriterion server allocation problem in a multidimensional Erlang loss system" (Jorge Sá Esteves, 2013)[37].

Referring to the same notation in it – which still uses the integral operator as in Equation 23 – I can rewrite the second order derivative of $G(n)$ with respect to n, $G''_{nn}(n)$, like:

$$G''_{nn}(n) = -\left[2 \cdot G(n) \cdot G'(n) \cdot \rho \cdot \int_0^{+\infty} e^{-\rho z} \cdot (1+z)^n \cdot \ln(1+z)\,dz\right] +$$
$$- G(n)^2 \cdot \rho \cdot \int_0^{+\infty} e^{-\rho z} \cdot (1+z)^n \cdot [\ln(1+z)]^2\,dz$$

Equation 25: second order derivative of $G(n)$ with respect to n

[37] Web site of the paper [http://www.sciencedirect.com/science/article/pii/S0377042713001209]

Jagers and Van Doorn in their paper "*On the continued Erlang loss function*" (A.A Jagers, 1986)[38] described the convexity of the Erlang's B formula (with respect to n) for n and $\rho \in \Re^{+} \times \Re_{0}^{+}$.

For the purpose of this thesis, because the values of n are positive integers, I can use the paper of Messerli, "*Proof of a Convexity Property of the Erlang B function*"[39] (Messerli, 1972). In this work the author proved that the second derivative of $G(n)$ with respect to n, *integer*, is always positive. So, I can conclude that:

$$G''_{nn}(n) > 0 \quad \forall\ \rho > 0$$

Equation 26: $G''_{nn}(n) > 0$ **for all** n **and** ρ **positive**

The fact that the second derivative of the function with respect to n is positive is important because we can now study the behaviour of the function[40]. The "*second derivative test*" proves if a *critical point* (points where the first derivative is equal to zero) is a *maximum* or a *minimum*. This test states: "suppose that c is a critical point at which $f'(c) = 0$, that $f'(c)$ exists in a neighbourhood of c, and that $f''(c)$ exists. Then f has a relative maximum value at c if $f''(c) < 0$ and a relative minimum value at c if $f''(c) > 0$."[41]

We are in the presence of a minimum point because the sign of the second order derivative is positive and this minimum point is where the first order derivative is equal to zero.

Because the curve is strictly decreasing, the minimum point is at the boundary.

---

[38] Web site of the paper [http://www.sciencedirect.com/science/article/pii/0167637786900994]
[39] Web site of the paper [http://onlinelibrary.wiley.com/doi/10.1002/j.1538-7305.1972.tb01956.x /abstract]
[40] Economics from the Toronto University, Math Tutorial [http://www.economics.utoronto.ca/osborne/ MathTutorial/LONF.HTM]
[41] Web site of the Mathematics Harvey Mudd College, paper: "Calculus: Second Derivative" [http://www.math.hmc.edu/ calculus/tutorials/secondderiv/]

The graphic representation of $G(n)$

We can draw the curves of $G(n)$, for different values of ρ.

Figure below shows the chart of these curves:

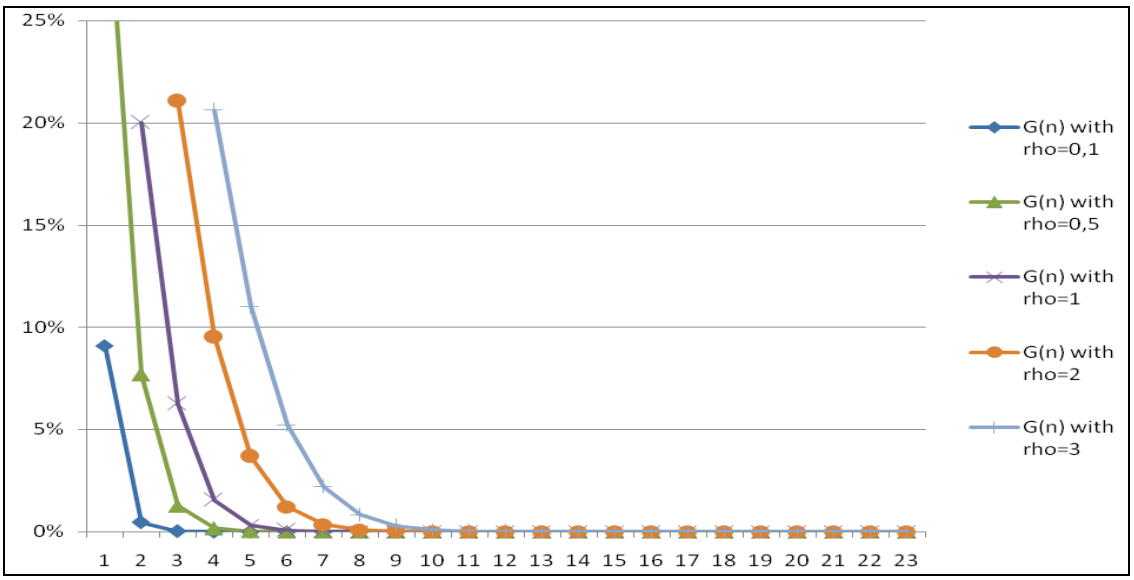

Figure 7: graphs of the values of $G(n)$ for different values of ρ

In the figure above we can see in the *x*-axis the number of workers in the retainer pool (n) and in the *y*-axis the probability of an empty pool, $G(n)$. This probability we can see decreases as n increases. The limit as n goes to infinity is zero and as ρ increases the curve is shifted to the right.

5.2: The function of Total Cost C_{tot}

In order to study analytically a function like the function of the Total Cost, C_{tot} I will analyze the following steps:

a) the domain of the function C_{tot} and the limits of it;

b) the first order derivative for the calculation of the maximum or the minimum of the function;

c) the second order derivative for the study of the concavity of the function;

d) the graphic representation of the function.

I can rewrite the formula of Total Cost C_{tot}, as Equation 17 states, extracting $G(n)$ from the parenthesis and it becomes:

$$C_{tot} = C_{task} G(n) + s\{n - \rho[1 - G(n)]\} = G(n) \cdot (C_{task} + s \cdot \rho) + s \cdot (n - \rho)$$

Equation 27: C_{tot} formula showing the behaviour of $G(n)$

The thesis wants to determine not only the reasons of the *existence of a minimum* point, but also to identify a possible *way to find this optimum number* of workers in the pool, n^*.

As we can easily see in the formula of the Total Cost above, n is present in two parts: it is part of the function $G(n)$ (the *probability of having an Empty Pool*) and it is inside the parenthesis of the second addend.

The domain and the limits of the function of Total Cost, C_{tot}

As we are referring to costs, penalties and number of physical workers, our domain has to be in the positive part of the x-axis and y-axis. As I demonstrated on page 56, $0 < G(n) < 1$, then $C_{task} > 0$, $s > 0$ and $0 < \rho < n$. For all these reasons and because $n > 0$, then $C_{tot} > 0$.

Which are the limits of C_{tot} as n goes to zero or to infinity?

The limit of C_{tot} as n goes to zero is:

$$\begin{aligned}\lim_{n\to 0} C_{tot} &= \lim_{n\to 0}\left(G(n)\cdot\left(C_{task}+s\cdot\rho\right)+s\cdot\left(n-\rho\right)\right)= \\ &= \lim_{n\to 0} G(n)\cdot\left(C_{task}+s\cdot\rho\right)+\lim_{n\to 0} s\cdot\left(n-\rho\right)= \\ &= \left(C_{task}+s\cdot\rho\right)-s\cdot\rho= \\ &= C_{task}\end{aligned}$$

Equation 28: limit of C_{tot} as n goes to zero

As I showed before (on page 56), the limit of the *probability of an Empty Pool* for the number of workers that decrease until zero is: $\lim_{n\to 0} G(n)=1$, then the limit of Total Cost as n goes to zero is a positive value which is C_{task}.

The limit of C_{tot} as n goes to infinity is:

$$\begin{aligned}\lim_{n\to+\infty} C_{tot} &= \lim_{n\to+\infty}\left(G(n)\cdot\left(C_{task}+s\cdot\rho\right)+s\cdot\left(n-\rho\right)\right)= \\ &= \lim_{n\to+\infty} G(n)\cdot\left(C_{task}+s\cdot\rho\right)+\lim_{n\to+\infty} s\cdot\left(n-\rho\right)=0+\infty= \\ &= +\infty\end{aligned}$$

Equation 29: limit of C_{tot} as n goes to infinity

I showed before (on page 56), the limit of the *probability of an Empty Pool* for the number of workers that goes to infinity is: $\lim_{n\to+\infty} G(n)=0$, then the limit of Total Cost as n goes to infinity is plus infinity.

We can also see that when n increases then the first part (the one representing the inability to serve in real time, with the probability given by the Erlang's B function) goes to zero. This means that, as we computed before, the

more the crowdworkers are the less problems I have in serving in real time, because I always have a lot of workers ready in the retainer pool. But this represents a cost for the requester, and this is easily shown by the second part of the limit: as n goes to infinity the second part (representing the cost given by the salary) increases.

This is the dichotomy of the formula of the Total Cost, and shows the reason of the importance of this optimization.

The first order derivative of the function of Total Cost, C_{tot} with respect to n

When we want to find a minimum or a maximum of a function we have to set the first order derivative equal zero. In these points the slope of the curve representing the function is zero and they are called *stationary points*[42].

We will use the formula of the first derivative of $G(n)$, that I wrote in the previous section in Equation 24.

Thus, the first order derivative of C_{tot} is:

$$C'_{n\,tot} = \frac{\partial G(n)}{\partial n} \cdot (C_{task} + s \cdot \rho) + s \cdot \left(\frac{\partial n}{\partial n} - \rho \right) = G'_n(n) \cdot (C_{task} + s \cdot \rho) + s$$

Equation 30: first order derivative of C_{tot} with respect to n, general formula

Then, using the notation with the integrals showed in Equation 24, it becomes:

[42] Definition given by Wolfram Math World [http://mathworld.wolfram.com/StationaryPoint.html]

$$C'_{n\,tot} = G'(n)\cdot(C_{task} + s\cdot\rho) + s = \left\{ -\frac{\int_0^{+\infty} e^{-\rho z}\cdot(1+z)^n\cdot\ln(1+z)\,dz}{\rho\cdot\left[\int_0^{+\infty} e^{-\rho z}\cdot(1+z)^n\,dz\right]^2} \right\}\cdot(C_{task} + s\cdot\rho) + s$$

Equation 31: first order derivative of C_{tot} with respect to n, integral formula of $G(n)$

In order to characterize an interior minimum point, the First Order Condition requires that $f'(x) = 0$ and the Second order Condition requires that $f''(x) > 0$. So in order to respect the FOC, we set $C'_{n\,tot} = 0$ that is:

$$C'_{n\,tot} = 0 \Rightarrow G'(n)\cdot(C_{task} + s\cdot\rho) + s = 0 \Rightarrow G'_n(n) = -\frac{s}{(C_{task} + s\cdot\rho)} \Rightarrow$$

$$\Rightarrow -\frac{\int_0^{+\infty} e^{-\rho z}\cdot(1+z)^n\cdot\ln(1+z)\,dz}{\rho\cdot\left[\int_0^{+\infty} e^{-\rho z}\cdot(1+z)^n\,dz\right]^2} = -\frac{s}{(C_{task} + s\cdot\rho)} \Rightarrow$$

$$\Rightarrow \frac{\int_0^{+\infty} e^{-\rho z}\cdot(1+z)^n\cdot\ln(1+z)\,dz}{\left[\int_0^{+\infty} e^{-\rho z}\cdot(1+z)^n\,dz\right]^2} = \frac{s\cdot\rho}{(C_{task} + s\cdot\rho)}$$

Equation 32: First Order Condition of C_{tot} : $C'_{n\,tot} = 0$

These are the computations to find the *stationary point* of the Total Cost function C_{tot}, mandatory for the optimal number of crowdworkers that minimize the function of Total Cost. In order to say that this stationary point is a minimum, I have to assess that the second order derivative is positive.

The second order derivative of the function of Total Cost, C_{tot} with respect to n

In order to prove that this stationary point is a minimum, I have to demonstrate that the second order derivative of C_{tot} with respect to n is positive, because in this case the concavity is upward[43].

The second order derivative of C_{tot} is:

$$C''_{nn\,tot} = \frac{\partial\left(G'(n)\cdot\left(C_{task} + s\cdot\rho\right) + s\right)}{\partial n} = G''_{nn}(n)\cdot\left(C_{task} + s\cdot\rho\right)$$

Equation 33: second order derivative of C_{tot} with respect to n, general formula

With some manipulations, using the computations of the second order derivative of $G(n)$ made in *Equation 25: second order derivative of $G(n)$ with respect to n*, it becomes:

$$C''_{nn\,tot} = \left\{ \begin{array}{l} -\left[2\cdot G(n)\cdot G'_n(n)\cdot\rho\cdot\int_0^{+\infty} e^{-\rho z}\cdot(1+z)^n\cdot\ln(1+z)\,dz\right] + \\ -G(n)^2\cdot\rho\cdot\int_0^{+\infty} e^{-\rho z}\cdot(1+z)^n\cdot\left[\ln(1+z)\right]^2 dz \end{array} \right\}\cdot\left(C_{task} + s\cdot\rho\right)$$

Equation 34: second order derivative of C_{tot} with respect to n, integral formula of $G(n)$

As I wrote before, $\left(C_{task} + s\cdot\rho\right)$ is always positive because a summation of two positive quantities. Also $G''_{nn}(n)$ is always positive (see *Equation 26: $G''_{nn}(n) > 0$ for all n and ρ positive*).

We can formally conclude that $C''_{nn\,tot} > 0$, always!

So I can now state that: C_{tot} is a strictly convex function, so it has a minimum which is when $C'_{n\,tot} = 0$. This happens when:

[43] This is part of the "Concavity theorem" in "Microeconomic Analysis" (Varian, 1992)

$$\frac{\int_0^{+\infty} e^{-\rho z}\cdot(1+z)^n\cdot\ln(1+z)dz}{\left[\int_0^{+\infty} e^{-\rho z}\cdot(1+z)^n dz\right]^2}=\frac{s\cdot\rho}{(C_{task}+s\cdot\rho)}$$

Equation 35: $C'_{n\,tot}=0$ **equation for the finding of** n^*

The graphic representation of the Total Cost function C_{tot}

The graphic representation of C_{tot}, with different values of ρ is easy to do calculating in a table the different values of C_{tot} for a fixed ρ, with n going from 0 to a limit number.

Equation 19: C_{tot} *with* $\rho=0.5$ and *Table 1: values of* C_{tot} *with fixed values of our variables* are the show of this process, that can be repeated for all the values of ρ. I chose three values of ρ: $\rho=0.5$, $\rho=1$ and $\rho=2.5$. I chose these values in order to have a demonstrative value of ρ: the first less than 1, one equal to 1 and one greater than 1. The chart is:

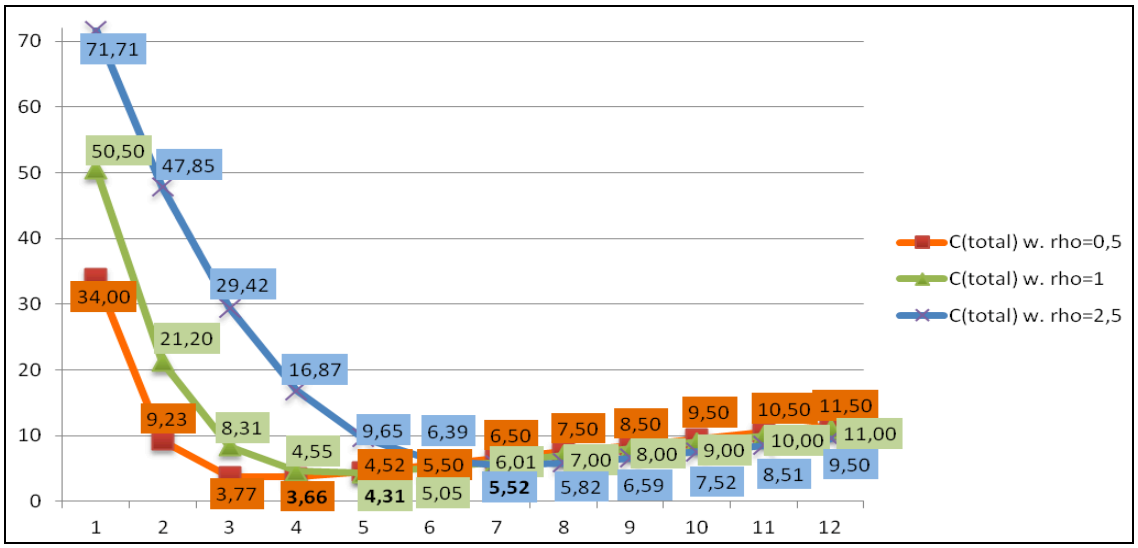

Figure 8: graph of different C_{tot}**, with different values of** ρ

We can see that all the three lines descend in a first region of the graph but after a minimum point they increase (the minimum points are in bold). This fits perfectly with the previous statements!

For $\rho = 0.5$ the stationary point is $n^* = 4$ (this is perfectly equal to the demonstrative case of the authors as in Table 1); for $\rho = 1$ the stationary point is $n^* = 5$; for $\rho = 2.5$ the stationary point is $n^* = 7$.

The question is: how to find the optimum number of workers, n^*, such that $C'_{n\,tot} = 0$, in an easier, faster and more accurate way that differs from doing a chart every time?

There could be several methods for solving Equation 35: Newton's method[44], Fibonacci's Method[45] or could be fascinating to build an efficient automatic algorithm.

5.3: The automatic algorithm

As we are talking about an efficient use of people, machines, computers, it could be a good point to have an algorithm able to show in few milliseconds the right answer in the number of workers. Of course this number, n^*, depends on some parameters that, in this model, are exogenous.

[44] Newton's method from the Massachusetts Institute of Technology [http://www2.myoops.org/course_material/mit/NR/rdonlyres/Sloan-School-of-Management/15-084JSpring2004/CCAD5994-3D8E-49C9-83BE-EE4B0A6721C8/0/lec3_newton_mthd.pdf]

[45] Fibonacci Search from the National Institute of Standards and Technology [http://xlinux.nist.gov/dads/HTML/fibonaccianSearch.html]

A list of algorithms built in order to optimize some problems is presented in the book of Professor Kelley, "Iterative methods for optimization" (Kelley, 1987)[46] in which he describes different algorithms for different functions.

Some of those have the same characteristics of the Total Cost function: the function is twice continuously differentiable and the minimum point is where the first order derivative is equal zero and the second derivative positive.

The algorithms described by Professor Kelley are written in MATLAB code[47] that is a "high-level language and interactive environment for numerical computation, visualization, and programming. Using MATLAB, you can analyze data, develop algorithms, and create models and applications."[48].

In order to obtain the optimal number of workers, n^*, the algorithm, written in MATLAB code, has to find the minimum value of *Equation 17: Total Cost of the Retainer Model equation,* C_{tot}, for given values of ρ, C_{task} and s. We also know that this minimum exists, because I proved it before in *5.2: The function of Total* Cost C_{tot}.

The way to find a cyclical process in order to build the algorithm

The algorithm has to calculate, using a cyclical approach, the amount of C_{tot} that the requester has to suffer when hiring one person, then it has to calculate the amount of C_{tot} suffered when hiring two people, then it has to calculate the amount of C_{tot} when three people are hired and so on until an established limit of workers.

[46] Web site of the book provides by Computational and Applied Mathematics of the Rice University, Houston, Texas [http://www.caam.rice.edu/~zhang/caam554/KelleyBooks/fr18_book.pdf]
[47] Web site with all the MATLAB code presented by Kelley in its book [http://www.siam.org/books/kelley/fr18/matlabcode.php]
[48] Web site of MathWorks, the developer of MATLAB [http://www.mathworks.it/it/help/matlab/learn_matlab/product-description.html]

The program has to stop when the amount of C_{tot} that the requester has to suffer hiring $\hat{n}$ people is bigger than the one the requester has to suffer hiring $\hat{n}-1$ people. So $\hat{n}-1$ is the optimum number of workers, the value that minimize the Total Cost function.

Once the requester has put the values of ρ, C_{task}, the algorithm has to start the process: calculating the value of $G(n)$ starting from 1 to $\hat{n}$ and put this value inside the formula of C_{tot} every time to see whether the previous result (with $\hat{n}-1$ people) was slower or not.

In order to write this algorithm to calculate the optimal value of workers to be paid in the pool, it is necessary to link the sequential values of $G(n)$.

How can we link two consecutive values of $G(n)$? Is it possible to write $G(n)$ using $G(n-1)$? The answer and its proof follow.

First I define $I(n)$ the inverse of $G(n)$:

$$I(n)=\frac{1}{G(n)}$$

Equation 36: $I(n)$, the inverse of $G(n)$

We know that $G(n)$ is the probability of having an Empty Pool and its formula is the Erlang's B formula (see *Equation 12: Erlang's B formula applied to the Retainer Model*). I can write these equations (*Equation 12: Erlang's B formula applied to the Retainer Model* and *Equation 36: $I(n)$, the inverse of $G(n)$*) rewriting the numerator and the denominator in a different way obtaining:

$$G(n)=\frac{\rho^n}{n!\cdot\sum_{i=0}^{n}\rho^i/i!}\Rightarrow I(n)=\frac{1}{G(n)}=\frac{n!\cdot\sum_{i=0}^{n}\rho^i/i!}{\rho^n}$$

Equation 37: $I(n)$ formal manner definition

In order to be able to write a cyclical process for the algorithm, the point is to find a formula that combines $G(n)$ and $G(n-1)$.

Using the same formula that describes $G(n)$, I can write the formula for $G(n-1)$ and the inverse of this equation, $I(n-1)$, that are:

$$G(n-1)=\frac{\rho^{n-1}}{(n-1)!\cdot\sum_{i=0}^{n-1}\rho^i/i!}\Rightarrow I(n-1)=\frac{1}{G(n-1)}=\frac{(n-1)!\cdot\sum_{i=0}^{n-1}\rho^i/i!}{\rho^{n-1}}$$

Equation 38: $G(n-1)$ and $I(n-1)$ formal definition

I now have the two values that I want to join, $I(n)$ and $I(n-1)$ (it is exactly the same of linking $G(n)$ and $G(n-1)$).

$I(n)$ can be written as:

$$I(n)=\frac{n!\cdot\sum_{i=0}^{n}\rho^i/i!}{\rho^n}=\frac{n!\cdot\left(\sum_{i=0}^{n-1}\rho^i/i!+\frac{\rho^n}{n!}\right)}{\rho^n}$$

Equation 39: first passage finding the linking formula between $G(n)$ and $G(n-1)$

I used one of the so called "*general manipulations*[49]" of the summation operator and I separated the last term of the summation from the other $n-1$ previous values in order to obtain:

$$\sum_{i=0}^{n} \rho^i / i! = \sum_{i=0}^{n-1} \rho^i / i! + \frac{\rho^n}{n!}$$

Equation 40: summation operator property used in the proof

Having divided the summation operator we can proceed in finding a link between $I(n)$ and $I(n-1)$. We can multiply $n!$ for the two addends and we can divide the formula into two parts, knowing that $n > 0$ and $\rho^n > 0$, and we obtain:

$$I(n) = \frac{n! \cdot \sum_{i=0}^{n-1} \rho^i / i! + n! \cdot \frac{\rho^n}{n!}}{\rho^n} = \frac{n! \cdot \sum_{i=0}^{n-1} \rho^i / i!}{\rho^n} + \frac{\not{n}! \cdot \frac{\not{\rho}^n}{\not{n}!}}{\not{\rho}^n} = \frac{n! \cdot \sum_{i=0}^{n-1} \rho^i / i!}{\rho^n} + 1$$

Equation 41: second passage finding the linking formula between $G(n)$ and $G(n-1)$

I can now simplify the first addend, using the property of the "*Consecutive neighbour's identities*"[50] of the factorial which says that $n! = n \cdot (n-1)!$ and the "*Product of powers property*"[51] which states that $x^m \times x^u = x^{m+u}$ for all x and m, u integers (like our case), to obtain:

$$I(n) = \frac{n! \cdot \sum_{i=0}^{n-1} \rho^i / i!}{\rho^n} + 1 = \frac{n \cdot (n-1)! \cdot \sum_{i=0}^{n-1} \rho^i / i!}{\rho \cdot \rho^{n-1}} + 1$$

Equation 42: third passage finding the linking formula between $G(n)$ and $G(n-1)$

---

[49] The term used in Wikipedia [http://en.wikipedia.org/wiki/Summation#General_manipulations].
I used $\sum_{n=s}^{r} f(n) = \sum_{n=s-1}^{r-1} f(n+1)$ manipulation.

[50] Property of the Consecutive neighbors
[http://functions.wolfram.com/GammaBetaErf/Factorial/17/01/01/]

[51] Product of poker property [http://hotmath.com/hotmath_help/topics/properties-of-exponents.html]

As I showed in *Equation 38:* $G(n-1)$ *and* $I(n-1)$ *formal definition*:

$$I(n-1)=\frac{1}{G(n-1)}=\frac{(n-1)!\cdot\sum_{i=0}^{n-1}\rho^{i}/i!}{\rho^{n-1}}$$

Equation 43 = Equation 38: $G(n-1)$ **and** $I(n-1)$ **formal definition**

As it easy to see, we can find this ratio in the equation above of $I(n)$ (*Equation 42*) and I am able to show how is possible to join $I(n)$ and $I(n-1)$, that is:

$$I(n)=\frac{n\cdot(n-1)!\cdot\sum_{i=0}^{n-1}\rho^{i}/i!}{\rho\cdot\rho^{n-1}}+1=\frac{n\cdot I(n-1)}{\rho}+1$$

Equation 44: proof of the linking formula between $I(n)$ **and** $I(n-1)$

Equation 44 is the equation with which we can join $I(n)$ and $I(n-1)$ and so, starting from this, I can also prove how to join $G(n)$ and $G(n-1)$.

Having defined $I(n)$ as the inverse of $G(n)$, I can rewrite the *Equation 44,* using $G(n)$ instead of $I(n)$, which is the formula that links the sequence of $G(n)$ and it is:

$$G(n)=\frac{\rho}{\frac{n}{G(n-1)}+\rho}=\frac{G(n-1)\cdot\rho}{n+G(n-1)\cdot\rho}$$

Equation 45: proof of the linking formula between $G(n)$ **and** $G(n-1)$

In order to see if these computations are correct, I made some tables (presented in the Appendix of the thesis) that show the values of $G(n)$ using the Erlang's B formula (*Equation 12*) and the values of $G(n)$ using *Equation 44* and *Equation 45.*

As we can see, the values of $G(n)$ calculated with the two different approaches are equal. So I can proceed with the algorithm using $I(n)$ as recursive factor:

$$I(n) = \frac{n \cdot I(n-1)}{\rho} + 1 \Leftrightarrow G(n) = \frac{G(n-1) \cdot \rho}{n + G(n-1) \cdot \rho}$$

Equation 46: $I(n)$ and $G(n)$ as recursive factors for the algorithm

The explanation of the algorithm

The structure of the MATLAB code of the algorithm[52] is reported in the appendix. What it is important is to understand what the algorithm produces and how. We ask the algorithm to produce the result of the minimum Total Cost function C_{tot}, with its relative optimized number of workers n^*.

The idea is that the user inserts the value of ρ, C_{task}, s (fixing them in the algorithm) in the first line and leaves the algorithm to calculate the starting value of $I(1)$. This value, depending on the value of ρ, is equal to:

$$I(1) = \frac{1 \cdot I(0)}{\rho} + 1 = \frac{1}{\rho} + 1$$

Equation 47: $I(n)$ formula with $n = 1$

[52] I want to thank professor Jorge Sá Esteves for his priceless help in the building of the algorithm. He wrote me that 20 years ago Dr. Jagerman helped him sending some material and I am grateful of his contribution.

The algorithm, having calculated $I(1)$, can easily determine the value of $G(1)$ (its reciprocal) and, starting from this value obtained with ρ, the algorithm can establish, using C_{task} and s, the value for the Total Cost formula, C_{tot} (which is *Equation 17*). This is the first value with which the algorithm starts its comparison between the following amount of Total Cost.

In fact, the algorithm has to start a recursive and cyclical process. Starting from n equal 2, it has to compute the value for C_{tot} and compare it whit the previous value of C_{tot}. When it finds that this value for $n = \hat{n}$ is bigger than the value obtained with $\hat{n} - 1$, it has to stop the process because it has found the minimum C_{tot} and so, it can show that n^* is equal to $\hat{n} - 1$.

In the appendix under the structure of the algorithm we can find how the MATLAB code of this cyclical process of the algorithm works.

The last line of the algorithm shows the two values of my search: the value of n^* and the value of C_{tot} minimum calculated using n^* in the Equation 17, that is:

$$C_{tot\,MIN} = C_{task} G(n^*) + s\{n^* - \rho[1 - G(n^*)]\}$$

Equation 48: last line of the algorithm showing $C_{tot\,MIN}$ using n^* calculated automatically

Examples of the results done by the algorithm

I considered three applications setting the values of the variables ρ, C_{task}, s and comparing the results given by the algorithm with the result obtained with the author's method.

Example 1. As first example, I set $\rho = 0.5$, $C_{task} = 100$ and $s = 1$ and, as we can see in Figure 10 in the appendix, the algorithm finds $n^* = 4$, and this value is

equal to the value of n^* we can check in *Table 4: values of* C_{tot} *with* $s = 1$, $C_{task} = 100$ *and* $0.1 \leq \rho \leq 1$. This value is also equal to the one found in *Figure 6:* C_{tot} *curve with* $\rho = 0.5$ and *Table 1: values of* C_{tot} *with fixed values of our variables*, using the same method of the authors. If we see in *Figure 8: graph of different* C_{tot}, *with different values of* ρ we can check that this value is correct.

Example 2. As second example, I set $\rho = 1.0$, $C_{task} = 100$ and $s = 1$ and, as we can see in Figure 11 in the appendix, the algorithm finds that $n^* = 5$ and that is equal to the n^* we can check in Table 4. Also this value is equal to the minimum point I found in *Figure 8: graph of different* C_{tot}, *with different values of* ρ

Example 3. As third and last example, setting $\rho = 2.5$, $C_{task} = 100$ and $s = 1$ we can see in Figure 12 that the algorithm runs perfectly because it finds that $n^* = 7$ and this value is equal to the n^* we can check in Table 6. This value is, also, equal to the minimum point I found in Figure 8: graph of different C_{tot}, with different values of ρ.

We can conclude that the algorithm is a useful tool in order to obtain this optimal number of workers without losing a lot of time in drafting a chart every time we need to calculate the minimum point.

After having described the process to obtain the minimum of the Total Cost function I want to focus on the behaviour of ρ.

5.4: Behaviour of ρ in n^*: the mixed derivative in the Stationary Point

In the appendix I attached the tables for the different values of ρ, given $C_{task} = 100$ and $s = 1$. I computed those tables using the approach used, for example, in Equation 19 and used by the authors to draw the charts in order to find the minimum.

I built the tables in this way: in the first column the different values of n (from 0 up to 23) and for each n, fixing the value of $C_{task} = 100$ and $s = 1$; I calculated the value of C_{tot}, substituting the value of $\hat{\rho}$ with a numerical value from 0.1 to 10 (with an increment of 0.1) in order to build 100 different columns for 100 different values of C_{tot}, using the formula:

$$C_{tot} = C_{task} G(n_i) + s\{n_i - \rho[1 - G(n_i)]\} \Rightarrow 100 \cdot \frac{\hat{\rho}^{n_i} / n_i!}{\sum_{i=0}^{n_i} \hat{\rho}^i / i!} + 1 \left\{ n_i - \hat{\rho} \left[1 - \frac{\hat{\rho}^{n_i} / n_i!}{\sum_{i=0}^{n_i} \hat{\rho}^i / i!} \right] \right\}$$

Equation 49: formula for the computation of the tables in appendix

We can see, as *the traffic intensity* ρ, increases the minimum number of workers that the system requires, n^*, increases too.

How to prove the behaviour of n^* when ρ changes? I know that, for the First order Condition, when $C'_{n\,tot} = 0$ I can find n^*. I also know that n^* depends on ρ, C_{task} and s. If I define $C'_{n\,tot} = F$ I can compute the derivative of F with respect to ρ. The answer will be the behaviour of n^* as ρ increases, and we saw, from the tables in the appendix, that the values of n^* have to increase. This means that I have to find that $F'_{\rho} > 0$

In order to demonstrate that $F'_\rho > 0$, I have to show that:

$$F'_\rho = \frac{\partial n^*}{\partial \rho} > 0$$

Equation 50: first order derivative of $F = C'_{n\,tot}$ with respect to ρ

In order to compute the first derivative of F $(= C'_{n\,tot})$ with respect to ρ, I use the "*Implicit function theorem*". This theorem affirms that "*we can figure out* $\frac{dy}{dx}$ *quite easily [...] this derivative is:*

$$\frac{dy}{dx} = -\frac{\partial g / \partial x}{\partial g / \partial y}$$ "[53].

Equation 51: Implicit Function Theorem

So, applying this theorem, we can compute the first derivative of F $(= C'_{n\,tot})$ with respect to ρ following Equation 50, obtaining:

$$F'_\rho = \frac{\partial n^*}{\partial \rho} = -\frac{\frac{\partial F}{\partial \rho}}{\frac{\partial F}{\partial n}}$$

Equation 52: application of the Implicit Function Theorem for the first order derivative of $C'_{n\,tot} = F$ with respect to ρ

[53] Web site of theFall University of North Carolina at Chapel Hill, Math for Economic theory 2001, [http://www.unc.edu/~swlt/fall2.pdf]

In order to obtain the result that I expected ($F'_{\rho} > 0$) I need to say that the numerator and the denominator have opposite signs so the total ratio is negative and with the negative sign in front of it, it becomes positive.

The numerator of the ratio, knowing that $C'_{n\,tot} = G'(n)\cdot(C_{task} + s\cdot\rho) + s$, is:

$$\frac{\partial C'_{n\,tot}}{\partial \rho} = \frac{\partial\left(G'(n)\cdot(C_{task} + s\cdot\rho) + s\right)}{\partial \rho}$$

Equation 53: general form of the numerator of Equation 52

In order to find the sign of the numerator I can divide it in two parts and analyze the relative signs:

$$\begin{aligned}\frac{\partial\left(G'(n)\cdot(C_{task} + s\cdot\rho) + s\right)}{\partial \rho} &= \frac{\partial\left(G'(n)\cdot(C_{task} + s\cdot\rho)\right)}{\partial \rho} + \frac{\partial s}{\partial \rho} = \\ &= \frac{\partial\left(G'(n)\cdot C_{task}\right)}{\partial \rho} + \frac{\partial\left(G'(n)\cdot s\cdot\rho\right)}{\partial \rho} = \\ &= C_{task}\cdot G''_{n\rho}(n) + G''_{n\rho}(n)\cdot s\cdot\rho + G'_{n}(n)\cdot s = \\ &= G''_{n\rho}(n)\cdot(C_{task} + s\cdot\rho) + G'_{n}(n)\cdot s\end{aligned}$$

Equation 54: final equation of the numerator of Equation 52

The sign of the second addend – $G'_{n}(n)\cdot s$ – is negative, it was checked before in the analysis of $G(n)$ on page 58, as Equation 24 affirms.

The first addend, $G''_{n\rho}(n)\cdot(C_{task} + s\cdot\rho)$, has to be studied: I know that the parenthesis $(C_{task} + s\cdot\rho)$ is positive because it is a summation of positive quantities.

I have to study the sign of $G''_{n\rho}(n)$ and say if it is positive or negative.

The paper "*Second order Conditions on the Overflow Traffic from the Erlang-B System*" (J. S. Esteves, 2009)[54] answers to this problem. In this paper it is proved that:

$$\hat{A}''_{ax}(a,x) = \hat{A}''_{xa}(a,x) < 0$$

$$\text{with } \hat{A}(a,x) = a \cdot B(a,x) \vee B(a,x) = \frac{a^x / x!}{\sum_{j=0}^{x} a^j / j!}$$

Equation 55: Theorem found in the paper "Second order Conditions on the Overflow Traffic from the Erlang-B System"

So, using this theorem, I can say that $G''_{n\rho}(n)$ is negative.

The numerator of the Equation 52 is negative. This because it is the summation of two negative quantities: $G''_{n\rho}(n)\cdot(C_{task} + s\cdot\rho)$ and $G'_n(n)\cdot s$:

$$\left[G''_{n\rho}(n)\cdot(C_{task} + s\cdot\rho) + G'_n(n)\cdot s\right] < 0 \Rightarrow \frac{\partial C'_{n\,tot}}{\partial\rho} = \frac{\partial F}{\partial\rho} < 0$$

Equation 56: proof of the negativity of the numerator of Equation 52

I have to analyze the denominator of the Equation 52. It is the well studied second order derivative of C_{tot} with respect to n (*Equation 26:* $G''_{nn}(n) > 0$ *for all* n *and* ρ *positive*) and, as I said above, it is positive.

So, having calculated the sign of the numerator and the sign of the denominator, I can find the sign of the entire Equation 52. As we wanted, the sign is positive.

[54] Web site for the paper [http://link.springer.com/article/10.1007%2Fs10958-009-9605-x]

The mathematical proof of the fact that ρ increases as n^* increases is this: as states above the denominator is positive $\left(\frac{\partial C'_{n\,tot}}{\partial n} > 0\right)$ and the numerator is negative $\left(\frac{\partial C'_{n\,tot}}{\partial \rho} < 0\right)$, but having the minus in front it becomes positive. So, the entire ratio is positive!

$$F'_{\rho} = \frac{\partial n^*}{\partial \rho} = -\frac{\frac{\partial F}{\partial \rho}}{\frac{\partial F}{\partial n}} = \frac{-\frac{\partial F}{\partial \rho}}{\frac{\partial F}{\partial n}} = \frac{-\frac{\partial C'_{n\,tot}}{\partial \rho}}{\frac{\partial C'_{n\,tot}}{\partial n}} > 0$$

Equation 57: proof of the positivity of the first order derivative of $C'_{n\,tot} = F$ with respect to ρ

We can formally conclude that as *the traffic intensity* ρ increases the minimum number of workers that the system requires, n^*, increases too.

Chapter 6:
Conclusions

In this last chapter, I want to summarize the main results and the most important passages of this thesis.

In the first chapter, we saw what crowdsourcing is and how important it is becoming. This is because, nowadays, the world is becoming more linked, interrelated and without temporal or spatial limits and boundaries.

As we can see, in everyday life, relations are becoming more woven and faster. I think that this *speed* is a gain and we should not be afraid of a possible loss of the *real* relationships. This is because relations cannot be true or real if the mere speed scratches them.

Crowdsourcing is important because it increases collaboration among networks, establishing new limits breaking the common boundaries.

Outsourcing to unknown workers some parts of a project or same tasks has been a brilliant idea, from which idea the future of interrelation could be built. Crowdsourcing, combining crowds of crowdworkers, guarantees the resolution of tasks obtained from a lot of different perspectives. Adding small portions combined in a unique work is the core part of this theme and represents the beauty and the peculiarity of crowdsourcing.

Crowdsourcing represents the new frontier in the relationship between the protagonists of the supply chain because inventions will be generated, more and more, from the bottom.

Pierre Lévy said: “No one knows everything, everyone knows something, all knowledge resides in humanity” and, in my opinion, it is possible to exploit this collective intelligence by linking and valorizing these new frontiers as crowdsourcing is.

An application that will always be very important in the next future is the *realtime* crowdsourcing. In this fast environment where we live avoiding time consuming processes has become the most frequent hobby for everyone: reasoning in these terms, it possible to apply the logic of the real time in the use of crowdsourcing. What we want is a kind of crowdsourcing that gives us a response in a few seconds.

As I wrote in chapter two, this application is not well studied because it is recent and only prototyped. Papers and experiments show the advantages of this theme. As described above, with the application of the *realtime crowdsourcing* we can, for example, overcome disabilities or build new fast virtual pools for multi decisions agencies.

Given the importance of this, I decided to study in depth and analyze a mathematical model called the *Retainer Model*. This model is the first (*and the only one as far as I am aware off*) that tries to develop a series of equations to explain crowdsourcing in real time. The model explains how to obtain answers for incoming tasks from unknown crowdworkers in real time. It grants this building a virtual pool of workers who are always ready to answer a task and who are paid to maintain their browser window open while lying in wait. This method is very different from the one utilised in actual crowdsourcing websites where priority of tasks is granted trough payment (paying the website to maintain the task in the first position of the list of all the tasks).

In the *Retainer Model* when a task that has to be solved arrives, a crowdworker exits from the retainer pool and an another worker is searched and

hired. The arrival of the tasks and also the crowdworkers refilling process can be considered a *Poisson* process.

So, using the *Queueing Theory*, the *Retainer Model* explains how is possible to calculate the probability of being able to answer to a task in *real time* with a certain amount of crowdworkers. The *model*, through the Erlang's B function, determines this probability by using the number of workers (and its re-filling time rate) and the incoming rate of the tasks. Logically, having an high number of crowdworkers reduces this probability to zero.

But, because the crowdworkers in the virtual pool cost even when inactive we have to minimize the number of idle workers. The *Retainer Model* builds the Total Cost function which consists in the cost of having hired n workers and the cost of missing a request in *real time*, which is thought as a penalty.

I did a deep and detailed analysis of the two parts of this trade-off: the *probability of an Empty Pool*, that main part of the *Expected Waiting Time* formula – which gives the expected time that a requester has to wait with a certain amount of crowdworkers – and the *Total Cost* formula of the model.

I found the reasons of the minimization and of the existence of a minimum point. These answers give us the certainty to say *why* and *how* is possible to minimize the Total Cost function. Because the Second Order derivative of the function of Total Cost is positive, this allows us to say that the minimum point exists and that is possible to find it by putting the First Order derivative of the function of Total Cost equal zero.

This value is extremely hard (*I'm out of my depth here*) to find analytically, so while the authors of the model used a graphic approach, I used a *new algorithm* to find it. The graphic approach is not precise and it is very time consuming so, with a recursive approach, this minimum point that could be used to achieve a response with no waiting time it is easy to find. The algorithm is based on a

recursive approach which uses the value of $G(n-1)$ to find $G(n)$. Doing so the algorithm finds the number of workers which grants the smallest value of C_{tot}.

After this *new algorithm* and its explanation, I found the reasons *why* the optimal number of crowdworkers, n^* adds up, increasing the *traffic intensity*. This could be logically understood: as the percentage of the resources of the system answering the incoming tasks increases, the number of crowdworkers to be hired in the pool has to increase, too. But words are words and in math we need proof to give certainty!

Next steps and future works could be the analysis of the realtime crowdsourcing applying these studies and this model to a real website application and with models using real data sets built for this purpose.

With this real application it could be important to understand a part of the Total Cost function which is, in my opinion, still mysterious: the "*cost for a missed task,* C_{task}". What could matter would be to study and to understand its correct amount or its plausible range.

Another possible future work is the minimization of a new Total Cost function containing all the n workers and their costs instead of calculating the loss function as the authors did. We can rewrite the Total Cost function, C_{tot}, in this way:

$$C_{tot\ NEW} = C_{task} + \rho\left[1 - G(n)\right] \cdot r + s \cdot n$$

Equation 58: new Total Cost function for possible future works

In this formula I added the variable r to represent the amount of money paid by the requester to those crowdworkers who complete the tasks given. In this new formula I calculated the entire cost suffered to hire all the n workers

who stay in the retainer pool and the number of active workers who answer the incoming tasks. This is different from the classical formula of Total Cost used in the retainer model (which calculates the amount of *nonworking* crowdworkers only) and it could be useful for further studies.

How to modify the relation between the *Retainer Model* it and the *Queueing Theory* could be a next potential study. It could be fascinating to use other formulas, different from the Erlang's B.

All this analysis has been for me an incredible work experience and a great possibility to demonstrate my abilities in *optimization*. I had the chance to connect the studies in different fields and to apply them in a new theme such as *realtime crowdsourcing*.

Acknowledgements/Ringraziamenti

First of all thanks to Professor Paola Ferretti, for her trust and for giving me the chance to work on this thesis, for having followed carefully and patiently all the phases of this work and for letting me to find, *independently*, the most suited way to my abilities.

Thanks to Professors Andrea Ellero and Jorge Sá Esteves for their immense generosity and for helping me to enrich this work with many precious ideas, contributions and suggestions.

Thanks to all my professors and teachers from the beginning to now through all these 21 years of education, for the important contribution in my life, as a student, but, more importantly, as a person.

Thanks to my Father and my Mother, for their endless lessons in everyday life and for the efforts they made so that I could get here today.

Thanks to my brother Pietro, because he deserves to be mentioned for the beautiful person he is.

Thanks to Chiara for always believing in me, for giving me patient and consistent support every day, for the determination, the nice stubbornness and the loving effort that distinguishes her in the challenges we faced and those that we will face. Together.

Thanks to my classmates and my friends, for making the route less arduous and less steep, through the affections and the warmth of friendship.
A special thanks to the working group of the *Presidio di Libera San Donà*, an association founded by Don Ciotti and supported throughout Italy by hundreds of committed greenhorns. To my nine fellows of the *Presidio,* Chiara, Ilaria, Ilaria, Elsa, Claudia, Sindi, Silvia, Niccolò and Alberto, wish you the strength to be able to change this Country.

Thanks to all those who, one way or another, have stood by my side so that I could be here today and have made me the person I am.

Thanks to myself, for having always believed in the possibilities of the world and never given up, in spite of everything and not without difficulty: because, as A.E. Roosvelt, said,

"*the future belongs to those who believe in the beauty of their dreams*".

Appendix:

These first two tables show that the values of $G(n)$ (obtained by the two methods presented on 73) find exactly the same values. So, we can use the second method in the algorithm structure:

Table 2: values of $G(n)$ using Erlang's B formula

ρ =	***0,5***		
n	numerator	Denominator	$G(n)$=numerator/ denominator
0	1.00000000000000000000	1.00000000000000000000	1.00000000000000000000
1	0.50000000000000000000	1.50000000000000000000	0.33333333333333300000
2	0.12500000000000000000	1.62500000000000000000	0.07692307692307690000
3	0.02083333333333330000	1.64583333333333000000	0.01265822784810130000
4	0.00260416666666667000	1.64843750000000000000	0.00157977883096367000
5	0.00026041666666666700	1.64869791666667000000	0.00015795293002685200
6	0.00002170138888888890	1.64871961805556000000	0.00001316257091335080
7	0.00000155009920634921	1.64872116815476000000	0.00000094018275272347
8	0.00000009688120039683	1.64872126503596000000	0.00000005876141859231
9	0.00000000538228891093	1.64872127041825000000	0.00000000326452324447
10	0.00000000026911444555	1.64872127068737000000	0.00000000016322616220
11	0.00000000001223247480	1.64872127069960000000	0.00000000000741937101
12	0.00000000000050968645	1.64872127070011000000	0.00000000000030914046
13	0.00000000000001960332	1.64872127070013000000	0.00000000000001189002
14	0.00000000000000070012	1.64872127070013000000	0.00000000000000042464
15	0.00000000000000002334	1.64872127070013000000	0.00000000000000001415
16	0.00000000000000000073	1.64872127070013000000	0.00000000000000000044
17	0.00000000000000000002	1.64872127070013000000	0.00000000000000000001
18	0.00000000000000000000	1.64872127070013000000	0.00000000000000000000
19	0.00000000000000000000	1.64872127070013000000	0.00000000000000000000
20	0.00000000000000000000	1.64872127070013000000	0.00000000000000000000

Table 3: values of $G(n)$ using the recursive approach $G(n)=\frac{1}{I(n)}$

ρ	***0,5***	
n	$I(n)=\frac{n\cdot I(n-1)}{\rho}+1$	$G(n)=\frac{1}{I(n)}$
0	1.0	1.00000000000000000000
1	3.0	0.33333333333333000000
2	13.0	0.07692307692307690000
3	79.0	0.01265822784810130000
4	633.0	0.00157977883096367000
5	6,331.0	0.00015795293002685200
6	75,973.0	0.00001316257091335080
7	1,063,623.0	0.00000094018275272347
8	17,017,969.0	0.00000005876141859231
9	306,323,443.0	0.00000000326452324447
10	6,126,468,861.0	0.00000000016322616220
11	134,782,314,943.0	0.00000000000741937101
12	3,234,775,558,633.0	0.00000000000030914046
13	84,104,164,524,459.0	0.00000000000001189002
14	2,354,916,606,684,850.0	0.00000000000000042464
15	70,647,498,200,545,600.0	0.00000000000000001415
16	2,260,719,942,417,460,000.0	0.00000000000000000044
17	76,864,478,042,193,600,000.0	0.00000000000000000001
18	2,767,121,209,518,970,000,000.0	0.00000000000000000000
19	105,150,605,961,721,000,000,000.0	0.00000000000000000000
20	4,206,024,238,468,830,000,000,000.0	0.00000000000000000000

Structure of the MATLAB CODE for the algorithm presented on page 67:

```
p= x.x, Ctask= x, s= x; % Note p is instead of rho,
nstar is instead of n*
I(1)=1/p+1; % Note G(0)=1, I(0)=1 and I(1)=I(0)/p+1;
G(1)=1/I(1);
Ctot(1)= Ctask*G(1)+s*[1-p*(1-G(1))];
 for n=2:1000
    I(n)=I(n-1)*n/p+1; %Recursive calculation;
    G(n)=1/I(n);
    Ctot(n)= Ctask*G(n)+s*(n-p*(1-G(n)));
    if Ctot(n)>Ctot(n-1) % If Ctot increases then stop;
       nstar=n-1;  % The minimizer is mf=n-1;
       Ctotmin=Ctot(n-1) % The minimum value of Ctot;
       break % Break the cycle and return;
    end
end
disp(nstar, Ctotmin)
```

Figure 9: MATLAB code of the algorithm for the calculation of n^*

The recursive and cyclical process, represented in the MATLAB code from line six up to line fourteen, that the algorithm performs, after having fixed the starting value of $G(1)$ and C_{tot} with $n = 1$ is:

- line six: find the value of $I(n)$ using the value of $I(n-1)$ (that for n=2 is $I(1)$ calculated in the first line of the code;
- line seven: compute the value of $G(n)$ using the value of $I(n)$ calculated in the previous line;
- line eight: calculate the amount of C_{tot} using $G(n)$ found in line seven;
- line nine: start the cycle "if": *if* the value calculated in line eight is bigger than the value of C_{tot} computed with $n-1$, then break the cycle and the final results are $n^* = n-1$ and the value of C_{tot} minimum is the amount of C_{tot} using $n-1$ (in the Equation 17: Total Cost of the Retainer Model equation, C_{tot});
- instead, *if* the two sequential values of C_{tot} with n and $n+1$, are decreasing, repeat the cycle using the next value of n.

As stated on pages 67 and following, here the screen-shots of the examples obtained running the algorithm with certain values of the parameters with the MATLAB code presented above. With this algorithm I found the same values (*better safe than sorry*) shown in the tables and found with the graphical approach used by the authors. These are:

Example 1. Setting $\rho = 0.5$, $C_{task} = 100$ and $s = 1$, we can see that the algorithm runs perfectly comparing the final results with Table 4 below in the appendix:

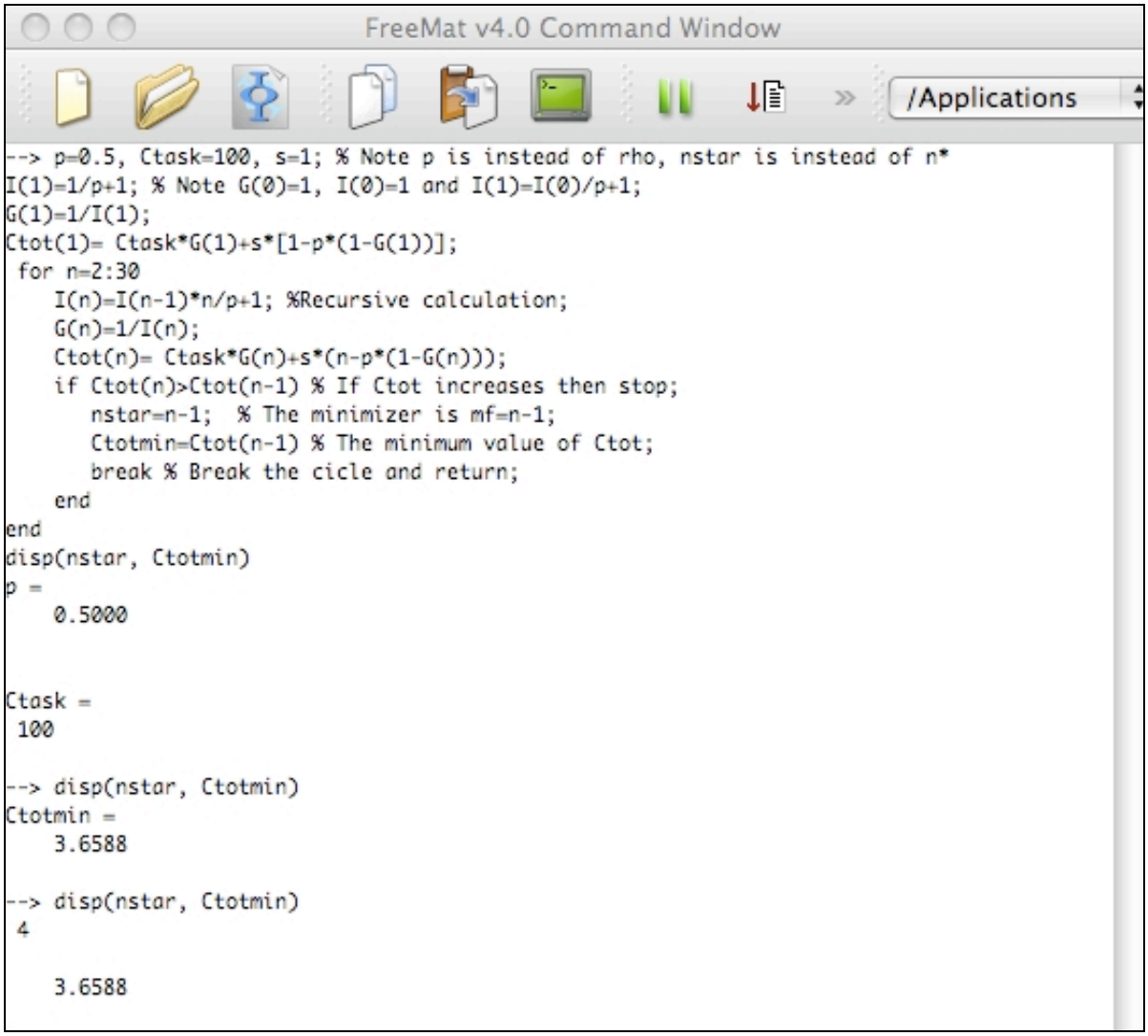

Figure 10: automatic results by the algorithm with $\rho = 0.5$, $C_{task} = 100$ and $s = 1$

Example 2. Setting set $\rho = 1.0$, $C_{task} = 100$ and $s = 1$ we can see that the algorithm runs perfectly comparing the final results with Table 4 below:

Figure 11: automatic results by the algorithm with $\rho = 1.0$, $C_{task} = 100$ and $s = 1$

Example 3. Setting $\rho = 2.5$, $C_{task} = 100$ and $s = 1$ we can see that the algorithm runs perfectly comparing the final results with Table 6 in the appendix:

```
--> disp(nstar, Ctotmin)
 7

    5.5233

-->
```

Figure 12: automatic results by the algorithm with $\rho = 2.5$, $C_{task} = 100$ and $s = 1$

The tables showing the different values of C_{tot}, computed fixing $s = 1$ and $C_{task} = 100$ and varying the value of ρ, can be found in this last part. This has been done in order to have, always ready, the amount of C_{tot} without losing time every time for calculate it.

How to read them: in the first column the different values of n (from 0 up to 23) are present and for each n, fixing the value of $C_{task} = 100$ and $s = 1$, I calculated the value of C_{tot}, substituting the value of $\hat{\rho}$ with a numerical value from 0.1 to 10 (with an increment of 0.1). This calculations has been made in order to build 100 different columns for 100 different values of C_{tot}, using *Equation 49: formula for the computation of the tables in appendix* presented on page 76.

Below Table 4 shows the values of C_{tot} with $s = 1$, $C_{task} = 100$ and $0.1 \leq \rho \leq 1$:

Table 4: values of C_{tot} with $s = 1$, $C_{task} = 100$ and $0.1 \leq \rho \leq 1$

C_{task}	***100***	100	100	100	100	100	100	100	100	100
s	***1***	1	1	1	1	1	1	1	1	1
ρ	***0.1***	**0.2**	**0.3**	**0.4**	**0.5**	**0.6**	**0.7**	**0.8**	**0.9**	**1**
n	C_{tot}	C_{tot}	C_{tot}	C_{tot}	C_{tot}	C_{tot}	C_{tot}	C_{tot}	C_{tot}	C_{tot}
0	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$
1	10.000	17.500	23.846	29.286	34.000	38.125	41.765	45.000	47.895	$\rho > n$
2	**2.353**	3.443	5.056	7.027	9.231	11.573	13.985	16.415	18.829	21.200
3	2.915	**2.909**	**3.034**	**3.318**	3.772	4.394	5.175	6.100	7.152	8.313
4	3.900	3.805	3.725	3.672	**3.659**	**3.698**	**3.801**	**3.974**	**4.224**	4.554
5	4.900	4.800	4.702	4.606	4.516	4.436	4.370	4.324	4.302	**4.310**
6	5.900	5.800	5.700	5.600	5.501	5.404	5.308	5.216	5.130	5.052
7	6.900	6.800	6.700	6.600	6.500	6.400	6.301	6.202	6.104	6.007
8	7.900	7.800	7.700	7.600	7.500	7.400	7.300	7.200	7.100	7.001
9	8.900	8.800	8.700	8.600	8.500	8.400	8.300	8.200	8.100	8.000
10	9.900	9.800	9.700	9.600	9.500	9.400	9.300	9.200	9.100	9.000
11	10.900	10.800	10.700	10.600	10.500	10.400	10.300	10.200	10.100	10.000
12	11.900	11.800	11.700	11.600	11.500	11.400	11.300	11.200	11.100	11.000
13	12.900	12.800	12.700	12.600	12.500	12.400	12.300	12.200	12.100	12.000
14	13.900	13.800	13.700	13.600	13.500	13.400	13.300	13.200	13.100	13.000
15	14.900	14.800	14.700	14.600	14.500	14.400	14.300	14.200	14.100	14.000
16	15.900	15.800	15.700	15.600	15.500	15.400	15.300	15.200	15.100	15.000
17	16.900	16.800	16.700	16.600	16.500	16.400	16.300	16.200	16.100	16.000
18	17.900	17.800	17.700	17.600	17.500	17.400	17.300	17.200	17.100	17.000
19	18.900	18.800	18.700	18.600	18.500	18.400	18.300	18.200	18.100	18.000
20	19.900	19.800	19.700	19.600	19.500	19.400	19.300	19.200	19.100	19.000
21	20.900	20.800	20.700	20.600	20.500	20.400	20.300	20.200	20.100	20.000
22	21.900	21.800	21.700	21.600	21.500	21.400	21.300	21.200	21.100	21.000
23	22.900	22.800	22.700	22.600	22.500	22.400	22.300	22.200	22.100	22.000
C_{tot} **minimum**	**2.353**	**2.909**	**3.034**	**3.318**	**3.659**	**3.698**	**3.801**	**3.974**	**4.224**	**4.310**

Below table 5 shows the values of C_{tot} with $s = 1$. $C_{task} = 100$ and $1.1 \leq \rho \leq 2$:

Table 5: values of C_{tot} with $s = 1$, $C_{task} = 100$ and $1.1 \leq \rho \leq 2$

C_{task}	100	100	100	100	100	100	100	100	100	100
s	1	1	1	1	1	1	1	1	1	1
ρ	**1.1**	**1.2**	**1.3**	**1.4**	**1.5**	**1.6**	**1.7**	**1.8**	**1.9**	**2**
n	C_{tot}	C_{tot}	C_{tot}	C_{tot}	C_{tot}	C_{tot}	C_{tot}	C_{tot}	C_{tot}	C_{tot}
0	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$
1	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$
2	23.512	25.753	27.917	30.000	32.000	33.918	35.754	37.511	39.192	$\rho > n$
3	9.563	10.885	12.264	13.685	15.134	16.601	18.076	19.551	21.019	22.474
4	4.964	5.454	6.021	6.660	7.368	8.137	8.963	9.838	10.758	11.714
5	**4.352**	**4.433**	**4.556**	**4.724**	4.940	5.203	5.516	5.878	6.287	6.743
6	4.983	4.926	4.885	4.862	**4.859**	**4.879**	**4.924**	**4.997**	**5.099**	**5.233**
7	5.913	5.822	5.734	5.652	5.577	5.509	5.451	5.405	5.371	5.351
8	6.902	6.803	6.706	6.609	6.514	6.422	6.332	6.246	6.164	6.088
9	7.900	7.800	7.701	7.601	7.502	7.404	7.306	7.209	7.114	7.019
10	8.900	8.800	8.700	8.600	8.500	8.401	8.301	8.202	8.103	8.004
11	9.900	9.800	9.700	9.600	9.500	9.400	9.300	9.200	9.100	9.001
12	10.900	10.800	10.700	10.600	10.500	10.400	10.300	10.200	10.100	10.000
13	11.900	11.800	11.700	11.600	11.500	11.400	11.300	11.200	11.100	11.000
14	12.900	12.800	12.700	12.600	12.500	12.400	12.300	12.200	12.100	12.000
15	13.900	13.800	13.700	13.600	13.500	13.400	13.300	13.200	13.100	13.000
16	14.900	14.800	14.700	14.600	14.500	14.400	14.300	14.200	14.100	14.000
17	15.900	15.800	15.700	15.600	15.500	15.400	15.300	15.200	15.100	15.000
18	16.900	16.800	16.700	16.600	16.500	16.400	16.300	16.200	16.100	16.000
19	17.900	17.800	17.700	17.600	17.500	17.400	17.300	17.200	17.100	17.000
20	18.900	18.800	18.700	18.600	18.500	18.400	18.300	18.200	18.100	18.000
21	19.900	19.800	19.700	19.600	19.500	19.400	19.300	19.200	19.100	19.000
22	20.900	20.800	20.700	20.600	20.500	20.400	20.300	20.200	20.100	20.000
23	21.900	21.800	21.700	21.600	21.500	21.400	21.300	21.200	21.100	21.000
C_{tot} **minimum**	**4.352**	**4.433**	**4.556**	**4.724**	**4.859**	**4.879**	**4.924**	**4.997**	**5.099**	**5.233**

Below table 6 shows the values of C_{tot} with $s = 1$. $C_{task} = 100$ and $2.1 \leq \rho \leq 3$:

Table 6: values of C_{tot} with $s = 1$, $C_{task} = 100$ and $2.1 \leq \rho \leq 3$

C_{task}	100	100	100	100	100	100	100	100	100	100
s	1	1	1	1	1	1	1	1	1	1
ρ	**2.1**	**2.2**	**2.3**	**2.4**	**2.5**	**2.6**	**2.7**	**2.8**	**2.9**	**3**
n	C_{tot}	C_{tot}	C_{tot}	C_{tot}	C_{tot}	C_{tot}	C_{tot}	C_{tot}	C_{tot}	C_{tot}
0	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$
1	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$
2	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$
3	23.911	25.327	26.719	28.085	29.422	30.730	32.007	33.254	34.470	$\rho > n$
4	12.703	13.717	14.752	15.803	16.866	17.937	19.011	20.087	21.160	22.229
5	7.244	7.788	8.371	8.992	9.647	10.334	11.049	11.790	12.553	13.336
6	5.398	5.597	5.829	6.095	6.394	6.727	7.092	7.489	7.916	8.372
7	**5.347**	**5.362**	**5.395**	**5.448**	**5.523**	**5.621**	**5.742**	5.887	6.057	6.252
8	6.017	5.954	5.899	5.854	5.819	5.795	5.784	**5.787**	**5.805**	**5.838**
9	6.927	6.838	6.751	6.668	6.588	6.514	6.445	6.382	6.327	6.278
10	7.906	7.808	7.712	7.616	7.522	7.430	7.339	7.251	7.166	7.083
11	8.901	8.802	8.702	8.604	8.505	8.407	8.310	8.213	8.117	8.023
12	9.900	9.800	9.700	9.601	9.501	9.402	9.302	9.203	9.104	9.006
13	10.900	10.800	10.700	10.600	10.500	10.400	10.300	10.201	10.101	10.001
14	11.900	11.800	11.700	11.600	11.500	11.400	11.300	11.200	11.100	11.000
15	12.900	12.800	12.700	12.600	12.500	12.400	12.300	12.200	12.100	12.000
16	13.900	13.800	13.700	13.600	13.500	13.400	13.300	13.200	13.100	13.000
17	14.900	14.800	14.700	14.600	14.500	14.400	14.300	14.200	14.100	14.000
18	15.900	15.800	15.700	15.600	15.500	15.400	15.300	15.200	15.100	15.000
19	16.900	16.800	16.700	16.600	16.500	16.400	16.300	16.200	16.100	16.000
20	17.900	17.800	17.700	17.600	17.500	17.400	17.300	17.200	17.100	17.000
21	18.900	18.800	18.700	18.600	18.500	18.400	18.300	18.200	18.100	18.000
22	19.900	19.800	19.700	19.600	19.500	19.400	19.300	19.200	19.100	19.000
23	20.900	20.800	20.700	20.600	20.500	20.400	20.300	20.200	20.100	20.000
C_{tot} **minimum**	**5.347**	**5.362**	**5.395**	**5.448**	**5.523**	**5.621**	**5.742**	**5.787**	**5.805**	**5.838**

Below table 7 shows the values of C_{tot} with $s = 1$. $C_{task} = 100$ and $3.1 \leq \rho \leq 4$:

Table 7: values of C_{tot} with $s = 1$, $C_{task} = 100$ and $3.1 \leq \rho \leq 4$

C_{task}	100	100	100	100	100	100	100	100	100	100
s	1	1	1	1	1	1	1	1	1	1
ρ	**3.1**	**3.2**	**3.3**	**3.4**	**3.5**	**3.6**	**3.7**	**3.8**	**3.9**	**4**
n	C_{tot}	C_{tot}	C_{tot}	C_{tot}	C_{tot}	C_{tot}	C_{tot}	C_{tot}	C_{tot}	C_{tot}
0	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$
1	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$
2	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$
3	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$
4	23.291	24.345	25.388	26.419	27.438	28.443	29.433	30.408	31.368	$\rho > n$
5	14.135	14.949	15.774	16.609	17.451	18.298	19.148	20.000	20.852	21.703
6	8.856	9.366	9.901	10.459	11.037	11.635	12.250	12.881	13.527	14.185
7	6.472	6.717	6.987	7.281	7.599	7.941	8.306	8.692	9.099	9.526
8	**5.887**	**5.954**	**6.038**	**6.141**	6.263	6.404	6.565	6.745	6.944	7.164
9	6.239	6.209	6.188	6.179	**6.181**	**6.195**	**6.223**	**6.263**	**6.318**	**6.387**
10	7.005	6.931	6.861	6.796	6.738	6.686	6.640	6.603	6.573	6.552
11	7.930	7.838	7.748	7.661	7.576	7.493	7.414	7.339	7.267	7.200
12	8.908	8.810	8.713	8.617	8.522	8.428	8.335	8.244	8.154	8.067
13	9.902	9.802	9.703	9.604	9.506	9.408	9.310	9.213	9.116	9.021
14	10.900	10.801	10.701	10.601	10.501	10.402	10.303	10.203	10.105	10.006
15	11.900	11.800	11.700	11.600	11.500	11.400	11.301	11.201	11.101	11.002
16	12.900	12.800	12.700	12.600	12.500	12.400	12.300	12.200	12.100	12.000
17	13.900	13.800	13.700	13.600	13.500	13.400	13.300	13.200	13.100	13.000
18	14.900	14.800	14.700	14.600	14.500	14.400	14.300	14.200	14.100	14.000
19	15.900	15.800	15.700	15.600	15.500	15.400	15.300	15.200	15.100	15.000
20	16.900	16.800	16.700	16.600	16.500	16.400	16.300	16.200	16.100	16.000
21	17.900	17.800	17.700	17.600	17.500	17.400	17.300	17.200	17.100	17.000
22	18.900	18.800	18.700	18.600	18.500	18.400	18.300	18.200	18.100	18.000
23	19.900	19.800	19.700	19.600	19.500	19.400	19.300	19.200	19.100	19.000
C_{tot} **minimum**	**5.887**	**5.954**	**6.038**	**6.141**	**6.181**	**6.195**	**6.223**	**6.263**	**6.318**	**6.387**

Below table 8 shows the values of C_{tot} with $s = 1$. $C_{task} = 100$ and $4.1 \leq \rho \leq 5$:

Table 8: values of C_{tot} with $s = 1$, $C_{task} = 100$ and $4.1 \leq \rho \leq 5$

C_{task}	100	100	100	100	100	100	100	100	100	100
s	1	1	1	1	1	1	1	1	1	1
ρ	**4.1**	**4.2**	**4.3**	**4.4**	**4.5**	**4.6**	**4.7**	**4.8**	**4.9**	**5**
n	C_{tot}	C_{tot}	C_{tot}	C_{tot}	C_{tot}	C_{tot}	C_{tot}	C_{tot}	C_{tot}	C_{tot}
0	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$
1	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$
2	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$
3	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$
4	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$
5	22.551	23.395	24.235	25.069	25.896	26.716	27.528	28.331	29.126	$\rho > n$
6	14.854	15.532	16.219	16.912	17.610	18.313	19.019	19.727	20.435	21.144
7	9.972	10.436	10.916	11.412	11.923	12.447	12.983	13.530	14.088	14.654
8	7.402	7.660	7.937	8.232	8.544	8.875	9.221	9.584	9.962	10.355
9	**6.471**	6.571	6.686	6.816	6.963	7.125	7.304	7.498	7.708	7.933
10	6.540	**6.538**	**6.547**	**6.566**	**6.597**	**6.639**	**6.693**	**6.759**	**6.838**	6.930
11	7.138	7.081	7.030	6.985	6.947	6.915	6.892	6.876	6.869	**6.870**
12	7.981	7.898	7.818	7.741	7.667	7.597	7.531	7.470	7.413	7.361
13	8.926	8.832	8.739	8.648	8.558	8.470	8.384	8.299	8.218	8.139
14	9.908	9.810	9.712	9.615	9.519	9.423	9.328	9.234	9.141	9.050
15	10.902	10.803	10.703	10.604	10.506	10.407	10.309	10.211	10.113	10.017
16	11.901	11.801	11.701	11.601	11.502	11.402	11.303	11.203	11.104	11.005
17	12.900	12.800	12.700	12.600	12.500	12.401	12.301	12.201	12.101	12.002
18	13.900	13.800	13.700	13.600	13.500	13.400	13.300	13.200	13.100	13.000
19	14.900	14.800	14.700	14.600	14.500	14.400	14.300	14.200	14.100	14.000
20	15.900	15.800	15.700	15.600	15.500	15.400	15.300	15.200	15.100	15.000
21	16.900	16.800	16.700	16.600	16.500	16.400	16.300	16.200	16.100	16.000
22	17.900	17.800	17.700	17.600	17.500	17.400	17.300	17.200	17.100	17.000
23	18.900	18.800	18.700	18.600	18.500	18.400	18.300	18.200	18.100	18.000
C_{tot} **minimum**	**6.471**	**6.538**	**6.547**	**6.566**	**6.597**	**6.639**	**6.693**	**6.759**	**6.838**	**6.870**

Below table 9 shows the values of C_{tot} with $s = 1$. $C_{task} = 100$ and $5.1 \leq \rho \leq 6$:

Table 9: values of C_{tot} with $s = 1$, $C_{task} = 100$ and $5.1 \leq \rho \leq 6$

C_{task}	100	100	100	100	100	100	100	100	100	100
s	1	1	1	1	1	1	1	1	1	1
ρ	**5.1**	**5.2**	**5.3**	**5.4**	**5.5**	**5.6**	**5.7**	**5.8**	**5.9**	**6**
n	C_{tot}	C_{tot}	C_{tot}	C_{tot}	C_{tot}	C_{tot}	C_{tot}	C_{tot}	C_{tot}	C_{tot}
0	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$
1	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$
2	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$
3	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$
4	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$
5	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$
6	21.852	22.558	23.263	23.964	24.662	25.356	26.045	26.729	27.408	$\rho > n$
7	15.229	15.811	16.399	16.992	17.589	18.190	18.794	19.400	20.008	20.616
8	10.762	11.181	11.614	12.057	12.512	12.976	13.450	13.932	14.422	14.919
9	8.174	8.429	8.700	8.984	9.283	9.595	9.919	10.256	10.605	10.965
10	7.035	7.153	7.285	7.430	7.587	7.759	7.943	8.140	8.350	8.573
11	**6.881**	**6.901**	**6.931**	**6.971**	**7.021**	**7.083**	**7.155**	7.238	7.332	7.437
12	7.315	7.275	7.241	7.213	7.193	7.179	7.174	**7.176**	**7.186**	**7.205**
13	8.063	7.990	7.920	7.854	7.792	7.735	7.682	7.634	7.591	7.553
14	8.959	8.870	8.783	8.698	8.615	8.534	8.455	8.379	8.306	8.237
15	9.920	9.824	9.729	9.635	9.542	9.450	9.359	9.269	9.181	9.095
16	10.906	10.808	10.710	10.612	10.514	10.417	10.321	10.225	10.130	10.035
17	11.902	11.802	11.703	11.604	11.505	11.406	11.307	11.209	11.110	11.013
18	12.901	12.801	12.701	12.601	12.501	12.402	12.302	12.203	12.103	12.004
19	13.900	13.800	13.700	13.600	13.500	13.401	13.301	13.201	13.101	13.001
20	14.900	14.800	14.700	14.600	14.500	14.400	14.300	14.200	14.100	14.000
21	15.900	15.800	15.700	15.600	15.500	15.400	15.300	15.200	15.100	15.000
22	16.900	16.800	16.700	16.600	16.500	16.400	16.300	16.200	16.100	16.000
23	17.900	17.800	17.700	17.600	17.500	17.400	17.300	17.200	17.100	17.000
C_{tot} **minimum**	**6.881**	**6.901**	**6.931**	**6.971**	**7.021**	**7.083**	**7.155**	**7.176**	**7.186**	**7.205**

Below table 8 shows the values of C_{tot} with $s = 1$. $C_{task} = 100$ and $6.1 \leq \rho \leq 7$:

Table 10: values of C_{tot} with $s = 1$, $C_{task} = 100$ and $6.1 \leq \rho \leq 7$

C_{task}	100	100	100	100	100	100	100	100	100	100
s	1	1	1	1	1	1	1	1	1	1
ρ	**6.1**	**6.2**	**6.3**	**6.4**	**6.5**	**6.6**	**6.7**	**6.8**	**6.9**	**7**
n	C_{tot}	C_{tot}	C_{tot}	C_{tot}	C_{tot}	C_{tot}	C_{tot}	C_{tot}	C_{tot}	C_{tot}
0	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$
1	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$
2	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$
3	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$
4	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$
5	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$
6	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$
7	21.224	21.832	22.440	23.045	23.649	24.251	24.850	25.447	26.040	$\rho > n$
8	15.422	15.931	16.445	16.964	17.486	18.011	18.539	19.069	19.601	20.134
9	11.336	11.717	12.108	12.508	12.916	13.332	13.756	14.186	14.623	15.065
10	8.808	9.055	9.314	9.584	9.866	10.158	10.460	10.773	11.094	11.425
11	7.554	7.682	7.821	7.971	8.133	8.306	8.490	8.685	8.890	9.106
12	**7.232**	**7.268**	**7.314**	**7.368**	**7.432**	7.506	7.589	7.682	7.785	7.898
13	7.521	7.496	7.476	7.463	7.457	**7.459**	**7.467**	**7.483**	**7.506**	**7.538**
14	8.170	8.107	8.048	7.993	7.943	7.897	7.856	7.819	7.789	7.763
15	9.010	8.927	8.846	8.768	8.691	8.618	8.548	8.480	8.416	8.355
16	9.942	9.849	9.757	9.667	9.578	9.490	9.404	9.319	9.236	9.155
17	10.915	10.818	10.721	10.625	10.530	10.435	10.341	10.248	10.155	10.064
18	11.905	11.806	11.707	11.609	11.511	11.413	11.315	11.218	11.121	11.025
19	12.902	12.802	12.702	12.603	12.504	12.404	12.305	12.206	12.108	12.009
20	13.900	13.801	13.701	13.601	13.501	13.401	13.302	13.202	13.103	13.003
21	14.900	14.800	14.700	14.600	14.500	14.400	14.301	14.201	14.101	14.001
22	15.900	15.800	15.700	15.600	15.500	15.400	15.300	15.200	15.100	15.000
23	16.900	16.800	16.700	16.600	16.500	16.400	16.300	16.200	16.100	16.000
C_{tot} **minimum**	**7.232**	**7.268**	**7.314**	**7.368**	**7.432**	**7.459**	**7.467**	**7.483**	**7.506**	**7.538**

Below table 11 shows the values of C_{tot} with $s = 1$, $C_{task} = 100$ and $7.1 \leq \rho \leq 8$:

Table 11: values of C_{tot} with $s = 1$, $C_{task} = 100$ and $7.1 \leq \rho \leq 8$

C_{task}	100	100	100	100	100	100	100	100	100	100
s	1	1	1	1	1	1	1	1	1	1
ρ	**7.1**	**7.2**	**7.3**	**7.4**	**7.5**	**7.6**	**7.7**	**7.8**	**7.9**	**8**
n	C_{tot}	C_{tot}	C_{tot}	C_{tot}	C_{tot}	C_{tot}	C_{tot}	C_{tot}	C_{tot}	C_{tot}
0	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$
1	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$
2	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$
3	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$
4	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$
5	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$
6	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$
7	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$
8	20.668	21.202	21.735	22.269	22.801	23.333	23.863	24.391	24.918	$\rho > n$
9	15.512	15.965	16.421	16.882	17.345	17.812	18.281	18.752	19.225	19.699
10	11.765	12.113	12.468	12.831	13.201	13.577	13.960	14.348	14.741	15.139
11	9.332	9.568	9.813	10.068	10.332	10.605	10.887	11.177	11.474	11.779
12	8.020	8.152	8.294	8.446	8.607	8.778	8.958	9.147	9.345	9.552
13	**7.577**	**7.625**	**7.681**	7.745	7.818	7.900	7.990	8.089	8.196	8.312
14	7.744	7.730	7.723	**7.722**	**7.728**	**7.740**	**7.759**	**7.786**	**7.819**	**7.860**
15	8.298	8.245	8.196	8.151	8.110	8.075	8.044	8.018	7.998	7.983
16	9.076	9.000	8.926	8.854	8.785	8.720	8.657	8.597	8.542	8.489
17	9.974	9.885	9.797	9.711	9.626	9.543	9.461	9.382	9.305	9.230
18	10.929	10.834	10.739	10.645	10.552	10.460	10.369	10.279	10.190	10.102
19	11.911	11.813	11.715	11.618	11.521	11.424	11.328	11.232	11.137	11.043
20	12.904	12.805	12.706	12.607	12.508	12.409	12.311	12.213	12.115	12.017
21	13.901	13.802	13.702	13.602	13.503	13.403	13.304	13.205	13.106	13.007
22	14.900	14.801	14.701	14.601	14.501	14.401	14.301	14.202	14.102	14.002
23	15.900	15.800	15.700	15.600	15.500	15.400	15.300	15.201	15.101	15.001
C_{tot} **minimum**	**7.577**	**7.625**	**7.681**	**7.722**	**7.728**	**7.740**	**7.759**	**7.786**	**7.819**	**7.860**

Below table 12 shows the values of C_{tot} with $s = 1$, $C_{task} = 100$ and $8.1 \leq \rho \leq 9$:

Table 12: values of C_{tot} with $s = 1$, $C_{task} = 100$ and $8.1 \leq \rho \leq 9$

C_{task}	100	100	100	100	100	100	100	100	100	100
s	1	1	1	1	1	1	1	1	1	1
ρ	**8.1**	**8.2**	**8.3**	**8.4**	**8.5**	**8.6**	**8.7**	**8.8**	**8.9**	**9**
n	C_{tot}	C_{tot}	C_{tot}	C_{tot}	C_{tot}	C_{tot}	C_{tot}	C_{tot}	C_{tot}	C_{tot}
0	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$
1	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$
2	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$
3	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$
4	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$
5	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$
6	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$
7	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$
8	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$
9	20.174	20.650	21.127	21.603	22.079	22.555	23.030	23.504	23.977	$\rho > n$
10	15.542	15.949	16.359	16.773	17.190	17.609	18.031	18.455	18.881	19.308
11	12.091	12.410	12.736	13.068	13.405	13.749	14.097	14.450	14.808	15.169
12	9.767	9.991	10.223	10.464	10.711	10.966	11.229	11.498	11.774	12.056
13	8.436	8.569	8.711	8.861	9.019	9.185	9.359	9.541	9.731	9.929
14	**7.908**	**7.964**	8.027	8.097	8.176	8.262	8.356	8.457	8.566	8.683
15	7.974	7.970	**7.972**	**7.981**	**7.995**	**8.016**	**8.044**	**8.078**	**8.118**	**8.166**
16	8.441	8.396	8.356	8.320	8.289	8.262	8.240	8.223	8.211	8.205
17	9.157	9.087	9.019	8.955	8.893	8.834	8.779	8.727	8.679	8.634
18	10.016	9.931	9.847	9.765	9.685	9.607	9.531	9.457	9.385	9.316
19	10.949	10.856	10.764	10.673	10.583	10.494	10.406	10.319	10.234	10.150
20	11.920	11.823	11.727	11.631	11.535	11.440	11.346	11.252	11.159	11.067
21	12.908	12.809	12.711	12.612	12.514	12.416	12.319	12.222	12.125	12.029
22	13.903	13.803	13.704	13.605	13.506	13.406	13.308	13.209	13.110	13.012
23	14.901	14.801	14.701	14.602	14.502	14.402	14.303	14.203	14.104	14.005
C_{tot} **minimum**	**7.908**	**7.964**	**7.972**	**7.981**	**7.995**	**8.016**	**8.044**	**8.078**	**8.118**	**8.166**

Below table 13 shows the values of C_{tot} with $s = 1$, $C_{task} = 100$ and $9.1 \leq \rho \leq 10$:

Table 13: values of C_{tot} with $s = 1$, $C_{task} = 100$ and $9.1 \leq \rho \leq 10$

C_{task}	100	100	100	100	100	100	100	100	100	100
s	1	1	1	1	1	1	1	1	1	1
ρ	**9.1**	**9.2**	**9.3**	**9.4**	**9.5**	**9.6**	**9.7**	**9.8**	**9.9**	**10**
n	C_{tot}	C_{tot}	C_{tot}	C_{tot}	C_{tot}	C_{tot}	C_{tot}	C_{tot}	C_{tot}	C_{tot}
0	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$
1	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$
2	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$
3	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$
4	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$
5	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$
6	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$
7	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$
8	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$
9	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$	$\rho > n$
10	19.736	20.165	20.595	21.026	21.456	21.886	22.317	22.746	23.176	$\rho > n$
11	15.535	15.905	16.277	16.653	17.031	17.412	17.795	18.180	18.567	18.956
12	12.345	12.639	12.939	13.244	13.554	13.869	14.189	14.512	14.840	15.171
13	10.134	10.346	10.565	10.791	11.023	11.262	11.507	11.758	12.015	12.277
14	8.807	8.938	9.078	9.224	9.378	9.538	9.706	9.881	10.062	10.250
15	8.220	8.281	8.348	8.423	8.504	8.593	8.688	8.790	8.899	9.015
16	**8.204**	**8.208**	**8.218**	**8.234**	**8.255**	**8.283**	**8.316**	**8.356**	**8.401**	8.453
17	8.593	8.557	8.524	8.496	8.472	8.453	8.438	8.429	8.424	**8.424**
18	9.249	9.185	9.124	9.066	9.011	8.959	8.910	8.865	8.823	8.786
19	10.067	9.986	9.907	9.830	9.755	9.682	9.611	9.542	9.476	9.412
20	10.976	10.886	10.796	10.708	10.621	10.535	10.450	10.367	10.286	10.206
21	11.933	11.837	11.743	11.648	11.555	11.462	11.369	11.278	11.187	11.098
22	12.914	12.816	12.718	12.621	12.524	12.427	12.331	12.235	12.139	12.044
23	13.905	13.806	13.707	13.608	13.510	13.411	13.313	13.215	13.117	13.019
C_{tot} **minimum**	**8.204**	**8.208**	**8.218**	**8.234**	**8.255**	**8.283**	**8.316**	**8.356**	**8.401**	**8.424**

Bibliography

Adelman, D. "A Simple Algebraic Approximation to the Erlang Loss System." University of Chicago, 2007.

Agnetis, A. "Appunti sui problemi di matching."

Baiocco, A. "Crowdsourcing, quando è "la folla" a generare valore per l'impresa."

Bauer, U. "Assignment Problem with Constraints." Technische Universitat Munchen - Fakultat fur Informatik.

Bernstein M., Karger D., Miller R., Brandt J. "Analytic Methods for Optimizing Realtime Crowdsourcing." *CI: Collective Intelligence 2012.* 2012.

—. "Crowds in two seconds: enabling realtime crowd-powered interfaces." *UIST: ACM Symposium on User Interface Software and Technology 2011.* 2011.

Bigham J.P., Jayant C., Ji H., Little G., Miller A., Miller R.C., Miller R., Tatarowicz A., White B., White S., Yeh T. "VizWiz: Nearly Real-time Answers to Visual Questions." *Proceedings of the ACM Symposium on User Interface Software and Technology (UIST 2010).* 2010.

Carbone, C. "Teoria del traffico telefonico."

Carpaneto G., Toth P. "PRIMAL-DUAL ALGORITHMS FOR THE ASSIGNMENT." University of Bologna.

Daren, Brabham. "Crowdsourcing as a Model for Problem Solving: An Introduction and Cases." *Convergence: The International Journal of Research into New Media Technologies*, 2008.

Esteves J. S., Craveirinha J. "On a bicriterion server allocation problem in a multidimensional Erlang loss system." *Journal of Computational and Applied Mathematics*, 2013.

Esteves J. S., Craveirinha J., Cardoso D. M. "Computing erlang–b function derivatives in the number of servers." *Communications in Statistics. Stochastic Models*, 1995.

—. "Second order Conditions on the Overflow Traffic from the Erlang-B System." *Journal of Mathematical Sciences*, 2009, 6 ed.

Esteves, J. S. "Efficient Algorithms for Higher-Order Derivatives of the Continued Erlang Delay Function." University of Aveiro.

—. "Equidade e Eficiˆencia na Optimizac˜ao de Sistemas Multidimensionais de Erlang-B." Universidade de Aveiro - Departamento de Matematica, 2006.

—. "Numerical Computation of High Order Derivatives of Erlang B and C Functions."

Gross D., Harris C.M. *Fundamentals of Queueing Theory.* John Wiley & Sons, 1985.

Howe, Jeff. *Crowdsourcing. Why the power of the crowd is driving the future of business.* Three Rivers Press, 2008.

Iversen, V. B. "TELETRAFFIC ENGINEERING and NETWORK PLANNING." Technical University of Denmark, 2010.

Jagerman, D. L. "MATHCALC." 1987.

—. "Methods in traffic calculations." 1984.

—. "An Inversion Technique for the Laplace Transform." *The Bell System Technical Journal*, 1982.

Jagers A.A., Van Doorn E.A. "On the continued Erlang loss function." *Operations Research Letters*, 1986, 1 ed.

Kelley, C. T. *Iterative Methods for Optimization (Frontiers in Applied Mathematics).* Society for Industrial Mathematics, 1987.

Kubasik, J. "ON SOME NUMERICAL METHODS FOR THE COMPUTATION OF ERLANG AND ENGSET FUNCTIONS." Elsevier Science Publishers B. V., 1985.

Limpaecher A., Feltman N., Treuille A., Cohen M. "Real-time drawing assistance through crowdsourcing." *ACM Transactions on Graphics (TOG) - SIGGRAPH 2013 Conference Proceedings*, 2013.

Luna, Riccardo. "Prefazione." In *Crowdsourcing. Il valore partecipativo come risorsa per il futuro del business*, by Jeff Howe. Luca Losella editore, 2010.

Marchi, S. De. "Codici Matlab/Octave." Università di Padova - Dipartimento di Matematica, 2011.

Masi, M. "Metodi Numerici perl'Ottimizzazione - Algoritmi e implementazioni." 2006.

"Matlab: esempi ed esercizi." Università degli studi di Pavia- Dipartimento di Ingegneria.

Messerli, E. J. "Proof of a convexity property of the Erlang B formula." *Bell System Technical Journal*, 1972, 4 ed.

Minka, T. P. "Beyond Newton's method." 2000.

Scaglia, M. "Esercizi sul calcolo di integrali indefiniti e definiti."

Sharma, A. "Crowdsourcing Critical Success Factor Model: Strategies to harness the collective intelligence of the crowd ."

Srdjan, V. "Erlang B and Engset formula calculation." 2010.

Stankovic, John A. "Misconceptions About Real-Time Computing: A Serious Problem for Next-Generation Systems." *Journal Computer*, 1988: 10-19.

Toth, E. "COMPUTATION OF SECOND DERIVATIVES OF ERLANG'S B AND WILKINSON'S FORMULAE AND ITS APPLICATION ON PLANNING OF JUNCTION NETWORKS." Elsevier Science Publishers B. V., 1985.

Varian. *Microeconomic Analysis (Third ed.).* W.W. Norton and Company, 1992.

Venturin, M. "Metodi di Line-search."

Whitla, P. "Crowdsourcing and Its Application in Marketing Activities." *Contemporary Management Research*, 2009.

Whitt, W. "IEOR 6707: Advanced topics in Queueing theory: Focus on customer contact centers." 2002.

Wikipedia. "Crowdsourcing." *Wikipedia.* 2013. http://en.wikipedia.org/wiki/Crowdsourcing (accessed 09 09, 2013).

Zeng, G. "Two Common Properties of the Erlang-B Function, Erlang-C Function and Engset Blocking Function."