Executive summary

The aim of this project was to design and implement an optimization framework capable of improving various trading strategies with use of cloud computing facilities. Optimization is performed with use of Genetic Algorithm, trading strategies which could be optimized are from Zero Intelligence Plus family and cloud computing facilities are provided by Amazon Web Services.

This project was undertaken due to the fact that there is a great potential in cloud computing which make it possible to complete very CPU-intensive tasks in the course of hours or days, which could normally take up to few weeks or months on single computer. Moreover, Amazon Web Services enable to do it cheaply making it possible to realize this thesis. Another thing was to benchmark developed optimization framework against results obtained by professor David Cliff in his paper about ZIP60 trading strategy presented in [2].

Contributions and achievements reached in this thesis:

- configuring Hadoop cluster from Elastic Compute Cloud instances from Amazon Web Services,
- designing a software capable of performing trading strategy optimization with use of Watchmaker and Mahout frameworks,
- benchmarking optimization framework against previous results of ZIP60 optimization performed by Cliff, which resulted in conclusion that the optimization framework is working properly and it can produce results quicker and cheaper than during previous experiments,
- showing that Amazon Web Services have potential which should be used by a single student when dealing with CPU-intensive tasks.

Contents

1 The aims and objectives of the project

This thesis describes results from using ”cloud computing” resources (Amazon Web Services) as the infrastructure on which automated optimization of advanced computerized market-trading strategies can be performed, simultaneously using many remote computers in parallel to dramatically reduce the time taken to achieve significant new results.

Currently there is major interest in automated trading within the global financial markets. Huge savings, and/or huge profits, are likely to be made by any bank or fund-management company that can run automated ”high-frequency trading” (HFT) systems. HFT systems typically operate at faster-then-human speeds, making decisions that involve the processing of huge quantities of raw data, often in real-time. Increasingly, hand-designed HFT strategies are being supplemented or replaced by strategies that have been fine-tuned by automated optimization.

Some early research on automated optimization of market designs and trading strategies, using genetic algorithms, was conducted at Hewlett-Packard Laboratories [37]. This Genetic Algorithm (GA) optimization was highly compute-intensive, and was accomplished by Cliff on clusters of 50 PCs at HP Labs, running all machines in parallel for a few weeks. If performed on a single standalone PC, Cliff’s experiments would have taken up to six years.

In recent years, new tools for implementing such compute-intensive tasks have been designed and made available to a wide range of ordinary users: these are based on ”cloud computing” web services. They are, potentially, a replacement for dedicated compute-clusters and they can give much more power which enables doing bigger experiments in less time. Such services are proposed or offered, for by Google [6], IBM [5], Microsoft [7] and Amazon [8].

For this thesis I decided to design and implement the optimization framework using Amazon Web Services (AWS), and to explore GA optimization of different trading-agent strategies such as Cliff’s "Zero Intelligence Plus" (ZIP) [2] or Vytelingum’s "Adaptive Aggressive" (AA) [4].

Aim resulting from previous objective is to check if this framework is really capable of performing such an optimization in reasonable time and if obtained results and performance are satisfactory. What regards above mentioned trading strategies, basing reasoning is as follows. Number of parameters in those trading strategies can grow, as it does to the use of compute-intensive automated optimization becomes even more attractive. Moreover it can be speeded up by using parallel banks of computers. Because of that cloud computing represented by AWS can give us that speed-up at low cost. The optimization framework is meant to be using open-source solutions and ideas.

The main goal can be broken down into a series of aims which have to be completed in order to complete the optimization framework. They are following:

- Exploring parallel, Distributed (cloud) computing for High-Performance Computing,
- Becoming familiar with AWS,
- Designing and developing the optimization framework software,

- Implementing simulation of Continuous Double Auction market,
- Testing and adjusting the cluster,
- Performing optimization of various trading strategies such as ZIP60 or AA.

This dissertation consists of five main chapters. Chapter 2 describes required background theory and techniques which one have to be familiar with in order to pursue this thesis goals. Chapter 3 presents the design and implementation of the optimization framework. It is followed by chapter 4 describing and discussing results obtained during execution of optimization experiments. Chapter 5 presents critical evaluation and discussion regarding obtained results. Final chapter 6 is devoted to further work and possible improvements which could be realized in scope of the optimization framework.

2 Background theory and techniques

This section describes solutions and technologies which will be utilized in this thesis. The structure of the rest of this chapter is as follows. Section 2.1 characterizes Continuous Double Auction. Next, in section 2.2 Amazon Web Services which will be used are presented. It is followed by section 2.3 describing Monster Muck Mashup architecture on which the framework designed and implemented in scope of this thesis is based. Java Agent Development framework which can be used in simulation of marketplaces populated with trading agents is characterized in section 2.4. Then Zero Intelligence Plus and the Adaptive Aggressive trading agents are presented in section 2.5 and 2.6 respectively. Section 2.7 describes GA which was used in the experiment. Past and present experiments which were realized are presented in section 2.8. Watchmaker framework, which is designed for programming of GA optimization, is described in section 2.9. Mahout framework responsible for preparing Hadoop job is presented in section 2.10 and Hadoop is described in section 2.11.

2.1 Continuous Double Auction

There exist many different auction mechanisms. Nevertheless the one which is nowadays widespread throughout international financial markets is the Continuous Double Auction market mechanism. In fact it is used by primary institutions for trading of equities, commodities and derivatives in markets such as NASDAQ and NYSE. In such a market there are buyers and sellers. They can perform trading in a fixed-duration trading periods by placing buy orders (bids) and sell orders (asks) at any time in a trading session. Moreover if there are bids and asks which are compatible in prices and there is enough of a given good that can be sold or bought according to those bids and asks, a transaction is taking place immediately. Additionally, all traders are informed when new orders are placed or trades are executed.

What is more this market mechanism has been well studied by economists and during experimental studies it has been shown that transaction prices tend to converge to so called equilibrium price. That is a price at which market's theoretical supply and demand curves intersect. But it also has a practical meaning. If transactions are taking place in a point when prices are far from this equilibrium price, that means nothing else but that someone is losing money to someone else's advantage. Because of that it is a desirable feature that CDA market transactions exhibit stable equilibration. It is further analysed in [11]. Graph presenting supply and demand curves with equilibrium price on their intersection is presented in Figure 1[1].

Due to constant computerization of our daily life, there is also such a tendency in auctions. E-commerce is such a phenomena. Due to this fact that transactions are now moving to virtual space, it is possible for software programs, being software trading agents, to participate in such auctions. They may interact with human traders as presented in [12]. Moreover they are far faster and very cheap when compared to humans and they are not

[1] Image from http://mobjectivist.blogspot.com/2006_04_01_archive.html

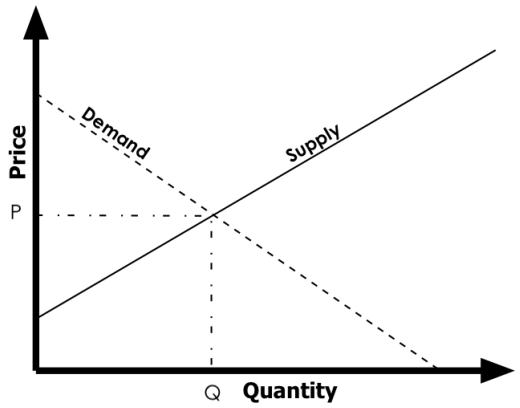

Figure 1: Supply and demand curves, P - equilibrium price, Q - equilibrium quantity

overwhelmed by the volume of data they must process in order to do their job. Even today human traders are also aided by algorithms in duties which were performed traditionally by human brains. Since in such a international financial markets speed of making decision if buying or selling will make profit is vital, humans cannot be left unaided. If there is good trading opportunity, such a program can execute transaction in just a fraction of a second after such an opportunity occurred. It can be observed that during the past few years this style of trading has become known as Algorithmic Trading according to [2]. The speed of reaction needed in financial markets can be easily reached by ZIP traders and scientific press [13] and financial press paid attention [14], [15] to this.

Above mentioned NASDAQ and NYSE are nowadays highly automatized, with a high amount of automated trading. They can be regarded as another instance of online e-marketplaces. Another example is international foreign-exchange markets with sales in a range of $1900bn/day. They are also increasingly computerized making them closer and closer to becoming another online e-marketplaces. The impact of software mechanisms such as ZIP artificial trading agents can be spectacular, according to [12] there is a possibility that online e-marketplaces will be eventually fully populated by software trading agents which may earn additional billions of dollars annually.

2.2 Amazon Web Services(AWS)

In order to optimize ZIP60 or AA trading agent parameters, potentially vast computing power is necessary. Such resources are provided by services given by Amazon.com. The main characteristics of AWS are as follows:

- they are affordable comparing favourably to self-ownership of a data-centre for many users,
- they provide its users with high-performance computing possibilities,
- they are scalable, thus can be used in solutions which are getting more and more resource intensive as the given project progresses.

Those features make it possible to go far beyond a single desktop application.

There are a number of Amazon Web Services [8], nevertheless those which are taken into consideration during the optimization framework implementation are following:

- Amazon Simple Storage Service (Amazon S3)
- Amazon Simple Queue Service (Amazon SQS)
- Amazon Elastic Compute Cloud (Amazon EC2)
- Amazon SimpleDB^{TM}

Above mentioned services provides following functionalities, as is stated in [16]. Amazon S3 is a durable, high bandwidth storage of data. Amazon SQS is a messaging backbone capable of exchanging considerable number of messages per minute. Amazon EC2 gives a scalable infrastructure designed for running applications. Amazon SimpleDB is a database, which provides efficient storage and retrieval of data to support high-performance web applications.

Additionally, those solutions can compete against high end infrastructure of large companies. The main difference is the minimization of costs. The users are only paying for the processing power, storage space and bandwidth which is used.

2.2.1 Connecting to AWS

Another desirable feature is that all AWS solutions can be accessed over the Internet, thus making them available from any point in the world. There are two technologies making it able to achieve. First is Simple Object Access Protocol (SOAP) [17] and another is Representational State Transfer (REST) [18]. Both of those use HTTP as the communication protocol. During implementation of the framework both of those may be used since they give different possibilities. They are presented below.

SOAP and REST differ in few significant ways. First one works as follows. According to [16], when using SAOP, AWS products can be called over the network. This is done in the similar way the remote objects can be called by using Java Remote Method Invocation (RMI) or Common Object Resource Broker Architecture (CORBA). However, SOAP uses XML as the application protocol embedded in the body of HTTP requests, which is not the case when taking Java RMI or CORBA into consideration. What is more, SOAP is already widespread used in the industry and it is attractive selection for Java developers. There exist Java API for XML-based Web Services (JAX-WS).

When compared to SOAP, with REST it is impossible to define custom methods for each type of operation. It only uses a subset of the standard HTTP operations, which are GET, POST, HEAD, PUT and DELETE. All of those are used to interact with given web service. What is more GET, POST, PUT and DELETE can be perceived as Create, Read, Update and Delete (CRUD) associated with relational databases.

To conclude in can be said that REST can be perceived as a data-access methodology and SOAP as a process-control methodology. However, in many cases REST is easier to implement than SOAP and use and that is why AWS provides access to its services through both protocols.

2.2.2 Amazon Simple Storage Service (Amazon S3)

It is designed to store unlimited number of objects with a size not exceeding 5 GB on highly durable storage. Amazon S3 can be accessed through SOAP or REST. Objects can be added, retrieved and deleted thought those protocols over HTTP. According to [16], the most popular Java API for using Amazon S3 is JetS3t, available at [19]. Amazon S3 can be used as a backup service, storage location for files of different formats. The files can be transferred as discrete packed data.

Below main concepts regarding Amazon S3 service are listed, as they are described in [20]. Since one of the features of this service is robustness, it is built with minimal number of features, which are as follows:

- Create buckets - a bucket is a fundamental data container
- Store data in buckets - limitless number of objects can be uploaded into single bucket, nevertheless each object may not exceed 5GB in size. They are retrieved using developer-assigned key.
- Download data - either a developer or other users can download given data.
- Permissions - developer may specify who may download or upload data to given bucket

There is also a number of Amazon S3 concepts which are vital to understand in order to start using this service properly. The are as follows:

- Bucket - container for objects, Amazon S3 namespace is organized through them at the highest level, they perform identification of an account responsible for storage and data transfer charges. Moreover they are responsible for access control and they serve as the unit of aggregation for usage reporting.
- Object - it consists of data and metadata. Data is opaque to Amazon S3. Metadata stores information about that object in name-value pairs. It is identified in a bucket by its key and version ID.
- Key - a unique identifier for an object within a bucket. Updates to keys are atomic.

Another feature is that a user can choose a geographical region where Amazon S3 will store the buckets. This is a vital selection, since on the region the latency, costs and address regulatory requirements might depend. The regions supported are following: US Standard, US-West (Northern California) and EU(Ireland). Amazon Web Services are not free and Amazon S3 service has its fees. Notwithstanding, it is designed in such a way, that a developer does not have to plan the storage requirements for an application. Amazon S3 charges only for the storage which is actually in use. This variable-cost service is a very good solution since it enable seamless growth in storage requirements.

2.2.3 Amazon Simple Queue Service (Amazon SQS)

As it is specified in [21] and in [8] Amazon SQS is a queue-based, message oriented middleware (MOM) for processing unlimited number of messages. Each message can have unlimited senders and receivers. It is a distributed queue system enabling fast communication between components of the application. Queue serves as a repository for messages which are waiting for processing. That queue can be perceived as a buffer between the component producing and saving data and the one receiving this data for processing. It can be a very useful tool in cases when the data producer is working faster than the receiver or in cases when both parties are not simultaneously connected to the Internet.

Additionally, SQS is a trustworthy solution, since it guarantees the delivery of each message at least once. Single queue can be used by many distributed application components and what is more those components do not need to coordinate in order to effectively use same queue.

Another Amazon SQS features are following. It is possible to configure queue settings per queue, thus each queue can be different. Message size is variable and can be up to 8KB. Very interesting solution is proposed if a message size exceeds 8KB. It is suggested by Amazon that Amazon S3 or Amazon SimpleDB can be used in such a case for storing the message and use Amazon SQS to hold the pointer to the object stored in one of those two services. Access control is also available since one can specify who can send messages to queue and who can retrieve them.

Exemplary usage may be following. One may decide to send the bucket name and object identifier of files in Amazon S3. This sending might be to hundreds of computers which will process those files in parallel. Thanks to the fact that the queue locks each message until its deletion or time out, one computer will process only one given file pointed to by this message.

2.2.4 Amazon Elastic Compute Cloud (Amazon EC2)

This web service provides resizable compute capacity in the cloud. It hosts the Linux and OpenSolaris based server-side applications. They can be managed by Amazon EC2 SOAP or REST web services. This service is supported by massive Amazon datacenters, which infrastructure is also used to support billions of transactions each year. Another interesting feature is that this service enable given developer's application to scale up as needed [22]. This service has a number of following desirable characteristics, according to [22]:

- Elastic - thanks to this one may rent or commission one, hundreds or even thousands of server instances in minutes. In scope of the framework, another key feature is that there is API enabling given application to automatically scale the number of server instances up or down.
- Completely controlled - each instance is given to the developer with root access. Instances can be rebooted remotely.
- Flexible - there are available multiple types of instances, operating systems and software packages making EC2 very flexible solution.

- Amazon EC2 can be used with other Amazon Web Services easily.
- Reliable - it is another vital feature, since the calculations which will proceed after implementation of the optimization framework, will take several days or even weeks to complete and in case of permanent failure of considerable number of instances the completion of calculations would be highly endangered. According to Amazon EC2 Service Level Agreement commitment is 99.95 % availability for each Amazon EC2 Region.
- Secure - Amazon EC2 provides developers with a number of security solutions including firewall settings, launching of Amazon EC2 resources within Amazon Virtual Private Cloud [23],
- Inexpensive - this is incarnation of economics of scale followed by Amazon. Thanks to this characteristic even a student with his limited budget can complete ambitious undertakings.

The characteristics mentioned above made the selection of AWS for realization of the thesis a good decision. Moreover there is a number of services, features provided in scope of Amazon EC2 service. The most interesting are as follows, based on [22]:

- Amazon CloudWatch - it is a web service providing monitoring for AWS cloud resources. It gives a great insight into resource utilization, operational performance and overall demand patterns.
- Auto Scaling - this feature makes it able to scale Amazon EC2 capacity up or down according to developer defined conditions. Because of that solution, there is no threat that rented instances will remain unused, but the money would be charged. Simply they will be decommissioned.
- Elastic Load Balancing - it is yet another helpful feature which automatically manages incoming application traffic across multiple Amazon EC2 instances. In an event of detection of unhealthy instance, traffic is rerouted to healthy instances until unhealthy one have been restored.
- Elastic IP Addresses - this feature enables to resolve quickly following situation. In case when an instance or Availability Zone failures, Elastic IP addresses allow the remapping of developer assigned public IP address to any instance in developer account.
- Amazon Elastic Block Store - it offers persistent storage for Amazon EC2 instances.

In scope of the implemented framework, the main need is for CPU power, rather than disk storage, so instances with strong CPU features were rented.

2.2.5 Amazon SimpleDB

This web service provides highly available, scalable and flexible non-relational data store [24]. Storing datasets and returning results is quick. Data is organized into domains. All data creation commands and queries are run against a particular domain. They contain

individual data items, which are described by attribute-value pairs. This service, as all described above, is scalable, highly available and flexible as well. It can be seamlessly used with another Amazon Web Services and it is inexpensive for the same reasons as the rest of the web services of Amazon.

Nevertheless, Amazon SimpleDB is not like relational database. It does not contain any schemas. Each item may contain up to 256 attributes. Because of that this web service domains are extremely flexible. This lack of schema is perceived as an advantage of Amazon SimpleDB held over traditional relational databases since it means that there is no need of predefining different data for items, but rather they can be changed as application requirements change. Moreover each item may contain multiple values unlike in relational databases. According to [24] it can be used to implement multiple states for same item. Finally, attribute value is always a string and these values are collated in lexicographical order.

2.2.6 AWS conclusions

All above mentioned Amazon Web Services seem to provide enough infrastructure and solutions in order to properly proceed with the goals of this thesis.

2.3 Monster Muck Mashup (MMM)

This solution is one of the main inspirations for doing evolutionary optimization with use of Amazon Web Services. MMM, implemented in Python programming language, shows many features of AWS such as scalability potential of web services. In the result concepts used in MMM can be utilized in the framework designed and implemented in Java programming language in scope of this thesis.

2.3.1 MMM architecture

Monster Muck Mashup architecture, following what is stated in [9], can be applied to many different application areas. However, MMM focuses on video format conversion from AVI to MPEG4. Additionally, the presented solution can be easily scaled to serve thousands of users. In order to achieve fast and reliable platform for performing this task, three Amazon Web Services are combined: Amazon Elastic Compute Cloud (EC2), Amazon Simple Storage Service and Amazon Simple Queue Service (SQS). As author of MMM states, such application could be constructed on a small server, however first problems would occur if someone else would also want to use MMM. There would be a problem of increase in user number, more CPU demands, greater hard disk space and bandwidth requirements. Thanks to AWS one may not be bothered by such problems, since web services already solved such issues, making it able to built it in inexpensive to operate way, easy to construct and able to scale almost indefinitely.

Figure 2 [2] presents the basic architecture of MMM. Python library named ”boto”, created by MMM author, is used to construct this application.

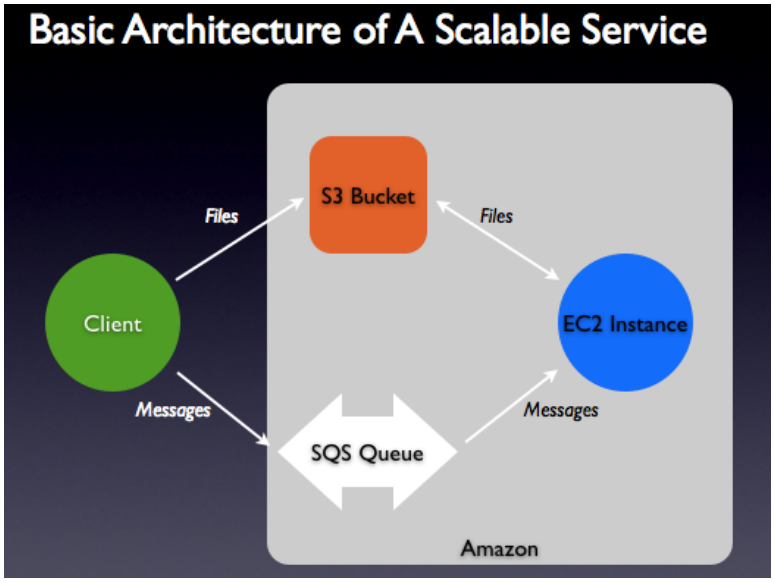

Figure 2: MMM architecture

Amazon S3 is used to store the video files to be converted and output files after conversion as well. Key feature is that there is no need to be worried if the disk space will be depleted as this service will always manage to scale up when necessary. Instructions are stored in SQS and the video conversion itself is made in EC2. The later provides elastic computing resources and EC2 instances can be created or decommissioned in case of need. The scalability and availability are another features which are also in use.

According to Garnaat, first thing to do is to create a new EC2 image called Amazon Machine Image. It contains all the software needed to create video conversion service. The details of the creation of such image are given in [25]. Next there are few basics steps taken by conversion service in order to perform its task. According to [9] they are following:

- read message from input queue,
- retrieval of input file from Amazon S3 and store in locally in EC2 instance,
- performing of video conversion, which results in production of one or more output files,
- storing generated output files in Amazon S3,
- writing a message to output queue which describes performed work,
- deletion of the input message from the input queue.

Details regarding Python code used in scope of MMM and ”boto” library are available in [9]. Nevertheless, the most important thing is overall architecture. In this respect one of the

[2] Image from [9]

features of EC2 comes handy. When it is started, it allows its user to pass arbitrary data to an instance. Thanks to this a variety of parameters can be passed to the service at instance creation time which allows to instruct it what it has to do.

2.3.2 MMM usage

First thing which is done to start the conversion process, however, is to submit a number of AVI files into Amazon S3 bucket. Then it is necessary to specify SQS queue which will be used as the input queue for the conversion service. When a file is stored in Amazon S3 bucket, a message is written to the specified queue being the information that a new file has been submitted for conversion. Since both components are prepared, namely files for conversion and messages, the very video conversion process may start. In such a case a number of EC2 instances can be created in order to process the messages in the input SQS queue. In first attempt only fifty video files for conversion were submitted and only one EC2 instance was started. In next attempt more servers were started and more video files submitted and here is the place were the beauty of AWS scalability shows itself. Table 1 shows the comparison between both cases.

Table 1: Comparison of two usages of MMM

No. of EC2 instances	No. of submitted files	Min processing time	Min processing time	Average processing time	Elapsed time	Throughput (transaction/minute)
1	50	2	58	17.82	896	3.348
10	500	2	60	17.79	928	32.327

As it can be seen from Table 1 there are ten times more EC2 instances and files for video conversion involved. Nevertheless, the conversion time is roughly the same thanks to the fact that whole video conversion service is so scalable.

Additionally, the costs of this undertaking are very small as promised by Amazon. Everything is summed in Table 2. According to Table 2, which is based on [9], the conclusion is

Table 2: Costs summary

Storage (S3)	2.5 GB	$0.38/Month
Transfer	2.5GB	$0.50
Messages (SQS)	1000	$0.10
Compute Resources (EC2)	EC2 instances for about 20 minutes	$0.80
	Total:	$1.78

stated that the cost of conversion of one video file was just $0.004 when dealing with 500 files. The author of MMM also wrote newer article about his solution [26]. It makes use of Amazon SimpleDB for logging status messages. To conclude it can be said that above mentioned solution with its architecture is an evidence that selecting Amazon Web Services for performing optimization tasks having high scalability demands is a good choice.

2.4 Java Agent Development framework (JADE)

This framework, as it name suggests, is designed for helping Java programmers in designing and implementing of multiagent systems. Since the designed thesis framework is in the need of simulating marketplaces populated with trading agents, JADE was taken under consideration.

2.4.1 The Foundation for Intelligent, Physical Agents (FIPA)

Nevertheless, it must be stated from what JADE emerged and what standards it follows to further reason this selection. The agency which devises and publishes the standard basis for multiagent framework is The Foundation for Intelligent, Physical Agents (FIPA) [27] organization was established in 1996 as an international non-profit association to develop a collection of standards relating to software agent technology. During its evolution many agent-related ideas have been developed. Those most important are agent communication, agent management and agent architecture [10].

Agents in a multi-agent system must communicate in order to achieve goals. According to FIPA a service-oriented model is used. It is a communication protocol stack with multiple sub-layer application protocols instead of single layer application protocol. The agent communication language is FIPA-ACL (Agent Communication Language). It is based on speech act theory which states that messages represent actions or communicative acts, also known as speech acts or performatives. FIPA-ACL has a set of 22 communicative acts based on the ARCOL proposal of France Telecom where every act is described using both a narrative form and a formal semantics based on modal logic [28] that specifies the effects of sending the message on the mental attitudes of the sender and receiver agents. This form of logic is consistent with the BDI or Belief, Desires, Intention reasoning model [29], [10]. Most common acts are inform, request, agree, not understood and refuse. Based on this, FIPA defined a set of interaction protocols. Each of them consists of a sequence of communicative acts to coordinate multi-message actions. Their examples are the contract net protocol or request protocol.

2.4.2 JADE basics

JADE itself is based on above-mentioned principles. Moreover it is the most well known and widespread agent-oriented middleware in use. It has a number of features which makes it even more usable. JADE is a completely distributed middleware system with a flexible infrastructure allowing easy extensions with add-on modules. This framework enables the user to develop a complete agent-based applications by means of a run-time environment implementing the life-cycle support features required by agents, the core logic of agents themselves, and a rich graphical suit. It is written in Java which makes it more flexible and easier to learn for person who previously worked with this programming language.

Current version of JADE provides its user with the possibility to easily introduce new functionality to the platform. From version 3.2, the JADE run-time structure stays in accordance with a new design know as the ”distributed coordinated filters architecture” [10].

This approach shapes the current JADE architecture which is built basing on following requirements [10]:

- Implementation of platform features as separated modules.
- Flexible support of the integration of a new features and modification of existing ones.
- Deploying the features across a distributed platform.
- The possibility to deploy only needed features.

According to FIPA, only the external behaviour of system components should be specified, leaving internal architecture and implementation details to the developers of individual platforms. Thanks to it, seamless interoperation between fully compliant platforms is possible. JADE fulfils this assumptions thanks to staying in accordance with primary FIPA2000 specifications(communication, management and architecture)[27] that provides the normative framework within which FIPA agents can exist, operate and communicate, while adopting a unique, proprietary internal architecture and implementation of key services and agents.

2.4.3 Agent management

Agent management is realised within normative framework within which agents can exist, work and be managed. It gives the logical reference model for the creation, registration, location, communication, migration and operation of agents [10]. It is depicted on Figure 3 [3].

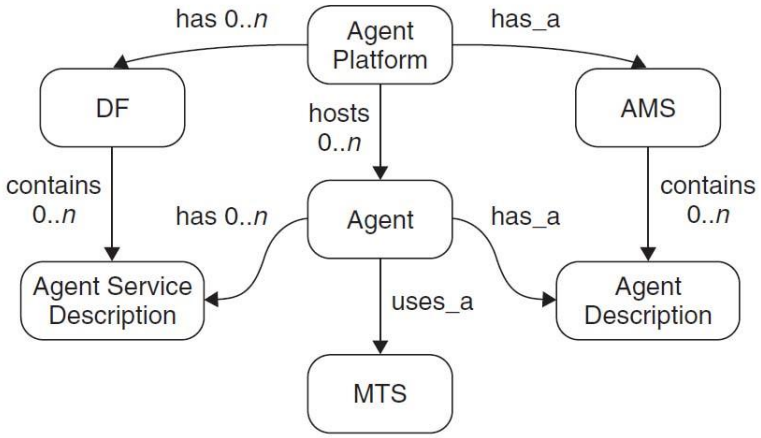

Figure 3: Agent management realization

- Agent Platform (AP): Gives the infrastructure in which agents are deployed. The AP constitutes from machines, operating systems, FIPA agent management components, the agents and any additional support software.

[3] Image from [10], page 15

- Agent: It is a computational process that inhabits an AP and offers one or more computational services that can be published as service description. An agent has at least one owner and must support at leas one notion of identity which can be described using FIPA Agent Identifier (AID) that labels an agent. Thanks to it the agent can be distinguished unambiguously.
- Directory Facilitator (DF): It is an optional component of AP. It provides yellow pages services to other agents. If an agent wants to offer its services, it has to find appropriate DF and request registration of its agent description. Moreover it can also request the deregistration of its services if such necessity arises. What is more an agent may request the DF to change its agent description. It is possible to the agent that it will issue a search request to DF in order to discover agent descriptions matching supplied search criteria.
- Agent Management System (AMS): It is an obligatory component of an AP and is responsible for managing the operation of an AP. The actions managed are creation and deletion of agents, and overseeing the migration of agents to and from the AP. It is required from every agent that it will register with an AMS in order to obtain an AID (Agent ID), which is then stored by the AMS as a directory of all agents present within the AP and their current state (active, suspended or waiting). Agent description within AMS can be modified. The life of an agent ceases its existence in AP when it deregisteres from the AMS. Agent descriptions can be searched in the AMS. It also keeps the AP description. AMS can request agent to terminate its execution. Only one AMS can exist in each AP and if AP spans multiple machines, the AMS is the authority across all those machines.
- Message Transport Service (MTS): It is a service provided by an AP to transport FIPA-ACL messages between agents being on any given AP and between agents on different APs. Message consists of transport envelope that constitutes the number of parameters specifying information such as to whom the message is to be sent.

2.4.4 JADE conclusions

Another important advantage is the fact that JADE implements the complete Agent Management specification and the key services of AMS, DF, MTS and ACC (Agent Communication Channel). This ensures the core compliance to FIPA specifications. What is more JADE also includes complete implementation of FIPAAgent Communication stack, making FIPA-ACL for message structure available, FIPA-SL for message content expression and additionally gives support for many of the FIPA interaction and transport protocols [10].

JADE also includes components which go far beyond FIPA specification, nevertheless making usage of this platform even better experience. It covers a distributed, fault tolerant, container architecture, internal service architecture, persistent message delivery, semantic framework, security mechanisms, agent mobility support, web-service interaction and graphical interfaces for multiagent system monitoring. Thanks to its open source status, strong support from industry and a broad user community, JADE is considered to be the leading FIPA-compliant open source agent framework.

2.5 Zero Intelligence Plus trading agents overview

Zero Intelligence Plus artificial trading agents were first introduced in [3]. They use just few rules and simple machine learning techniques and even though they outperform human traders in auction-market environments. They are the same as those used in experimental economic studies held by Smith in 1962 [30]. Those studies concerned the CDA among other auction mechanisms. First attempt to develop Zero Intelligence traders was made by Gode & Sunder's in 1993 [31]. Nevertheless it was not successful. Those traders only performed well in highly controlled auction environment and when some alterations were introduced, they failed to adapt quickly. According to Cliff it was main motivation to ([2], p.7)*"develop adaptive trading agents that could be used in distributed and decentralized market-based control systems (e.g., [32]), and also to extend the scope of research in adaptive behaviour to include the study of interactions among economic agents within market mechanisms [33]."*

As it was experimented by IBM [12], and stated above, ZIP and IBM's MGD trading agents outperform human traders. The setting for those experiments was following - there were human traders put against artificial traders in CDA marketplaces. IBM experiments showed that robot traders were consistently a few percents better than humans. What is even more interesting in scope of this thesis is the fact that ZIP had the joint-highest average efficiency amongst trading strategies compared in IBM experiments. For ZIP it was 1.030, for MGD it was 1.023 and for humans 0.876. Nevertheless when ZIP was put against MGD, the latter did better. Moreover there was another algorithm developed by [34] called GDX, based on MGD, which did better than MGD and ZIP during robot-vs-robot competitions. Another interesting thing is that MGD and ZIP did perform equally well during human-robot challenges.

According to Cliff, there were no order books used showing the price-ordered list of best offers and of best bids being quoted currently in the market. Such design resulted in following the Smith's [30] arrangements of reacting only to the most recent quote in the marketplace. In the result experimental markets are closer to foreign exchange than to equities. Moreover, as it is said in [2], it is easy to adopt ZIP to work in different markets such as asynchronous ordered-book-based ones. It was realized by [12] and [35]. Only minor extensions were necessary to be applied to ZIP.

In the first version the numerical parameters maintaining the work done by ZIP traders were set by hand. In next attempts ([36], [37]) another approach was used, an evolutionary computation technique was utilized. The selection was made to use "plain vanilla" genetic algorithm (GA) to perform automatic optimization of the parameter set.

2.5.1 Zero Intelligence Plus 8 trader

The main paper describing ZIP 8 is [3]. Following rules and parameters apply for ZIP trader. Each i^{th} agent is given:

- λ_i - private secret limit-price. For a seller it is the price below which it must not sell and for a buyer is the price above which it must not buy. Another notion is zero

utility. It occurs when transaction is completed at the price λ_i. Utility greater than zero is desired.

- $\mu_i(t)$ - time-varying utility margin. Each trader is given an initial value $\mu_i(0)$ which is altered to suit the market environment with help of the Widrow-Hoff rule being a learning technique. It is used in back-propagation neural networks [38] and in learning classifier systems [39]. $\mu_i(0) = U[\mu_{min}, \mu_{min} + \mu_\Delta]$.
- β_i - learning rate parameter. It maintains the speed of convergence between trader $p_i(t)$ and $\tau_i(t)$. $\beta_i = U[\beta_{min}, \beta_{min} + \beta_\Delta]$.
- $\tau_i(t)$ - idealized target price. It is calculated with additional small random absolute perturbation with magnitude generated from $U[0, c_a]$ and a small random relative perturbation generated from $U[1 - c_r, 1]$ (when decreasing $\tau_i(t)$) or $U[1, 1 + c_r]$ (when increasing $\tau_i(t)$).
- c_a and c_r - global system constants.
- $p_i(t)$ - quote-price, for seller by a formula $p_i(t) = \lambda_i(1 + \mu_i(t))$ and for buyer by $p_i(t) = \lambda_i(1 - \mu_i(t))$, aim is to maximize the utility over all trades. Utility is the difference between accepted $p_i(t)$ and λ_i.
- γ_i - momentum parameter for each trader. It is used to smooth over the noise in the learning system. It is commonly used in back-propagation neural networks. $\gamma_i = U[\gamma_{min}, \gamma_{min} + \gamma_\Delta]$.

Pseudocode of ZIP Algorithm [2], is presented below in Algorithm 1 and Algorithm 2. It shows how qualitative margin heuristics works. They can be perceived as a simple decision tree with terminal nodes. Those nodes would then specify if the trader should raise or lower its margin or leave it unchanged. However if a margin is to be changed, a set of quantitative steps is specified also and presented later [2]. Most detailed description of ZIP8 is presented in [3].

In order to initialize whole ZIP-trader market it is vital to have values for: μ_{min}, μ_Δ, β_{min}, β_Δ, γ_{min}, γ_Δ, c_a and c_r. Such 8-parameters vectors can be considered as genotypes in a GA. As a result it is possible to find a set of genotype vectors which best satisfy an appropriate evaluation function.

2.5.2 ZIP8 Quantitative Margin Adjustments

In order for the ZIP8 trader to proceed, there is a number of quantitative margin adjustments as they are listed below ([2], p. 41):

- $p_i(t) = \lambda_i \cdot (1 + \mu_i(t))$ - function for calculating current quote-price at time t
- for sellers: $\mu_i(t) \in [0, 8] \forall t$, raising a margin of a seller results in increasing of $\mu_i(t)$, it is decreased to lower the margin.
- for buyers: $\mu_i(t) \in [-1, 0] \forall t$, raising a margin of a buyer increases the (negative) magnitude of $\mu_i(t)$, it is then closer to -1.0. When the margin is lowered it results in reduction of magnitude making it being closer to 0.

Algorithm 1 Pseudocode presenting ZIP8 qualitative margin heuristics for sellers

```
1: if q.accepted == true then
2:     for all sellers s_i do
3:         if s_i.p = q.price then
4:             raise(s_i.μ)
5:         end if
6:     end for
7:     if q.type == "bid" then
8:         for all active sellers s_i do
9:             if s_i.p = q.price then
10:                lower(s_i.μ)
11:            end if
12:        end for
13:    end if
14: else
15:    if q.type == "offer" then
16:        for all active sellers s_i do
17:            if s_i.p = q.price then
18:                lower(s_i.μ)
19:            end if
20:        end for
21:    end if
22: end if
```

Algorithm 2 Pseudocode presenting ZIP8 qualitative margin heuristics for buyers

```
1: if q.accepted == true then
2:     for all buyers b_i do
3:         if b_i.p = q.price then
4:             raise(b_i.μ)
5:         end if
6:     end for
7:     if q.type == "offer" then
8:         for all active buyers b_i do
9:             if b_i.p = q.price then
10:                lower(b_i.μ)
11:            end if
12:        end for
13:    end if
14: else
15:    if q.type == "bid" then
16:        for all active buyers b_i do
17:            if b_i.p = q.price then
18:                lower(b_i.μ)
19:            end if
20:        end for
21:    end if
22: end if
```

Moreover then the learning rule "delta" of the discrete-time Widrow-Hoff for adjusting of the output A with respect to desired output D is used. It is asymptotically approached at a rate determined by the learning rate parameter β:

- $A(t+1) = A(t) + \Delta(t)$
- $\Delta(t) = \beta \cdot (D(t) - A(t))$
- It is then modified to accommodate dumping factor $\gamma \in [0,1]$: $A(t+1) = \gamma \cdot A(t) + ((1-\gamma) \cdot \Delta(t))$

In case of ZIP $A(.)$ updated by the Delta Rule is the profit margin μ:

- $\mu_i(t+1) = (p_i(t) + \Delta_i(t))/\lambda_i - 1$
- then δ is defined relative to a desired τ: $\Delta_i(t) = \beta_i \cdot (\tau_i(t) - p_i(t))$;
  $\tau_i(t) = (A_i(t) + R_i(t) \cdot q(t))$.

In above equations $q(t)$ is the price of the last quote in the marketplace. $A(.)$ and $R(.)$ are stochastic absolute and relative perturbation functions involving uniform distributions bounded by the constants c_a and c_r.

When above equations are combined they result in the following:
$\mu_i(t+1) = (p_i(t) + \Gamma_i(t))/\lambda_i - 1$;
where $\Gamma_i(0) = 0$, and $\Gamma_i(t+1) = \gamma_i \cdot \Gamma_i(t) + ((1-\gamma_i) \cdot \Delta_i(t))$.

2.5.3 ZIP60

The motivation standing behind extending the number of parameters from eight to sixty is that some market dynamics might be better described, if trading agents would have more parameters which would enable description of such situations as it is presented in [2].

As a starting point following situation might be selected. It is possible that if buyers would use different set of parameters than sellers they would perform better. In such a case an optimizer would have to deal with sixteen parameters - eight for buyers and eight for sellers - ZIP8 becomes ZIP16.

Another four cases give the following. The situation might occur that traders being buyers and sellers have to increase and decrease their margins. It might be useful to have for each situation within each trading agent separate sets of eight parameter values. This will constitute ZIP32.

According to [3] there are three cases in which both buyers and sellers alter their margins. Seller margin is raised if the last quote was accepted and current price is less than the price of the current quote. Seller margin is lowered if:

- the last quote was an accepted bid, given seller is active one and its price is greater than the price of the last quote,
- the last quote was an offer that was accepted and the seller is active and its price is greater than the price of the last quote.

In above mentioned cases the number of parameter-value vectors for sellers and for buyers is three for each. In such situation there are six vectors with eight parameters each giving ZIP48.

According to Cliff, it is also worth to have for each trader its own values of c_a and c_r. In such case they are generated from the uniform distributions, $c_{a,i} = U[c_{a:min}, c_{a:min} + c_{a:\Delta}]$ and $c_{r,i} = U[c_{r:,min}, c_{r:min} + c_{r:\Delta}]$. The final vector for each of six cases mentioned above now have the following parameters: $\mu_{min}, \mu_\Delta, \beta_{min}, \beta_\Delta, \gamma_{min}, \gamma_\Delta, c_{a:min}, c_{a:\Delta}, c_{r:min}, c_{r:\Delta}$, so in total gives ZIP60.

Following Cliff in [2], consecutive naming is now introduced. Let P be one of $\mu, \beta, \gamma, c_a, c_r$, t one of min, Δ, n one of $1, , 6$. Then homologous set of P_t parameter values is the set of six values for any one parameter-type, for example $P_{t:1}$. In ZIP60 interpretation for the six cases in each homologous set is as follows ([2], 20): ”

- $P_{t:1}$*: value of* P_t *for the case when a Seller raises its margin.*
- $P_{t:2}$*: value of* P_t *for the case when a Seller lowers its margin and* $q.type = Bid$.
- $P_{t:3}$*: value of* P_t *for the case when a Seller lowers its margin and* $q.type = Offer$.
- $P_{t:4}$*: value of* P_t *for the case when a Buyer raises its margin.*
- $P_{t:5}$*: value of* P_t *for the case when a Buyer lowers its margin and* $q.type = Bid$.
- $P_{t:6}$*: value of* P_t *for the case when a Buyer lowers its margin and* $q.type = Offer$. ”

According to [2] additional computation costs when compared to ZIP8 in case of using ZIP60 are nearly zero. Increase of marketplace performance is notable, yet the overall cost of switching from ZIP8 to ZIP60 is negligible.

2.6 Adaptive Aggressive (AA) trading agent

2.6.1 Motivation behind AA

The motivation standing behind designing of Adaptive Aggressive trader, as it is stated by Vytelingum in [4], is that he wanted to develop more efficient strategies for CDA. According to the author of AA, more efficient trading strategies than ZIP can be developed and Adaptive Aggressive trading agents answers to such call. The main advantage of AA over others is that it assumes that market is an dynamic environment and other trading strategies perceive it as static one. As an argument to follow such way of thinking it is stated that NASDAQ and NYSE are very dynamic markets with frequent market shocks.

Moreover previously described agents were designed specifically for static markets, however they were said to be adaptable to work in dynamic markets as well. According to Vytelingum it was a wrong assumption, because there are major differences between static and dynamic marketplaces. First of all, the main difference regards the sporadically changing competitive market equilibrium. From that the conclusion was driven stating that different behaviours for trading agent are needed in two different cases, when market is stable

and when it is changing. When trading agent is dealing with static market, it can be effective due to assumption that the competitive equilibrium does not change much. On the other hand in dynamic case such assumption no longer holds and trading agent has to learn along with competitive equilibrium changes. Final motivation standing behind AA design was that AA should perform well in homogenous population of same trading agents and in heterogeneous population in which there are trading agents adopting different trading strategies.

2.6.2 AA algorithm

The author of AA focuses on bidding aggressiveness as a measure of successful trading agent in the market. In this respect an aggressive trader is placing bids which are not necessary most profitable. In such case another trading strategies would place bids with highest profit, nevertheless they are less likely in completing any transaction. The Definition 5.1 of Aggressiveness from ([4], p. 97) is following: "*Aggressiveness is defined as the inclination to interact more actively in the market. The aggressive trader submits better offers than what it believes the competitive equilibrium price to be, to improve its chance of transacting, and trades off profit for that purpose. The passive trader is less inclined to transact and more inclined to win a profitable transaction and thus submits offers that are worse than what it believes the competitive equilibrium price to be. The neutral trader submits offers at what it believes is the competitive equilibrium price, which is the expected transaction price.*". Aggression can be described by parameter $r \in [-1,1]$, where $r < 0$ means aggressive, $r = 0$ means neutral and $r > 0$ meaning passive. Level of aggression are defined through learning mechanism. Throughout [4] p^* denotes profitable transaction, $\hat{p}^*$ denotes more profitable transaction.

AA trader is composed of two key decision making elements: a bidding one making decision what bid or ask should be submitted, this is based on trader's current level of aggressiveness and a second one is a learning mechanism updating trader's behaviour according to the market. Aggressiveness model maps τ - target price, $\hat{p}^*$ and an intrinsic parameter θ. Second component represents the adaptive part of the strategy. It is composed from short-term and long-term learning mechanism which purpose is to update bidding behaviour of the trader. The first learning mechanism updates r and the second one updates θ.

In order to calculate equilibrium estimator $\hat{p}^*$ moving average method is used. In the equation presented below N represents a number of most recent transaction prices. $\hat{p}^* = \frac{\sum_{i=T-N+1}^{T} w_i p_i}{\sum_{i=T-N+1}^{T} w_i}$ where $w_T = 1$ and $w_{i-1} = \lambda w_i$ ([4], p. 100). In this equation $(w_{T-N+1}, ..., w_T)$ is the weight given to the N most recent transaction prices, $(p_{T-N+1}, ..., p_T)$, and T is the latest transaction. λ is set to 0.9 basing upon simulation and N to number of daily transactions.

2.6.3 Aggressiveness model

According to [4], agent can be of two types:

- intra-marginal - a buyer (seller) is of that type if its limit price is higher (lower) that the competitive equilibrium price,
- extra-marginal - a buyer (seller) is of that type if its limit price is lower (higher) than the competitive equilibrium price.

Following formulas hold for intra and extra marginal seller and buyer ([4], p. pp. 102 - 103):

- For intra-marginal buyer i the formula for aggressiveness model is:

$$\tau = \begin{cases} \hat{p}^*(1 - re^{\theta(r-1)}) & \text{if } r \in (-1,0) \\ (\ell_i - \hat{p}^*)(1-(r+1)e^{r\underline{\theta}}) + \hat{p}^* & \text{if } r \in (1,0) \end{cases}, \text{ where } \underline{\theta} = \frac{\hat{p}^* e^{-\theta}}{\ell_i - \hat{p}^*} - 1$$

- For intra-marginal seller j the formula for aggressiveness model is:

$$\tau = \begin{cases} \hat{p}^* + (p_{max} - \hat{p}^*)re^{(r-1)\theta} & \text{if } r \in (-1,0) \\ \hat{p}^* + (\hat{p}^* - c_j)re^{(r+1)\underline{\theta}} & \text{if } r \in (1,0) \end{cases}, \text{ where } \underline{\theta} = log\left[\frac{p_{max} - \hat{p}^*}{\hat{p}^* - c_j}\right] - \theta$$

- For extra-marginal buyer i the formula for aggressiveness model is:

$$\tau = \begin{cases} \ell_i(1 - re^{\theta(r-1)}) & \text{if } r \in (-1,0) \\ \ell_i & \text{if } r \in (1,0) \end{cases}$$

- For extra-marginal seller j the formula for aggressiveness model is:

$$\tau = \begin{cases} c_j + (p_{max} - c_j)re^{(r-1)\theta} & \text{if } r \in (-1,0) \\ c_j & \text{if } r \in (1,0) \end{cases}$$

2.6.4 Adaptive component

Short-term mechanism is composed of the rules which are making decisions about how to update aggressiveness of a trader every time a bid or an ask is submitted or a transaction occurs in the market. As it was designed for ZIP trading strategy, in AA Widrow-Hoff algorithm is used to increase or decrease the aggressiveness $r(t)$ at time step t. It is presented in the following formula ([4], p. 105):

$$\begin{aligned} r(t+1) &= r(t) + \beta_1(\delta(t) - r(t)) \\ \delta(t) &= (1+\lambda)r_{shout}, \lambda = \{-0.05, 0.05\} \end{aligned}$$

In above equations $\delta(t)$ is current desired aggressiveness and β is learning rate parameter. Moreover following short-term learning rules apply for buyer and seller and they are similar to those used in ZIP and they are presented in Figure 5.4 of [4].

Long-term learning influences on a long-term basis the θ parameter. It can be influenced after every transaction in order to improve efficiency of AA. According to [4], θ depends

on the price volatility. Formula presented below, describes the learning mechanism of θ ([4], p. 106):

$$\theta(t+1) = \theta(t) + \beta_2(\theta^*(\alpha) - \theta_t)$$
$$\alpha = \frac{\sqrt{\frac{1}{N}\sum_{i=T-N+1}^{T}(p_i - \hat{p}^*)^2}}{\hat{p}^*}$$

Moreover, following formula approximates well the optimal θ parameter ([4], p. 106):

$$\theta^*(\alpha) = (\theta_{max} - \theta_{min})(1 - (\alpha - \alpha_{min})/(\alpha_{max} - \alpha_{min}))e^{2((\alpha-\alpha_{min})/(\alpha_{max}-\alpha_{min})-1)} + \theta_{min}$$

In above mentioned formula $[\theta_{min}, \theta_{max}]$ is the range over θ, α_{max} is updated, α_{max} is the maximum α that occurs in the market, and α_{min} is the minimum α.

One last component is bidding. A set of bidding rules is applied to decide is submission of a bid or an ask is reasonable thing to do and what should be the price of such submitted bid or ask. Those rules are presented in Figure 5.7 of [4].

Basing on above description, it can be stated that the following parameters will be optimized by the GA: β, N, η, λ, θ_{min}, θ_{max},

2.7 Optimization method which was used

In [2] EDA is suggested to be used in further research about Zero Intelligence Plus traders. It is of course a good idea nevertheless there must be a reason for selecting GA (described in [40] and [41]) used in earlier experiments described in [36], [37], [42], [43], [44], [45] and [46]. The reason is quite simple. Since GA was used before, for the sake of comparing results with previously obtained ones, now same optimization algorithm was used likewise.

According to experiment design by Cliff, all vectors constituting a evaluated population were assigned a fitness value. Population of next generation was generated through mutation and crossover on parents identified using rank-based tournament selection. Elitism was also used.

As it was stated in [2] genome of each population member was a vector of real values. In case of current experiment it were vectors of each trading agent being optimized. Each vector was additionally extended with Q_s parameter.

Moreover, as it was used in previous experiment described in [2] and was specified for this experiment, following statements hold:

- initial population is created by generating random real values from $U[0,1]$,
- crossover points are between numeric values,
- mutation is implemented by adding random values from $U[-m(g), +m(g)]$ where $m(g)$ is the mutation limit at generation g,
- mutation is applied to each real locus in each genotype on each individual generated from a reproduction event, $m(g)$ is reduced by by an exponential-decay annealing

function: $log_{10}(m(g)) = log_{10}(m_s) - ((g/999 \cdot log_{10}(m_s/m_e))$. It was used in the GA in [36], [37], [42], $m_e = 0.05$ - start mutation limit, $m_e = 0.0005$ - end mutation limit.

- if a value at a locus fall outside the $[0.0, 1.0]$ range, it was clipped,
- genome values of $\mu_\Delta, \beta_\Delta, \gamma_\Delta$ are clipped to satisfy constrains $(\mu_{min} + \mu_\Delta) \leq 1.0, (\beta_{min} + \beta_\Delta) \leq 1.0, (\gamma_{min} + \gamma_\Delta \leq 1.0)$,
- fitness will be calculated as it was realized in [36], [37], [42], [43], [44], [45], [46] and [2].

Each trial include a following sequence of events, this trail regards a given genome:

- it is a genome for initializing a ZIP-trader or AA-trader marketplace,
- ZIP or AA traders will operate for a given number of fixed trading periods,
- after each trading period stock and currency are replenished,
- during each trading period Smith's [30] α measure is monitored and it is used to calculated weighted average of α across trading period in the trial,
- fitness value is then calculated for each individual as the arithmetic mean of 100 such trials,
- individuals with lower scores are more likely to have greater reproductive fitness.

2.8 Past and present experiments

In each experiment performed by Cliff following marketplace setting were used as they are depicted in Figure 4[4].

Markets depicted in Figure 4 were used in evaluation process of ZIP8 and ZIP60. Such procedure is common in human-based experimental economics as it is described in [30]. They were also used in same manner in the experiments being in scope of this thesis.

Those markets were used in 18 dual-shock (triple-schedule) experiments, being trading periods, with following arrangements, each having following form: M121, M212, M232, M323, M123, M321 and so on. Each market, for example in M121 experiment, is used for six trading periods and so whole M121 experiment lasts for 18 trading periods. The results of this experiments made on ZIP8 and ZIP60 are presented in [2] along with their analysis. What regards AA it is evaluated in [4] and its efficiency as well . Also same shock market arrangements were used and the results of the experiments are also presented in [4].

2.9 Watchmaker

Thanks to this open-source framework it is possible to implement Genetic Algorithm in Java, hence making whole implementation platform independent on operating system [49]. Moreover it is extensible and assures high-performance, which makes it a good

[4]Image from [2], page 14

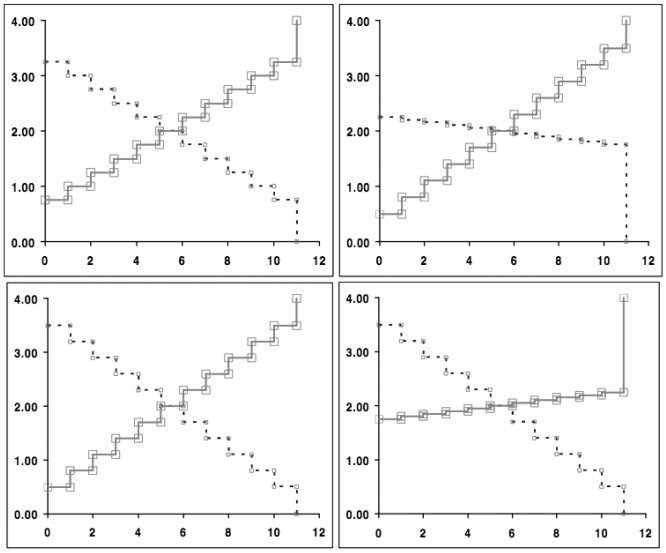

Figure 4: Different marketplaces, M1 - top left, M2 - top right, M3 - bottom left, M4 - bottom right. Supply curve - solid line, demand curve - broken line

selection for the purposes being in scope of this thesis. Moreover, it has a number of vital features. Watchmaker framework can work effectively on multi-core and multi-processor machines. One may choose any selection strategy he or she wants from those included within described framework or implement their own. There is also a possibility to choose different models of evolution depending on what is being optimized. Finally, the most important feature of all - possibility to distribute the evaluations of fitnesses using Hadoop via Apache Mahout project. This was a decisive factor which convinced the author to choose Watchmaker framework to implement Genetic Algorithm used in optimization of trading strategies.

When programming GA with this framework, it is necessary to know that the core component is so called Evolution Engine. Among its arguments Candidate Factory, Mutation, Crossover and method of selecting candidates for breeding have to be specified. Following paragraphs describe them in more detail.

2.9.1 Candidate Factory, Crossover and Mutation

In order to start any optimization some initial population has to be created. To achieve this, the programmer has to implement a class extending AbstractCandidateFactory class and code the function responsible for generating random candidate. This initial population is then used in first generation of evolution.

Nevertheless it is also important to perform breeding between most fit candidates from one generation to another. It is realised by implementing class responsible for crossover, which extends AbstractCrossover class. This mechanism works as follows. In order to gave birth to their offspring, parents have to be an arrays of parameters. Next some crossover points have to be selected. Then some genes - parameters from first parent and some parameters

from the second parent are being copied to two of their offspring as in real life. Of course both children obtained different parameters in the process.

Moreover mutation to genome of each child is introduced. According to specified function every parameter is slightly altered. Thanks to a class implementing EvolutionaryOperator interface, situation described below is avoided. If there would be no mutation and crossover being only breeding mechanism, the resulting children genomes would be just a mix of initial population genomes and far less candidate solutions could be tested in such a case, which can lead to missing possible best candidate with the best fitness.

2.9.2 Selection mechanisms

One must specify the way in which parents for breeding next generation will be selected as well. As it was written earlier, different types of selection of best parents can be used. One of those methods is rank selection. It simply selects candidates for breeding basing on selection probability which is proportional to relative fitness rather than absolute fitness. Advantage of such approach is that it will tend to avoid premature convergence by tempering selection pressure for large fitness differentials that occur in early generations. In later generations the selection pressure is increased. Moreover, it can be combined with other selection strategies such as tournament selection. Its working principle includes picking two individuals from the population of a given generation. Then a random number grater than or equal to zero and smaller than or equal to one is generated. If this value is smaller than or equal to the selection probability, which is a parameter specified by the programmer, the fitter candidate is selected. In other case weaker candidate gets selected. Thanks to the possibility to specify mentioned probability parameter, the selection pressure can be adjusted, nevertheless it is always desirable to set its value to be greater than 0.5 in order to facilitate that fitter candidates will be selected more often.

Yet another thing, which has to be programmed is fitness evaluation of each candidate. Since it is the most CPU-intensive part of GA, it is of utmost importance to code this part in as optimal way as possible. It is realized by implementing FitnessEvaluator interface. Function responsible for fitness evaluation simulates a system loaded with candidate solution parameters and evaluates its performance. This is a measure of how fit is a given candidate - it is its fitness.

2.9.3 Termination conditions

It is also necessary to specify some termination conditions for GA to stop. If not the program would be performing optimization forever, thus wasting time and money. As a termination condition reaching specified value of fitness for best candidate or stopping whole evolution on arbitrarily chosen generation can be set. It is very important to specify such a values carefully, because even if they would be set, reaching given fitness could not be possible while optimizing given system.

It is worth noticing that preserving unchanged number of candidates from one generation to another may significantly improve the overall performance of Genetic Algorithm. If elitism

is used, it is an assurance that even if breeding of new generation would lead to producing very bad candidate solutions, at least one best candidate solution from previous generation is secured, hence the best solution is kept intact from one generation to another. It is only replaced if new, better candidate was produced.

Whole process of initializing population, evaluating fitness, selecting candidates for breeding and performing offspring production is summarized in Figure 5 presented below.

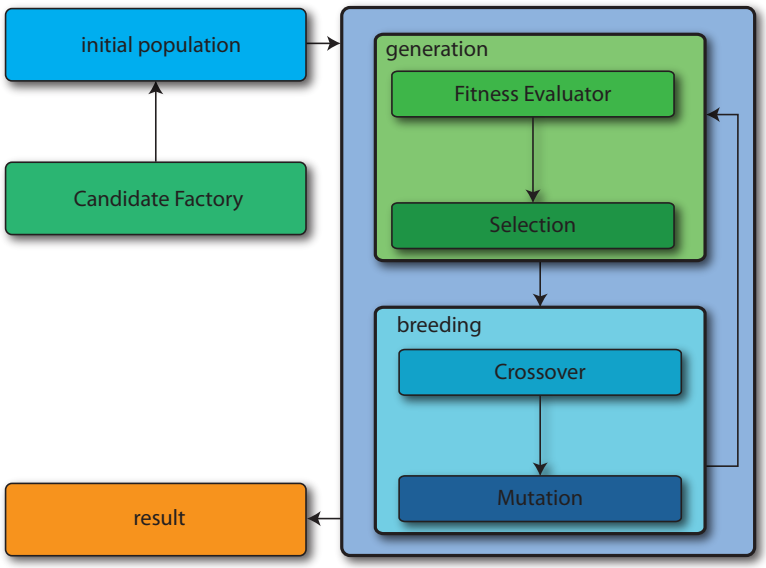

Figure 5: Watchmaker framework overview

2.10 Mahout

Framework being described now is a set of scalable machine learning libraries [50]. Core algorithms realized in scope of this open-source project are implemented on top of Apache Hadoop using the map/reduce paradigm. Nevertheless, the most important factor answering why it was considered to be used in this thesis is that it enables distributed fitness evaluations developed with Watchmaker to be calculated on Hadoop cluster. Moreover, it is possible to program an Amazon Machine Image which enables usage of Mahout and Hadoop frameworks as well which is yet another advantage.

The difference between Watchmaker based GA optimization performed locally and the one performed in distributed manner is the usage of different Fitness Evaluator. In latter case one must use Mahout Fitness Evaluator. The Fitness Evaluator normally used by Watchmaker is wrapped in Mahout one. Then, when fitness of each candidate from each generation needs to be calculated, Mahout creates a job for Hadoop containing all those calculations. What follows is distributing this task over a Hadoop cluster. Finally, properly prepared Amazon Machine Image is necessary, which will be loaded into EC2 instances used in Hadoop cluster.

2.11 Hadoop

This open-source project, including a number of subprojects, enables usage of clusters of servers to be utilized in order to perform given distributed calculations in reliable and scalable way [51]. It is suitable to work with thousands of nodes and to process petabytes of data. Moreover it was inspired by papers by Google about Google's MapReduce and Google File System (GFS). It is widely used by such renown companies as Yahoo!, Amazon, Adobe or IBM [52].

2.11.1 Filesystems

Hadoop Common, included in Hadoop, provides access to the filesystems supported by Hadoop. What is pointed out and what regards those filesystems is that they should provide location awareness for effective scheduling of work. Thanks to that Hadoop applications, while using this information, may run work on the node where the data is, hence reducing backbone network traffic. The list of supported filesystems include:

- HDFS - Hadoop Distributed FileSystem. It is designed for being able to scale to petabytes of storage and to run on top of other filesystems. Moreover it replicates data being processed and tries to keep different copies of the data on different racks, so in case of failure the data is preserved.
- Amazon S3 filesystem. No rack-awareness is available in it since everything is remote.
- CloudStore being rack-aware.
- FTP filesystem
- Read-only HTTP and HTTPS file systems.

The first mentioned filesystem stores large files across multiple machines. It is build from a cluster of data nodes. They are serving blocks of data over the network using a block protocol specific to HDFS. Another interesting feature is that they also serve data over HTTP, which allows access to all content from a web browser. It is very useful especially during software development phase where each modification has to be checked against the time needed to complete given calculation and thanks to this feature it can be done easily. It is also possible for data nodes to communicate in order to rebalance data, to move copies around and to keep data replication high.

Moreover there exists also so called name node in a Hadoop cluster. It is a one unique server and a single point of failure for HDFS as well. There is also a secondary name node which builds a snapshots of the primary name node directory information. Thanks to such operations it is possible to restart a failed primary name node without having to replay the entire journal of filesystem actions.

2.11.2 MapReduce

The MapReduce engine works above the filesystem. Applications submit MapReduce jobs to master Job Tracker which is consisted in it. The Job Tracker then pushes work to available Task Tracker nodes in the cluster. It is realised in such a manner to keep work as close to data as possible. If working in rack-aware filesystem, it knows which exact node contains the data. Moreover if it fails or times out the part of the job is rescheduled.

Such a slave Task Tracker consists of available slots for map and reduce tasks. Each task takes up one slot. What is more, when one Task Tracker is very slow it can delay completion of whole job, which is most unwanted thing to happen. Nevertheless, when speculative-execution is enabled, a single task can be executed on multiple slave nodes.

What regards MapReduce job, it splits the input-data set into independent chunks which are processed by the map tasks in a completely parallel manner. Then the outputs of the maps are sorted and form an input for the reduce tasks. Input and output of the job are stored in a filesystem.

When setting up a Hadoop cluster one must be aware of what kind of job it will be feed with. [53] provides a very good guidelines for specifying many important parameters. Figure 6[5] presents overview of how multi-node cluster looks like.

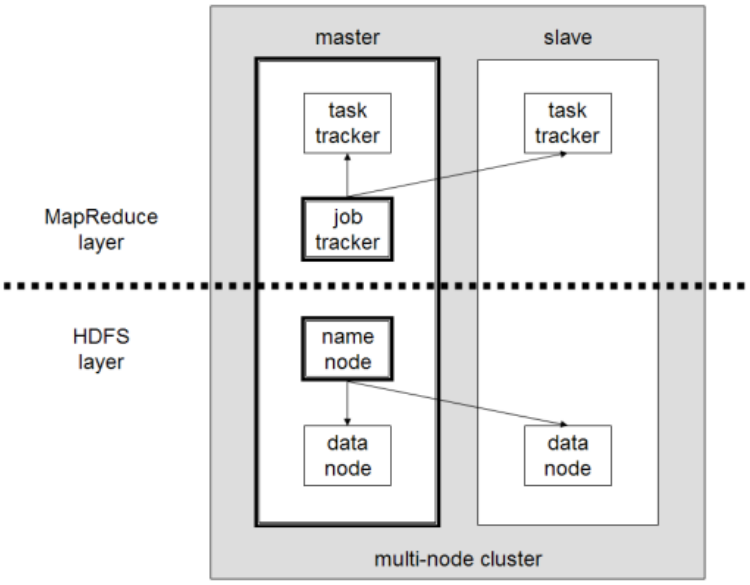

Figure 6: Multi-node cluster overview

[5]Image from http://www.michael-noll.com/wiki/File:Hadoop-multi-node-cluster-overview.png

3 Optimization Framework design and implementation

This chapter describes how the optimization framework was designed and implemented. Section 3.1 illustrates how proper AMI which is to be used within Hadoop cluster was constructed and which EC2 instance was used. Next section 3.2 describes why Hadoop cluster settings where given certain values and why certain number of nodes within this cluster was used. Following section 3 is devoted to explaining how the optimization framework was constructed from programming point of view and why. Each section also describes and gives solutions to problems which were faced during construction of the framework.

3.1 Amazon Machine Image and Elastic Compute Cloud instance

3.1.1 Basic AMI selection

Selecting proper AMI is the first decision which has to be made in order to start an EC2 instance. There exist some which are just basic versions of operating systems such as Linux or Windows, others have more advanced features built in. Nevertheless, those images proposed by Amazon or by other developers have to be properly altered to make them useful within an instance utilized in Hadoop cluster operating with Mahout framework. This resulted in a search for a guide or tutorial which would cover such issue and resulted in finding of the following web site [47].

This tutorial [47] suggests starting the building of an AMI with Claudera Ubuntu 64 bit version. Unfortunately, if someone wants to use 64 bit system, he or she may use only expensive multi-core EC2 instances since only on this type of instance such system can be loaded. Moreover, in first stages of the optimization framework development, especially during AMI programming phase, it was quite obvious that many start-ups and shutdowns of such an instance would occur. If those expensive ones would be used, it could result in significant costs, since starting up a new instance results in charging for whole instance hour. Due to this fact, the decision was made to use Claudera Ubuntu 32 bit version. Table 3 presented below compares the prices of usage of 32 bit and 64 bit system versions on given instances which were considered to be used.

Table 3: Instance price per hour comparison

System type	**Standard Instances**			**High-CPU Instances**	
	Small	*Large*	*Extra large*	*Medium*	*Extra large*
32 bit	0.085 $			$0.17	
64 bit		$0.34	$0.68		$0.68

Since Standard Small Instance has the smallest price per instance hour as presented in Table 3, it seems that it is the best selection for AMI development phase.

3.1.2 Building AMI

Due to the fact that basic version of Claudera Ubuntu 32 bit has only API Amazon EC2 API Tools installed, it is necessary to add Mahout and Hadoop as well. First of all proper versions of both frameworks had to be selected. It has revealed from Mahout documentation that Mahout 0.3 and Hadoop 0.20.2 are compatible and what regards the first framework, it can work with Watchmaker 0.7.1 as well. Moreover some programs, such as Ant, Subversion and Maven2 are useful while installing Mahout and because of that were added to an AMI which was developed.

After completing installation of Mahout and Hadoop, it is necessary to test if everything is working correctly. Due to this fact Mahout includes some exemplary programs which can be used for this purpose. Of course, in order to perform this action, Hadoop must be temporarily set up to be used as pseudo-distributed configuration as specified in [48]. After successful completion of those tests all configurations made before for Hadoop were removed. Newly created AMI has to be bundled, then sent to S3 and finally registered so it can be used on any 32 bit EC2 Instance.

3.1.3 Selecting proper EC2 instance

This step was realized when first version of the optimization framework was produced. The main problem which had to be dealt with was which type of EC2 instance should be used in Hadoop cluster. The main decisive factors were running time of given calculations and price of instance hour. Throughout whole development phase as a benchmark for assessing the efficiency of a given instance a time needed for performing calculations for 5 generations in optimization was used. As a reference point benchmarking was done for Standard Small Instance. Since the developed version of a system was 32 bit one, only one more instance could be used, which is High-CPU Medium Instance. Table 4 presents results of tests on both types of instances.

Table 4: Comparison of calculation times

Instance Type	**Time to calculate one generation (s)**	**Instance hour cost**
Standard Small Instance	194	0.085 $
High-CPU Medium Instance	64	0.17 $

One may conclude that it is more economic to use the High-CPU Medium Instance (5 GHz double core processor), since it is only two times more expensive than Standard Small Instance (1,2 GHz processor), but it performs approximately three times better. Because of that whole Hadoop cluster is cost-effective. Other instances were rejected, since they focused on high memory and hard disk space such as High-Memory Instances and Standard Instances.

3.2 Hadoop cluster setup

In order to have a working Hadoop cluster one must follow certain path. The first step on this path necessitates the familiarization with Hadoop EC2 software provided within Hadoop package. It includes programs which enable EC2 instance manipulations, starting them with proper settings and terminating whole cluster as well. The file storing Hadoop cluster settings is hadoop-ec2-init-remote.sh.

There exist many presentations and tutorials such as [54],[55] and [56] which give some guidelines saying how proper Hadoop cluster should be set up. Nevertheless, all of them are focusing on settings useful for clusters processing large amounts of data with low CPU usage. On the other hand, in the optimization framework situation is different, since GA optimization itself is a very CPU intensive task, but it does not require dealing with vast data at all.

All those guidelines have something in common - they focus on most important value, the number of map and reduce task slots per one Task Tracker. It is necessary to understand that each slot is effectively a new thread started on a processor of underlying EC2 instance. Another key factor is a number of map and reduce tasks per given Hadoop job. Moreover, Java heap size is also important since Hadoop jobs prepared by Mahout require considerable amount of RAM. While experimenting with different settings, also those proposed by Amazon Web Services were tested as well. It is suggested to use four slots for map tasks and two slots for reduce tasks per Task Tracker operating on High-CPU Medium Instance.

Selecting the proper number of nodes is also a key decision. As stated in above mentioned guidelines, it is just enough to add one more node to a cluster and one would easily divide the calculation time by two. In such a case a calculation time would be lowered with addition of every node in a half-life fashion. Unfortunately, it is only the case when performing analysis requiring small CPU consumption on large datasets and GA optimization works in quite an opposite way. Due to this fact adding new nodes to the cluster was not always a good idea and after reaching certain node quantity, no improvement in calculation time was observed. Results of experimenting with different settings and number of nodes in the Hadoop cluster are presented in Table 5. Previously described benchmark of five generations was used. As a comparison calculation time for MacBook having 2,4 GHz Intel Core 2 Duo processor was presented. Population of 30 candidates, 1 elite candidate, 5 generations and 100 market simulations for each candidate was used. In each test High-CPU Medium instances were utilized.

Each second gained in running time of 5 generations is important. When extended to 500 generations, each second gained when checking calculation time for 5 generations results in reduction of whole experiment time for around 1.5 minutes.

3.2.1 Proper settings

As it can be seen from Table 5 the best ratio of map to reduce task slots is two to one. Moreover, since one double core processor is in High-CPU Medium Instance, it is the best to set map task slots to two and reduce task slots to one. When one slot for each task type

Table 5: Different settings tests results

no. of intances	time taken()	map task slots	reduce task slots	Java heap size
1	325	4	2	725
2	240	8	8	725
2	229	4	4	725
2	225	4	4	725
2	229	8	4	725
2	242	8	8	725
2	254	8	8	725
4	195	8	8	725
4	198	8	8	725
4	182	8	8	725
6	171	12	12	725
6	173	12	12	725
6	176	12	12	725
6	170	12	12	725
8	176	8	8	725
8	172	8	8	725
8	171	16	8	2000
8	165	16	16	2000
8	193	32	16	725
8	181	32	16	725
8	166	16	8	2000
8	165	16	8	2000
8	174	8	8	725
8	170	16	8	2000
8	175	16	16	725
8	168	16	16	2000
8	167	16	16	725
8	170	16	16	2000
8	165	16	16	725
9	149	16	8	2000
9	151	16	8	2000
9	138	16	8	2000
9	145	16	8	2000
9	166	16	16	725
9	160	16	8	2000
9	162	16	16	725
9	167	16	8	2000
9	148	16	8	2000
9	164	16	8	2000
10	162	16	8	2000
10	165	16	8	2000
10	161	16	8	2000
15	153	30	15	512
15	154	30	15	512
15	153	30	15	2000
15	156	30	15	2000
15	161	30	30	725
15	165	30	30	725
15	163	30	15	2000
16	155	30	15	2000
16	149	30	15	2000
16	144	30	15	512
16	147	30	15	512
16	152	30	15	2000
16	152	30	15	2000

was used, the performance was slightly worse. Moreover, different settings for Java heap size were tested. It has revealed that, since memory consumption of GA optimization is low, they have not any impact on cluster performance.

Moreover number of nodes was also found. Cluster performance with different number of nodes is presented in Figure 7.

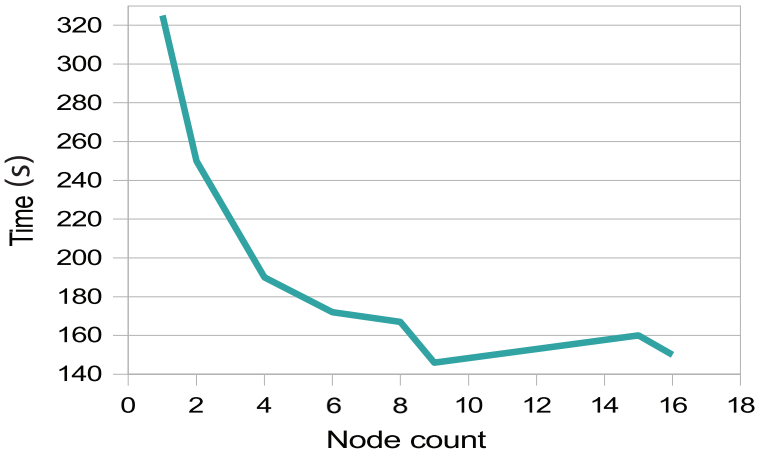

Figure 7: Node count in the cluster

A very interesting thing may be observed in Figure 7. Cluster efficiency is increasing when number of nodes is gradually raised from one to nine. Nevertheless, when testing the cluster with higher number of nodes, it is clear that there is no difference in calculation time. Probable reason for this state is due to Hadoop framework design. Since the process of mapping and reducing also requires time and with addition of each node the capacity of map/reduce tasks increase, the overall calculation time decrease is used for maintaining additional maps and reduces. On the other hand, when more market simulations for each candidate would be used, maybe such a larger cluster would have same calculation time as for lower number of market simulations per candidate and while using smaller cluster an increase in calculation time could be noticed. Nevertheless, this problem is not further investigated and can be realised in scope of a further work.

Another interesting finding was that when there were more map/reduce task slots than a given Hadoop job required, the overall performance of the cluster increased. It can be justified due to the fact that in papers mentioned above it is stated that Namenode thread residing on master node should have one free CPU core to operate with higher efficiency.

Having constructed the Hadoop cluster, one may come to a conclusion that such cluster with given efficiency is not enough. Adding new nodes to this cluster is pointless, since it is not decreasing a calculation time. In order to achieve better results one may simply start more than one cluster. In such a case, after given time, instead of having one optimization result, one would obtain several results depending on how many clusters were started.

3.2.2 Elastic MapReduce Amazon Web Service

During the course of finding the best settings for the Hadoop cluster designed for GA optimization, a comparison was made between above mentioned Amazon Web Service and Hadoop cluster built from scratch. This web service makes it very easy to start up a Hadoop cluster and use it. It is just enough to upload a jar with compiled Java program into S3 bucket, specify bucket name in Hadoop job setup, EC2 instance type and their quantity to be used as well. Than a cluster is immediately proceeding with calculations. Nevertheless, the first drawback of this solution is that a user cannot specify such settings as number of slots for map and reduce tasks per Task Tracker or maximal map/reduce tasks capacity of whole cluster. Motivation behind such approach was probably to give a general purpose web service. Few tests were realized which used 5 generation benchmark in order to assess a performance of Elastic MapReduce. Results are presented in Table 6.

Table 6: Test results for different MapReduce clusters

instance type	**no. of instances**	**generations**	**simulations**	**population**	**time taken (s)**
m1.small	1	1	100	30	194
c1.xlarge	1	1	200	100	223
m1.small	1	1	200	100	1119
c1.medium	1	1	100	30	69
c1.medium	2	1	100	30	88
macbook	1	1	100	30	50

Table 6 shows that Elastic MapReduce is performing approximately with the same efficiency like one node Hadoop cluster presented in Table 5. For faster EC2 instances calculation time is decreasing as expected. Nevertheless, when using two node Elastic MapReduce cluster, it can be easily seen that calculation time started to increase. No further experiments were realized on Elastic MapReduce, since it would just show what was already depicted in Table 6, that this web service cannot be used efficiently with GA optimization.

3.3 General Optimization Framework overview

In order for the reader to better understand the general idea standing behind the optimization framework design and basing on knowledge presented in chapter 2 following Figure 8 is given.

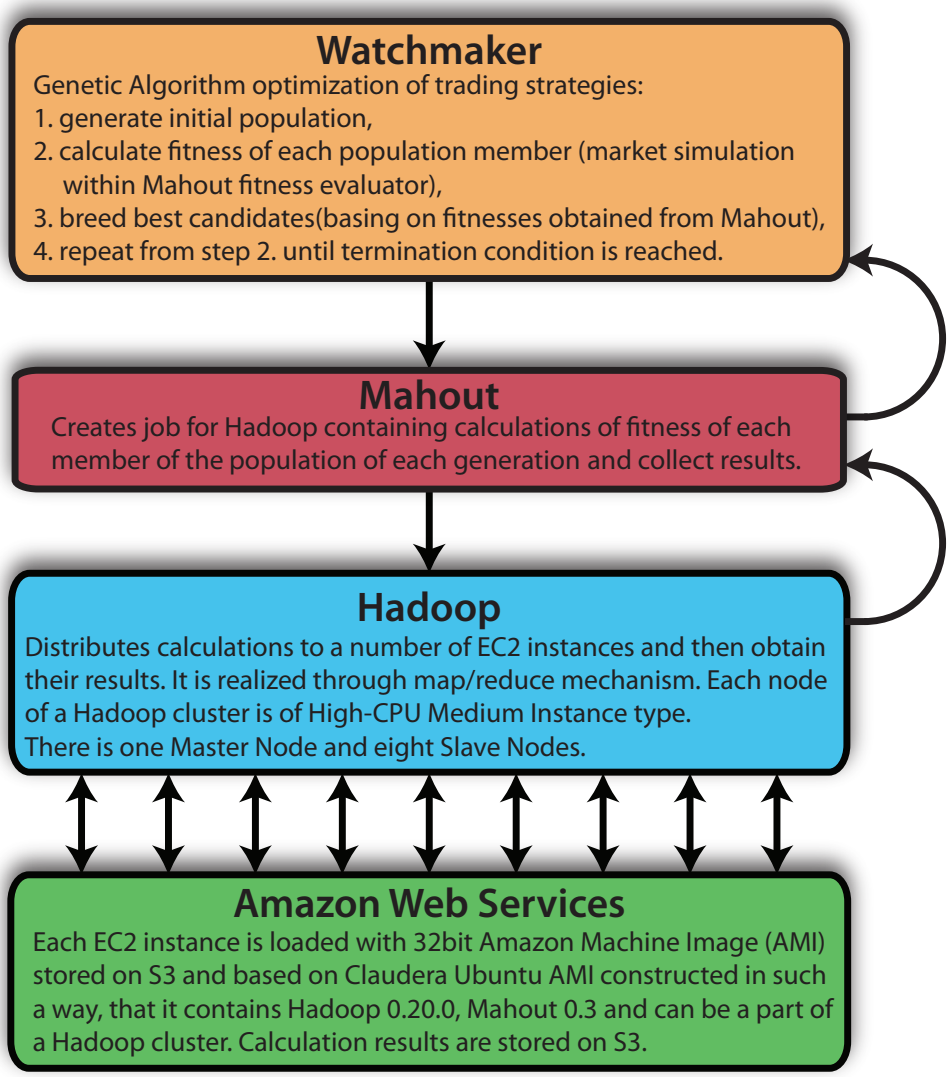

Figure 8: Optimization Framework overview

GA optimization is maintained by Watchmaker framework. Then Mahout framework is forming a Hadoop job from each generation. It includes calculations of fitnesses of each candidate solution. Next, Hadoop distributes all those calculations over all cluster nodes. Then, when reaching termination condition, results are obtained.

3.4 Optimization Framework - Java implementation

3.4.1 Optimization Framework programming

Bearing in mind that the purpose of existence of this framework is to provide an easy way to optimize different trading strategies, it was constructed in order to ensure easy usage. Because of that following design resulted. Code is organized in such a way, to force a programmer to implement proper classes. Otherwise optimization would be impossible. Two classes which have to be implemented are responsible for trading strategy operation and given trading strategy agents initialization, another three are responsible for generating initial population and maintaining crossover and mutation. Classes included into the optimization framework are presented in Table 7. Those responsible for market simulation and ZIP trading agents were based on code implemented by Cliff in his paper "Minimal-Intelligence Agents for Bargaining Behaviours in Market-Bases Environments" [3].

Table 7: Classes and packages

packages		
basega	**baseagent**	**ZIP8**
AgentEvaluator	BaseAgent	ZIP8Agent
EvolutionaryATS	BaseAgentInterface	ZIP8AgentUtils
collectingdata	BaseAgentUtils	ZIP8Crossover
DrawGraphs	BaseAgentUtilsInterface	ZIP8Factory
StoringDataS3	CtlAgentInterface	ZIP8Mutation
optimframework	DayData	**ZIP60**
gaMain	ExpCtl	ZIP60Agent
Main	ExpExecution	ZIP60AgentUtils
SettingsXMLReader	SD	ZIP60Crossover
	Smith	ZIP60Factory
	TradeData	ZIP60Mutation
	XMLReader	

3.4.2 baseagent package

This package contains all classes needed for simulating market populated with given agents. Moreover, it includes classes, which have to be extended in order to properly implement trading strategy which is to be optimized (BaseAgent, BaseAgentUtils). The functionality of the most important classes is described below.

XMLReader In order for simulation to proceed first one must input all necessary data regarding supply and demand schedules which will be used during each market simulation.

Because of that the decision was made to use XML file as a way of inputting this data. XML parser had to be written for this purpose. It reads a file which usually is called settings.xml and stores information in the ExpCtl class. This data includes controls for number of agents participating in market simulation, minimum and maximum number of trades allowed per trading period and number of trading periods. Any number of trading periods may be simulated, nevertheless each must be contained within demand/supply schedules. It is possible to have as many supply and demand schedules as needed, but its total length must not span beyond a number of trading periods defined earlier.

DayData This class is responsible for storing and calculating the most important value within whole framework, namely the alpha value. After each trading period (trading day) it stores its scored alpha value. When more than one simulation is taking place, which is the case in this optimization framework, following situation occurs. Alpha value for each repeated day is updated and its mean value over simulations of same trading day is updated as well. In the result, for example, after one hundred simulations, a mean of alpha value calculated over all those simulations and all days within each simulation is returned.

SD It is responsible for calculating theoretical and actual equilibrium prices within a given market during each trading day. Then this data is stored in TradeData class.

Smith A duty of this class is to perform a simulation of a specified number of trading periods within a market populated with trading agents of same trading strategy. It is also responsible for initializing each trading period so agents could have proper limit prices.

Whole simulation of trading process looks as follows. First it must be decided if buyer or seller will place its bid or ask. It is decided upon generating random number. When this number is greater than market Q value (This value defines the proportion of buyers to sellers. When Q = 0.5 there is same number of sellers and buyers in the market.) then bid is placed and ask otherwise. Then, depending on the type of placed offer, in case of first one, sellers are checked if their limit prices stay in accordance with bid price and in case of second one, buyers. If number of sellers or buyers willing to trade is greater than zero, then a deal may commence. In such case, during the course of the draw given seller or buyer gets the deal depending on the offer type. The rest of traders change their prices they want to place in their future bids or asks in accordance with a trading strategy they are using. If there are too many fails in finding traders willing to trade or maximum number of trades is reached, the trading period finishes. Block diagram presented below in Figure 9 presents the overview of a trading process.

ExpExecution There must be a place in which all simulations are coordinated. This is realized in this class. It is executing proper number of market simulations and outputs the alpha value in the result of those simulations.

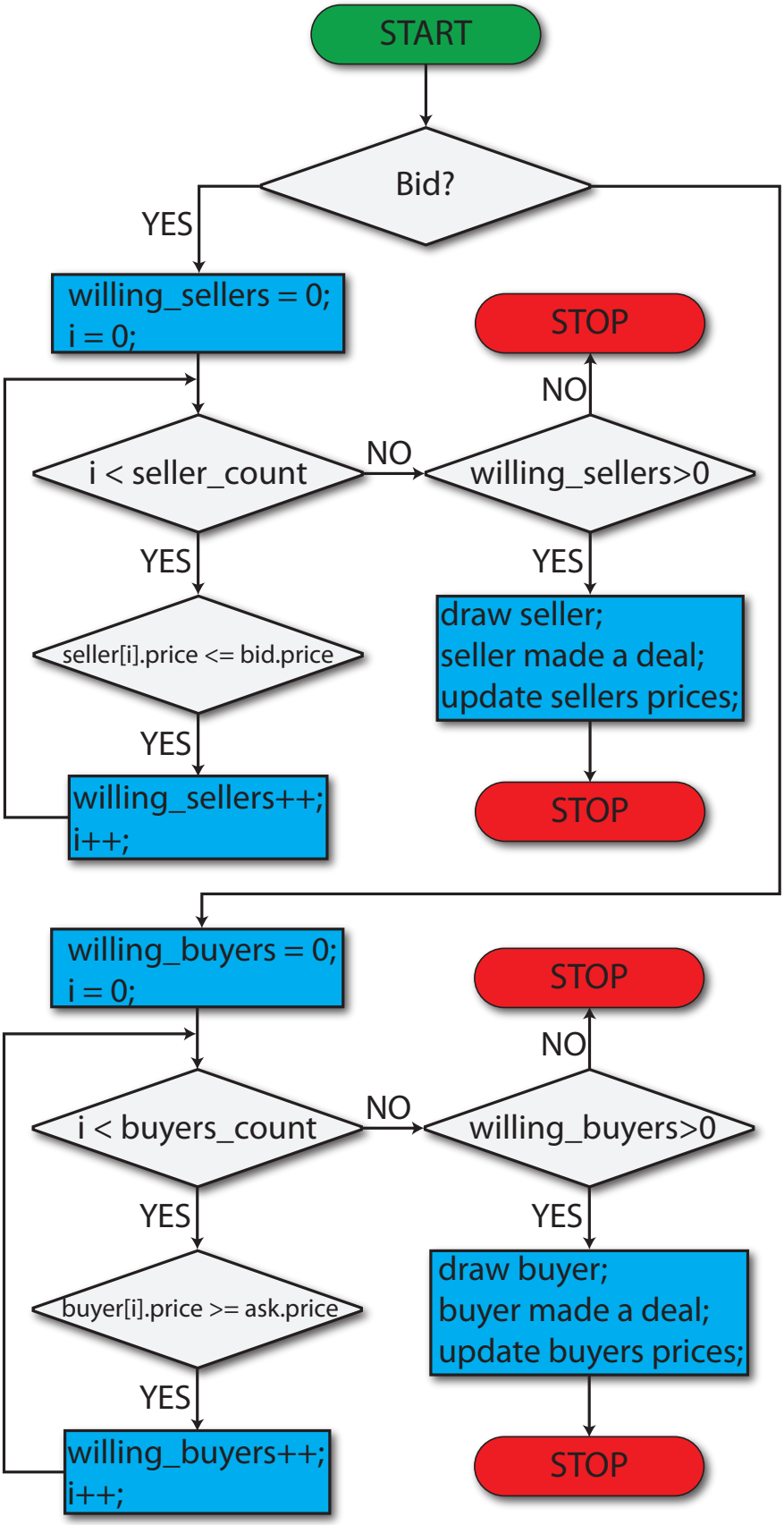

Figure 9: Trade algorithm overview

BaseAgent, BaseAgentUtils In order to prepare a trading strategy for optimization, a programmer has to code two classes responsible for trading agent operation. First one should only include functions responsible for altering the target price accordingly and the second one must be responsible for initialization of buyers and sellers. Moreover it should update the target price of a given agent, following the rules of used trading strategy.

3.4.3 basega package

Classes contained in this package are responsible for performing GA optimization itself.

AgentEvaluator It maintains actual fitness evaluation of each candidate solution. It works in the following way. Candidate solution is used as set up parameters for each market simulation and in the process its fitness is calculated. Then this information is used to select best candidates for breeding next generation of candidate solutions.

EvolutionaryATS This class is responsible for starting whole optimization process. GenerationalEvolutionEngine is started and in the result of its execution set of optimized parameters for given trading strategy is obtained. It takes crossover and mutation designed for optimized trading strategy as one of its parameters. Moreover it is necessary to specify which selection strategy it will use and in case of this framework it is tournament selection used after rank selection. Another duty of this class is to collect data necessary to perform further result analysis. In this respect fitness of elite candidate from each generation is recorded.

3.4.4 collectingdata package

Since data collecting and presenting it in user friendly form is also important, two classes where implemented in order to secure proper data presentation in scope of this package. DrawGraphs class as its name states draws results on a plot. It can be either plot of fitnesses of elite candidates through all generations or plot of prices resulting from simulation of one market populated with trading agents set with optimized parameters. Finally, it can also draw a plot presenting averaged trasnaction prices obtained in 500 market simulations. StoringDataS3 is responsible for preparing data to be stored on Simple Storage Service. This data is a set of optimized parameters and above mentioned fitnesses of best candidates from each generation.

3.4.5 ZIP8 and ZIP60 packages

Those packages hold trading strategies which can serve as an example of how one should implement other trading strategy in order to optimize it properly within the framework. During the development phase the ZIP8 served as the test bed for all tests which were realized during cluster set up and software building. Then ZIP60 was coded in order

to proceed with some exemplary optimizations in order to check how the framework is performing and to benchmark it against Cliff's results.

3.4.6 optimframework package

Classes dealing with program execution and user input interpretation are placed in this package.

gaMain This class starts execution of a given number of optimizations of a specified trading strategy. Then it initializes data storing functions and calculates total execution time of all experiments.

Main The following functionality is given by the program to its users:

- -cs <arg> the cluster is already started, give master node address
- -d use default settings
- -D <value> specify values for direct gaExec() start
- -dpf <arg> draw plot of fitnesses of elite candidates
- -elite <arg> elite candidates count
- -expcount <arg> number of experiments
- -gen <arg> number of generations is termination condition
- -le execute all experiments locally
- -nt <arg> number of trials for each candidate
- -pop <arg> population size
- -s <arg> localization of settings.xml
- -sf <arg> localization of localSettings.xml
- -sim <arg> simulate a market with given parameters stored in a file
- -sim500 <arg> simulate 500 markets with given parameters stored in a file
- -trst <arg> trading strategy which can be selected for optimization: 1 = ZIP8, 2 = ZIP60, 3 = AA

Two files must be supplied to the program: settings.xml holding information about all supply and demand schedules for market simulations and localSettings.xml. The latter one stores information about locations of jar file containing the program, keypair used for secure communication with AWS and commands which will be executed on the Hadoop cluster.

The program provides a number of options. On of them is making it possible to start an optimization locally. It can be used to compare the execution time of the cluster with single computer. Moreover, if the cluster is already started, it is enough to specify an address of

a master node in order to proceed with new optimization execution which can be realized, for example, with different number of generations or population size. This possibility to specify different experiment settings was very useful during the development phase.

3.4.7 Data storage on S3 - storings3 program

When a cluster is terminated, all optimization results are lost. Of course one may simply download all data from the master node to his or her own computer, nevertheless, since such an optimization is a time consuming process and it can take a considerable time, it would be wise to store those results in the S3. The main advantage is as such that a single computer may be stolen or may malfunction in most undesirable moment. When storing those results in above mentioned web service it is almost 100 % assurance that this data will be safe and secure. Because of that another program was written in scope of the same framework. Its only duty is to transfer files with results from master node to proper S3 bucket. It gives to each file an identifier stating which experiment it is a part of. Two types of files need two different descriptive IDs in order to quickly navigate for desired results, hence fitnesses of best candidates are stored in files having ”BCF” in their names and those containing ”R” have optimized parameters for a given trading strategy.

3.4.8 Program operation

First, the program starts the cluster with specified number of nodes. As it was said before, nine nodes is the best node count. When the cluster is started jar files containing both programs, optimization and data storage, are copied to the master node. Then a user is provided with commands saying what to do in order to log in to the master node and start the optimization and, after it completes, what should be done to store results in S3. Figure 10 pictures the process of program execution from the master node point of view.

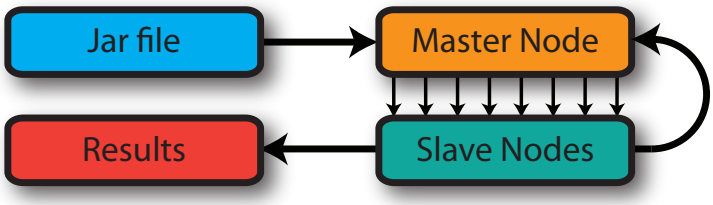

Figure 10: Program execution overview

3.4.9 Execution time

During the course of the optimization framework development, a number of different problems of technical origin had to be tackled. The measure of how well the framework is performing is the time needed to perform one full optimization of given trading strategy. In order to complete it in a quickest way possible a number of factors had to be taken into consideration during software development.

First of all the software itself had to be coded in such a way, so that each operation would take as short time to complete as possible. When one is developing a program which is just running for a few seconds, he or she may not notice any difference in running time when a given change or addition to a program code is implemented. On the other hand when a program is presumed to be running for hour or days, it is vital to code every operation in a way which is as optimal as possible. Gaining even a small fraction of a second on a single calculation may result in overall speed up of a whole program which can be measured in hours in case when this operation is repeated thousands or millions of times. Bearing in mind those conclusions, from the very beginning every new function introduced to the framework was tested if it is not increasing the running time of whole program.

The framework most CPU-intensive part is calculation of fitnesses of each candidate solution in every generation of GA. Due to this fact everything must be programmed with caution. Dealing with target price update by each agent after every transaction influenced the execution time significantly. When most object oriented approach was used, namely calling a function from a class instance, execution time was longer than the one obtained with usage of another solution. It has revealed that coding a function which will traverse through all agents and update their target prices is faster. In fact such coding made execution time of whole optimization two times shorter.

Moreover implementing a tables with hard coded size seems to be a better idea than using complex types for data storing, for example vectors. On the other hand there is no difference in execution time when optimized trading strategy, such as ZIP8 has less parameters than the other, like ZIP60.

3.4.10 Program testing

First tests of the framework where realized locally on the author's computer before any Hadoop cluster was ready. Those tests regarded mostly the proper usage of Watchmaker for implementing GA optimization. Then Mahout was introduced to the framework, so all calculations could proceed on the Hadoop cluster. Consequently, tests were helpful when checking if the framework may indeed work on such a cluster. At first it was only a one node cluster started locally. This approach was used throughout software development process, since each starting and usage of the Hadoop cluster constructed from EC2 instances is not free of charge. Nevertheless, the framework had to be started for the first time on such a remote cluster.

This first execution was successful, since only a few Standard Small Instances were used. Since they were a single core servers, no problems were revealed regarding number of threads executed per core. On the other hand when whole cluster was started on nine EC2 High-CPU Medium instances having double core processors each, something unexpected happened. Since Generational Evolution Engine was used during optimization and it can utilize easily all cores within given machine, two simultaneous optimizations taken place. Each core served thread for completely different optimization. It lead of course to errors which resulted in halting whole optimization process. It has revealed that it is the Hadoop cluster which has to have control over starting a specified number of threads on each server.

The solution to the problem was switching off this multi-thread feature of Generational Evolution Engine in the program code.

Testing the optimization framework in the nine node Hadoop cluster revealed yet another problem resulting from initial construction of the program. In first versions the supply and demand schedules were read into the program each time new simulation was started. It seemed as not very optimal solution, since reading a file takes some time, nevertheless it was left unchanged for the time being. That has revealed itself as a bug when the program was unable to proceed with the optimization. The reason to that situation was that each node was trying to read a file from its own local directory, and the file was uploaded only to the master node. It was necessary to read this file once, in the beginning of the program. There were two gains, first that the program execution time was smaller due to only one file read and second that the optimization could commence properly from that moment.

4 Obtained results

The optimization framework itself is the result of this thesis. Nevertheless, it is vital to assess its efficiency by commencing a number of experiments. They can show what is the execution time. It is the most important parameter in scope of the optimization framework efficiency. Another important result is the optimization result of ZIP60 trading strategy since it can be used to benchmark the optimization framework against experiments realized by Cliff.

4.1 Single computer vs the Hadoop cluster

First optimizations were realized on a single MacBook computer. Since number of market simulations was the value defining CPU-intensity of GA optimization, it was set to be five, because at that stage, software was still in development phase so results of tests should be obtained as quick as possible. Table 8 presents comparison of execution times between MacBook and the Hadoop cluster.

Table 8: Execution times

instance type	gen. count	market sim.	trading days	time taken (s)
MacBook	500	5	12	2868
MacBook	5	100	12	246
MacBook	500	100	12	25018
MacBook	5	100	18	352
MacBook	500	100	18	35516
c1.medium 9 nodes	5	100	12	149
c1.medium 9 nodes	500	100	12	15922
c1.medium 9 nodes	5	100	18	180
c1.medium 9 nodes	500	100	18	19324
c1.medium 15 nodes	5	200	12	183

As it can be seen from Table 8, when doing experiments on MacBook with number of market simulations set to five, execution time was pretty long in case of 500 generations and 12 trading periods per market simulation. It was even longer when 18 trading periods were used. What can be observed and what is a desirable result, is that when increasing the number of market simulations to 100, MacBook performs much worse than the Hadoop cluster. The execution time for the computer is almost twice as long when compared with the cluster.

On the other hand it is quite interesting finding, not included in this table, that one generation evaluation for 12 trading periods with five market simulations per candidate solution is much longer on the Hadoop cluster than on MacBook. The reason for this may be as such that work scheduling for nine cluster nodes itself is time consuming and when dealing with quick calculations, it takes much more time than those calculations itself.

The desirable feature of the Hadoop cluster is that it of course outperforms MacBook when performing optimization with 100 market simulations. It is worth noticing that increasing the number of calculations from 5 to 100 increased the execution time, nevertheless it is not increasing in the same manner, almost linearly, as it is in case of MacBook. It is illustrated when using 15 nodes cluster with 200 market simulations. When it would be necessary to optimize given trading strategy with more market simulations per candidate solution, it could be done easily as it was proven above.

One may conclude that doing some initial tests on his or her computer may be useful. Some money can be saved on running an experiment locally. On the other hand it is impossible to prepare properly an experiment settings while only running it locally. Different execution environment may induce other adjustments which have to be realized in order to obtain results which are reasonable. One of those could be finding the proper number of trading periods and market simulations.

4.2 Termination conditions and optimized ZIP60

When the software development reached the state in which it was possible to perform some first tests, it was necessary to find the best settings for the GA optimization. It includes the number of market simulations per candidate solutions and the number of generations which will be the termination condition. A target fitness of the best candidate was taken into consideration in this discussion.

From different runs of an optimization for ZIP60 it cannot be said which target fitness will be obtained. It depends on a number of factors such as initial population, random choices made within trading strategy to mention the most significant ones. If one would set a target fitness to two, it may result in the following two outcomes. First, it may be that this fitness would not be reached after any number of generations. In such a case the experiment could run for infinitely long time and not achieve this termination condition. Second, when this target fitness would be reached too early, the best solution would be skipped because optimization was terminated too soon. Maybe better solution could be obtained when more generations would be calculated. Basing on above mentioned conclusions, reaching given target fitness was rejected to be a termination condition.

First tests were realized with use of ZIP8 trading strategy and on a single computer. Five market simulations per candidate solution were used, since a performance of this computer is not as good as of the Hadoop cluster. Moreover, the question could be answered, how the number of market simulations influence the alpha value of elite candidates throughout all generations. Exemplary plot of such ZIP8 optimization is presented in Figure 11.

After observing Figure 11 one may came to a conclusion that after generation 400 the plot of elite candidate fitnesses is becoming stable. The average value of fitness is almost not changing from generation 400 to generation 500 so it is pointless to extend calculations beyond this later number of generations. Because of that 500 generation number is used as a termination condition.

Another observation is that this plot is noisy. Elite candidate fitnesses do converge to a given value, but it is better seen when a mean is calculated. This plot noisiness is a result

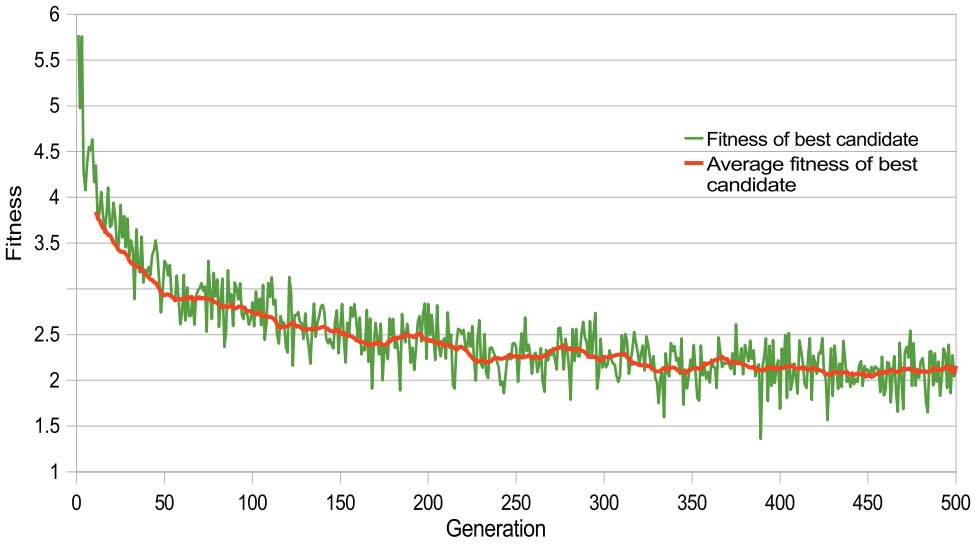

Figure 11: ZIP8 optimization

of the fact that only five market simulations were used per candidate solution. Every time a market is simulated, new set of trading agents is prepared. It results in obtaining a different alpha value from one simulation to another, nevertheless it is oscillating within some certain range. If more simulations are realized, those values are more concise from one generation to the next and better quality of the optimized parameters can be obtained. It is shown in Figure 12. The lower the fitness, the better.

The resulting plot is much less noisy, since 100 market simulations per candidate solution were used. It is of course much more time consuming to calculate them, nevertheless the resulting vector of parameters for ZIP60 is of much better quality and thanks to it the market populated with those trading agents is converging faster to a given market theoretical price. What is more, over different optimizations, different initial populations are used which results in different fitnesses of elite candidates over all generations. When fitness of an elite candidate after first generation is high, it is likely that after 500 generations it will be higher than after an optimization having lower fitness of elite candidate after its first generation. Due to this fact it is reasonable to depict in Figure 13 the averaged plots of elite candidates fitnesses from 3 different runs of this optimization. Table 9 presents fitnesses of best candidates of those 3 runs on generation 500.

Table 9: Results fitnesses

experiment no.	**result fitness**
1	2.098
2	1.697
3	2.198

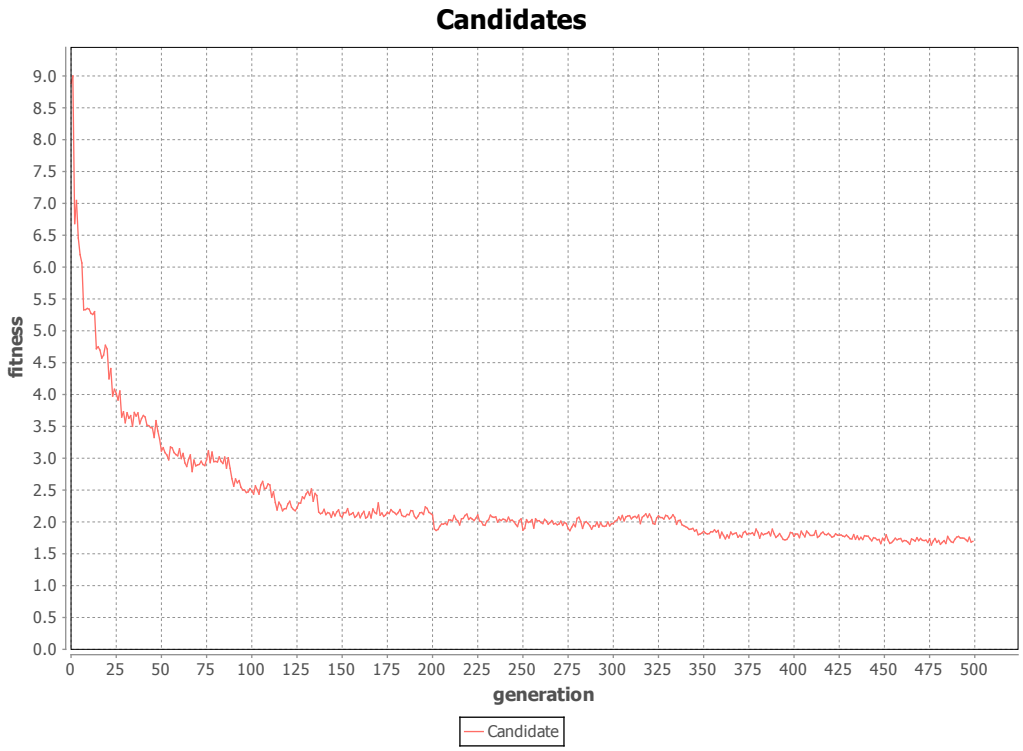

Figure 12: ZIP60 optimization

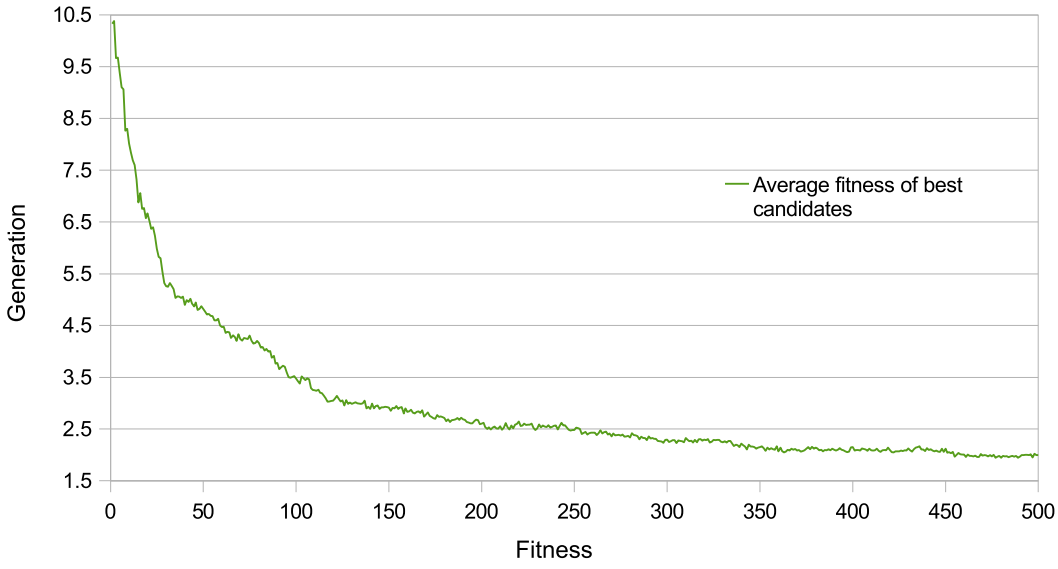

Figure 13: ZIP60 three experiments averaged

As it can be seen those resulting fitnesses are quite different, so it is most reasonable to select the best candidate solution in order to obtain the best results while utilizing ZIP60 trading agents using a set of parameters provided by that solution.

Figure 14 shows how this equilibrium is changing over 18 trading days in a market simulation used in optimizations presented in Figure 12 and in Figure 13.

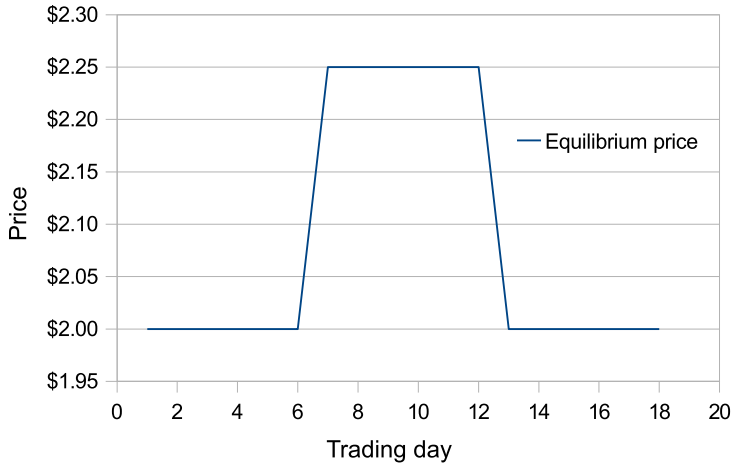

Figure 14: Market equilibrium prices over 18 trading days

The equilibrium price is changing from 2.00$ after 6 trading days to be 2.25$. After next 6 trading days it returns to 2.00$ and remains as such for last 6 trading days. Figure 15 shows only one exemplary simulation of a market populated with optimized ZIP60 trading agents.

Transaction prices are converging to theoretical market equilibrium price. When this price is changed, the shock is introduced to the market, some peaks can be seen in the plot as in the beginning of a trade, nevertheless transaction prices reach equilibrium quickly. To further prove that everything is working as intended and that the optimization framework is indeed outputting optimized set of parameters for ZIP60 trading strategy, the plot showing averaged transaction prices over 500 market simulations is depicted in Figure 16.

Figure 16 shows that transaction prices reach equilibrium prices quickly after just a few transactions.

4.3 Optimization execution time and a price

Over the course of implementing the optimization framework, the most important thing was the execution time of one optimization. On average, during testing phase and while performing actual optimization, experiments using 12 trading periods per market simulation took around 4 hours 10 minutes and those using 18 trading periods per market simulation took around 5 h 20 minutes. They used 500 generations, 30 candidates in initial population, 100 market simulations per candidate solution and 1 elite candidate. Moreover, it can be concluded that adding 33% more of trading periods to market simulation does not

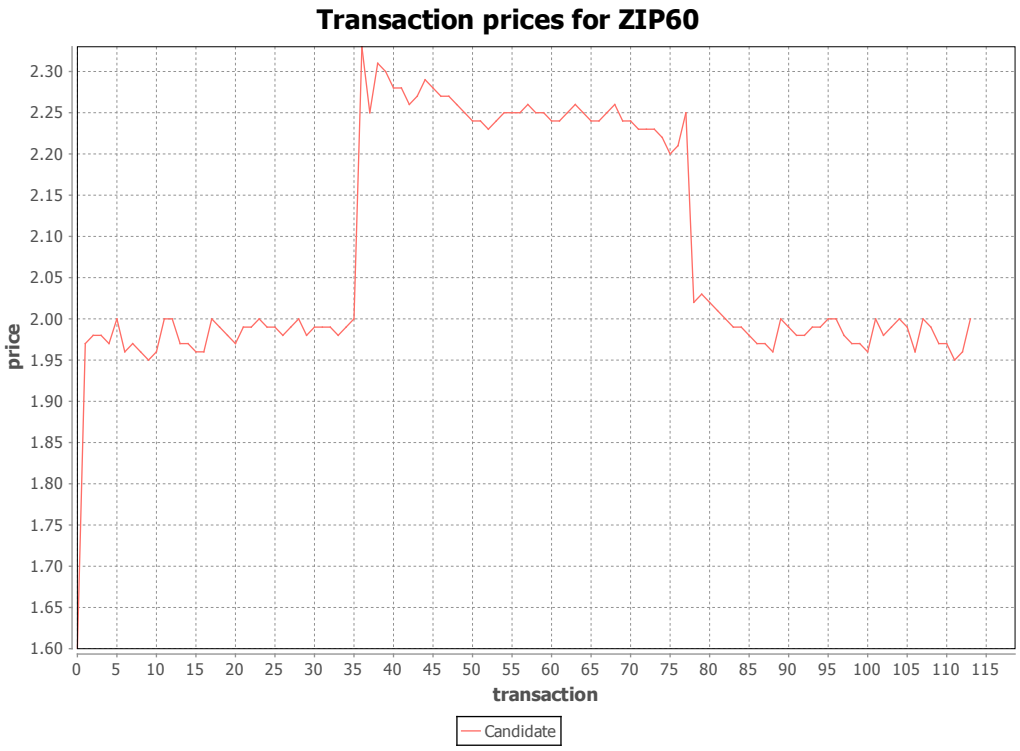

Figure 15: ZIP60 one market simulation

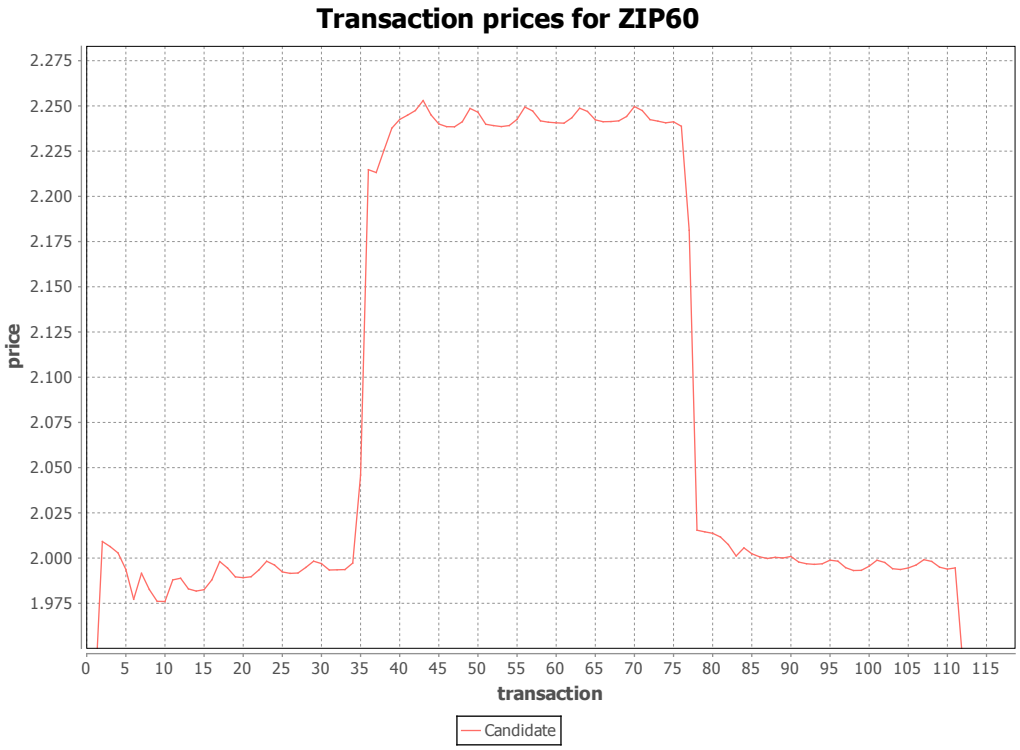

Figure 16: ZIP60 500 market simulations

imply 33% increase in total execution time, thus making whole framework more useful for optimizing various trading strategies with different number of market simulations.

The pricing of one optimization is summed up in the following Table 10.

Table 10: One optimization cost

Price for instance	$0.17
Number of EC2 instances	9
Cluster per hour	$1.53
Experiment running time (h)	5
Total cost	$7.65

As it can be seen the price of one optimization is small fraction of a price of a cluster of computers. It is far cheaper and faster to execute this optimization on Amazon Web Services, than to buy nine servers, connect and test them so they work properly, not to mention time and money which would be consumed by maintenance and electric bills. Moreover, paying $7.65 for simulating around 170000000 transactions is also a good deal.

4.4 Optimization M123 benchmark

In order to compare the alpha values of optimization results obtained by Cliff and with use of this framework, some experiments should be performed using same supply and demand schedules. In this case the decision was made to use M123 market. Nevertheless, even without performing this experiment, some comparison can be made. It regards the execution time of one optimization. According to data collected by Cliff, one experiment made with his software would take up to 16 hours. On the other hand when using the optimization framework, this time is more than 3 times shorter, being around 5 hours for one experiment.

In order to properly assess the performance of optimization, one have to compare new results with those obtained in previous experiments. In case of the optimization framework, such reference results were given in [2]. The most important figure in this case is the one from page 22 of [2] presenting optimization results for different demand and supply schedules. The decision was made to execute an optimization of M123 because it is using three different supply/demand schedules.

As it can be concluded basing on above mentioned figure, optimization M123 achieves result, which is calculated as the mean elite score over the last ten generations of 5 best optimizations selected from 50 repetitions. It is around 3.48 with standard deviation of approximately 0.1. It is a score which can be used as a benchmark for the optimization framework. Since 50 repetitions of M123 optimization were used in Cliff's experiment, it would be the best if same number of optimization could be used in this benchmarking. Nevertheless, due to still significant time which would be required to complete all 50 repetitions with use of the optimization framework, which in such a case could take up

to 11 days of running calculations, it was necessary to reduce this number of repetitions to 10. In this latter case all optimizations taken around 52 hours. Table 11 presents fitnesses of elite candidates from generation 500 from those 10 optimizations.

Table 11: M123 optimizations resulting fitnesses

Optim 1	**Optim 2**	**Optim 3**	**Optim 4**	**Optim 5**
3.65	4.01	4.10	4.03	4.33
Optim 6	**Optim 7**	**Optim 8**	**Optim 9**	**Optim 10**
4.4	3.81	3.62	4.07	3.83

As it can be concluded from Table 11 the resulting alpha values are staying within the range of values obtained by Cliff, which is a good sign showing that the optimization framework is indeed working. Figure 17 presents the plot of one optimization realized with M123 schedule and Figure 18 depicts averaged values of fitnesses of elite candidates over all ten optimizations. The market equilibrium price has the same value throughout all trading days in M123 supply/demand schedule and limit prices of trading agents are changing.

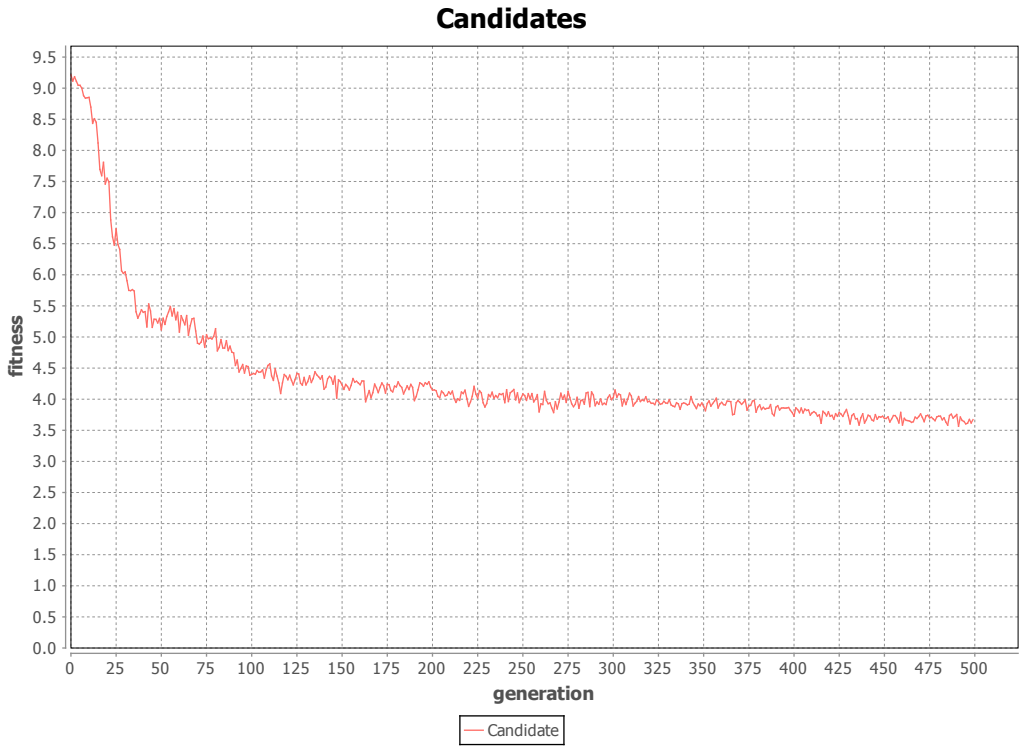

Figure 17: ZIP60 one M123 optimization

What can be derived from Figure 17 and Figure 18 is that those plots are similar in their shapes to those presented earlier in Figures 12 and 13. It is a good indication as well that the optimization framework is working with different supply/demand schedules used and

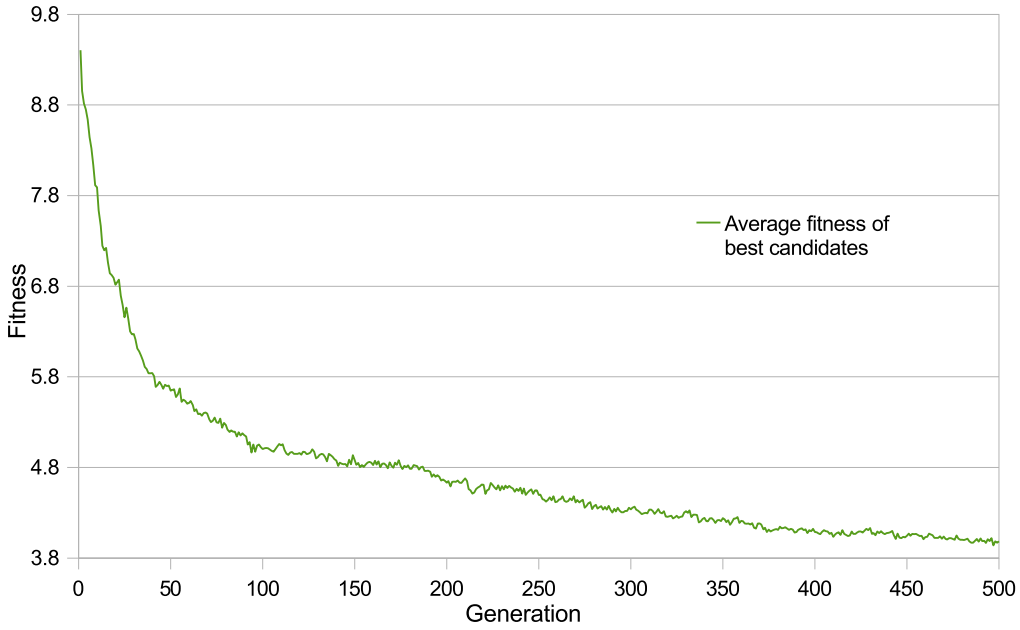

Figure 18: ZIP60 ten M123 optimizations averaged

alpha values are in same range of values from one optimization to another, thus stating that ZIP60 is being properly optimized within different market setups. Yet another interesting finding can be made while using parameters optimized for M123 schedule in a simulation using a different supply/demand schedule utilized in earlier experiment described in this thesis and presented in Figures 12 and 13. Results of such one market simulation and 500 market simulations are presented in Figure 19 and 20.

One may observe that even though the parameters for ZIP60 were optimized for different market, the trading strategy is performing very well. It quickly converges to equilibrium prices and stays there until market shock is introduced. Moreover, same behaviour may be observed in case of averaging transaction prices from 500 market simulations, namely basing just on this plot one may easily see when each trading period is starting and ending, which is shown by peaks in each trading day and it is a common thing for Figure 20 and Figure 16. It is yet another evidence that the optimization framework is working properly, especially the implementation of market simulation.

Nevertheless, it is worth to present result of 1 and 500 market simulations with M123 supply/demand schedule. They are presented in Figure 21 and Figure 22.

As it was stated earlier, M123 supply/demand schedule does not have any change in market equilibrium price, nevertheless it can be observed from Figure 21 and Figure 22 that every 6 trading periods there is a sudden peak in transaction price, nevertheless ZIP60 trading strategy assures that equilibrium price is reached within just a few transactions.

However, in order to properly compare results of optimizations realized within the optimization framework and in experiments realized by Cliff, it is necessary to calculate standard deviation and mean elite score over last ten generations in case of recently performed best five M123 optimizations. The results are presented in Table 12.

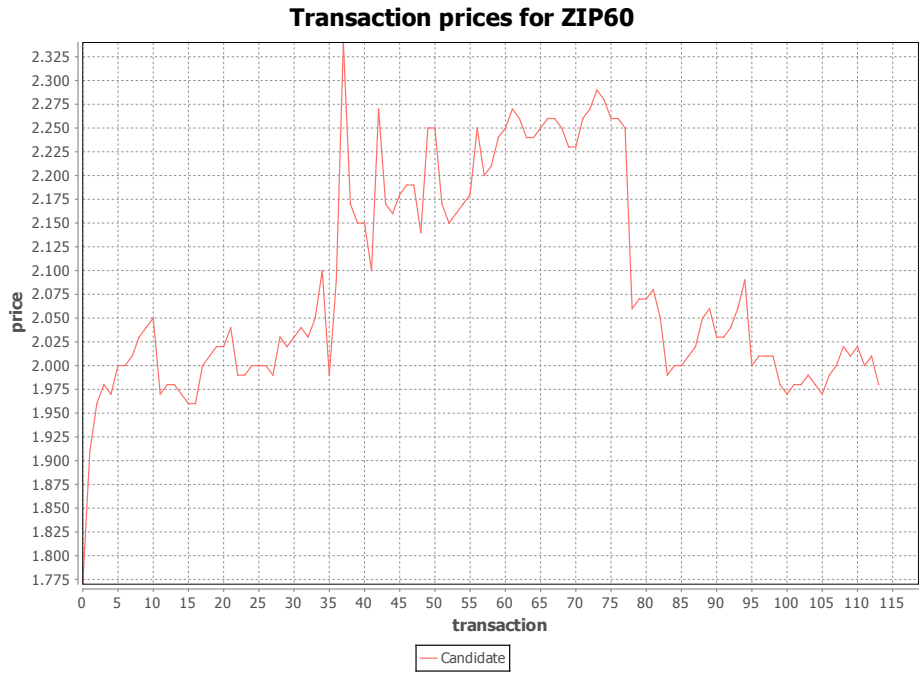

Figure 19: ZIP60 one market simulation with parameters optimized for M123, but used with different supply/demand schedule in that simulation

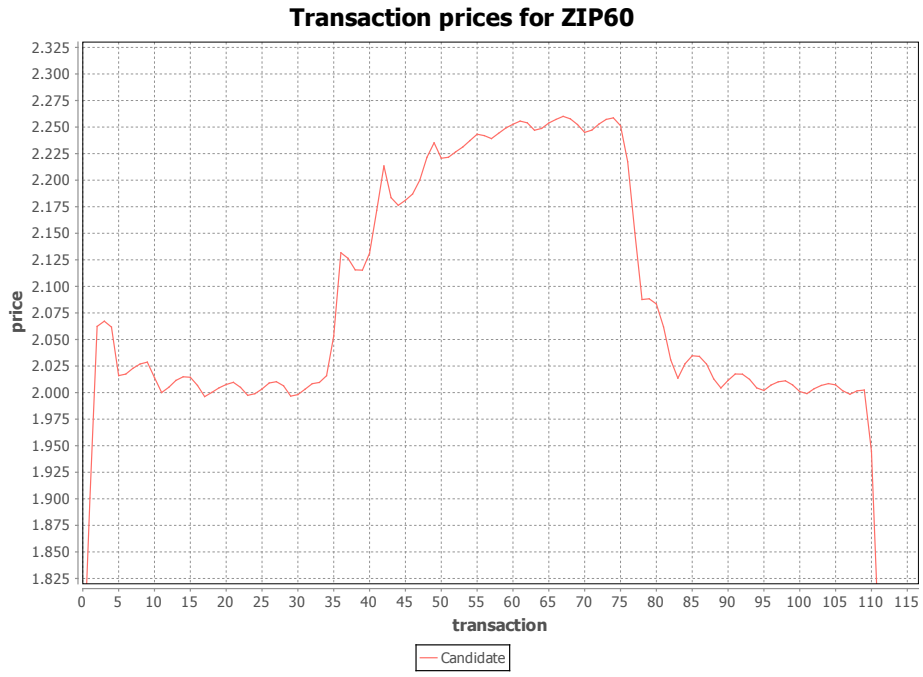

Figure 20: ZIP60 500 market simulations with parameters optimized for M123, but used with different supply/demand schedule in those simulations

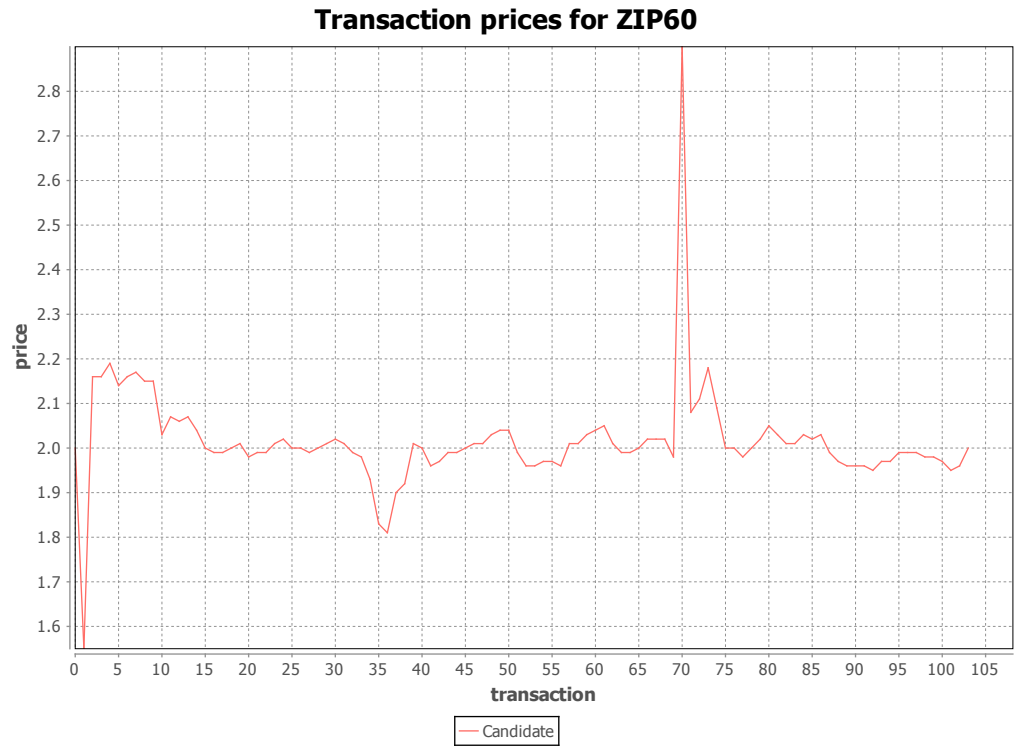

Figure 21: ZIP60 one market simulation with parameters optimized for M123

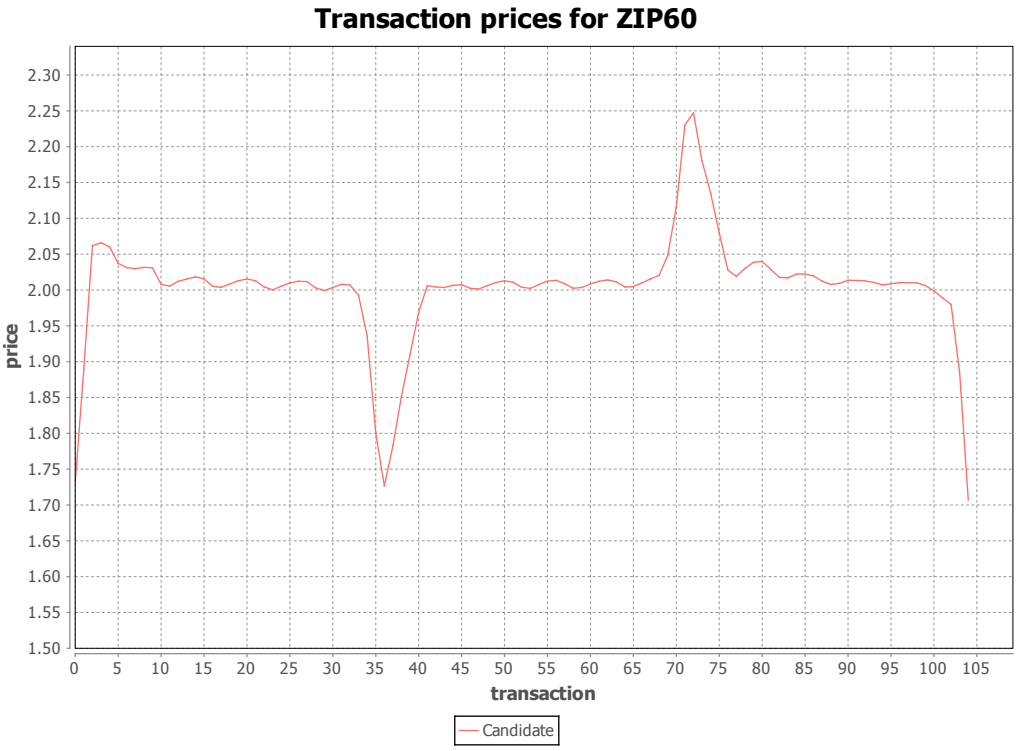

Figure 22: ZIP60 500 market simulations with parameters optimized for M123

Table 12: M123 optimizations means and standard deviation

Optim means	**Mean of means**
3.65	3.78
4.01	**Std dev**
3.81	0.16
3.62	
3.83	

As it can be seen from Table 12, obtained mean from means of elite candidates from last 10 generations of each optimization is slightly worse than the one obtained by Cliff. Moreover, standard deviation is higher than in previous experiments. Just 10 instead of 50 repetitions were completed, hence there were fewer results to choose from and to select the best ones. It is likely that when more results would be generated, it would be easier to achieve lower standard deviation of results, since they would have more similar values. What is more, since the results obtained in best 5 repetitions present worse mean than those obtained by Cliff, it is likely that when more repetitions could be completed, the best results, which could be selected, would be better. The clue for that might be derived from following fact that those results range from 3.62 to 4.4.

Finally, one may conclude that benchmarking of the optimization framework against Cliff experiments was a good idea since it was the only existing attempt to optimize ZIP60 with GA. Basing on above mentioned results it can be written that the newly developed software is performing its job well and as intended - everything is completed much faster than in previous experiments. Moreover, obtained results are also of good quality since they are comparable with results obtained by Cliff.

5 Evaluation of the work

The optimization framework is a complex software, basing on many different ideas and working on remotely located hardware. Many choices had to be made during its development. The time has come to analyse if initial objectives were achieved.

5.1 Initial objectives vs achieved objectives

The main objective, which was designing and developing the optimization framework for trading strategies using Amazon Web Services was achieved for sure. The framework is working as expected, it outperforms single computer by far and its usage is very cheap. Moreover its easy to be used, since it is programmed in such a way, so that a programmer has to implement just 5 classes constituting trading strategy, how candidate solutions should mutate and perform crossover during the optimization and finally, how initial population has to be constructed.

Objectives, which were meant to be reached during the software development, were achieved as well. Familiarization with Amazon Web Services was realized, otherwise running the Hadoop cluster on EC2 instances would be rendered impossible. Simulation of Continuous Double Auction market was implemented as well, agents are trading with each other as intended and they follow the rules of CDA.

On the other hand one may argue if really the best possible, in scope of optimizing trading strategies, optimization algorithm was used. In other papers, for example those written by Cliff [2], it is suggested in the further work chapter to use different optimization method than Genetic Algorithm. Those suggestions were not followed in this framework implementation. One may say that it is a good example of not exploring all possibilities. Yes, its true, no other optimization algorithm, apart from GA, was checked if it gives better results. Maybe it would give a result of same fitness, but in less time, being faster than GA. Nevertheless, one may come to a bit different conclusion that using only the GA was done on purpose which was the case in this optimization framework. If other optimization algorithm would be used, it could be hard to compare the performance of this new software, which regards execution time and obtained results, with work completed earlier by Cliff. In case of using same optimization algorithm in both experiments, it is more reasonable to say, that comparison is having same reference point, thus can actually be conducted.

What was described in previous paragraph may induce one more question. Since the comparison is made, should the newly developed software be running on the equipment used by Cliff in his previous experiments in order to perform really good comparison? The answer depends on what actually is being compared. Is it only a software performance or optimization execution time realized in completely different way from the old one which one is comparing. In case of the first approach, it would be true that all tests should be done on the previously used hardware. Nevertheless, it would reduce whole project just to developing an optimization framework with use of new ideas. This task itself is not small, however it lacks novelty of using completely different hardware, which implies different approach to software construction. In such a case for sure the main goal of the thesis

would not be achieved. Fortunately, the thing which one can be more interested in is how whole optimization process can be realized with use of cloud computing. This is why the second approach was followed, introducing new approach to realizing CPU-intensive tasks. Thanks to it comparison is not only made between different software, but between different approaches to realization of the same task as well.

Another objective was to perform proper optimization of a given trading strategy. In this case it was realized for ZIP60. Again, this objective was achieved, since obtained result, being the set of parameters for ZIP60, is indeed optimized. After being compared with Cliff results, it can be said that the framework performed its task well, since both results are comparable and of pretty same alpha values for given supply and demand schedules. The optimization framework also outperforms the older software made by Cliff in scope of optimization execution time, so it is yet another evidence that the main goal was achieved.

Another thing which could be completed was to compare AA trading strategy with ZIP60 trading strategy. As it has revealed, programming of the optimization framework and testing it was a very time consuming process. Moreover construction of the Hadoop cluster and finding the best settings for it took a considerable amount of time as well. It is likely that when more time could be utilized on implementing yet another trading strategy so it could be tested, AA would be optimized and compared with ZIP60 as well, but AA experiment would take more time to complete, since this trading strategy is performing much more calculations in order to properly specify a target price for a given agent. Nevertheless, this task can be done and it is further described in chapter 6. The number of optimizations performed for ZIP60 could be higher as well in order to achieve even better results, however it would require more time scheduled for whole project in order for those optimizations to complete. All in all, the conclusion can be made that main objective has been achieved which is the most important thing in scope of the thesis.

5.2 Choices made

Throughout the optimization framework design and development a number of decisions and choices had to be made. First of all, what scheme in the optimization framework design should be followed. After familiarization with MMM and Watchmaker framework better one had to be selected. Both have their advantages, nevertheless the latter seems to be a better solution from the time perspective. First of all, Watchmaker can be used on Hadoop via Mahout which is the most important thing. It deals with all things regarding Hadoop job construction and distributing calculations over several servers with help of Mahout. Nevertheless, not all things can be controlled programatically when such solution is used. On the other hand the MMM scheme provides idea on how Amazon Web Services can be used, but for pretty different application than for performing GA optimization. Everything, starting from instances communication to job scheduling has to be programmed, thus programmer may expand greater control over system he or she is building. Nevertheless, since programming GA algorithm, choosing best EC2 instance and constructing market simulation is time consuming as well it would be too risky to follow MMM scheme. One may simply run out of time. Instead of dealing with problems connected with proper

framework programming, one would delve into problems loosely connected with GA optimization.

Another notion was market simulation. JADE, as described earlier, was considered as a framework which could be used for this purpose. At first, it seemed to be a good idea, since simulation is dealing with trading agents and this framework is designed for maintaining agent to agent interactions. It performs well, when execution time is not so important, since each interaction takes some time. Of course, it takes fraction of a second for two or more agents to interact, nevertheless, in scope of this thesis, one of the most important things was the optimization execution time and because of that other, faster method of market simulation was selected as it is described earlier.

Other selection was made when selecting EC2 instance which will be used within the Hadoop cluster. Many different factors had to be taken into consideration such as calculation efficiency, pricing and overall performance within the cluster. Moreover, new instance was introduced when the optimization framework construction was reaching its finished, it is Cluster Compute Instance. Maybe it would perform better than whole cluster. Nevertheless, since earlier the decision was made to use High-CPU Medium Instance within the Hadoop cluster, testing this new instance can be treated as a further work. It has revealed that this High-CPU instance is performing well within the Hadoop cluster with nine nodes.

What regards cluster node count, it can also be taken into this discussion. One may question this number of nodes, but for the purposes of this framework, cluster with this size is performing its duty well. Maybe usage of more nodes would bring results faster? This question can be answered in the following way. If the decision process would be different, namely first the number of nodes would be selected and then tests of Hadoop cluster with such number of nodes would proceed, settings would be found for cluster of this size. Nevertheless, as it was shown earlier in the chapter 3, there is no time improvement when adding more nodes. When starting from 16 nodes straight away, one could simply waste money on optimizing the cluster which could be reduced to fewer nodes and have same performance. From the time perspective, it can be said that chosen path during Hadoop cluster construction is better.

Finally, the selection of Hadoop should be reasoned as well. It has revealed that it can be utilized when performing GA optimization. Nevertheless, Hadoop was primarily designed for processing large amounts of data while performing not very CPU-intensive calculations. One may argue that Hadoop should not be used for GA optimization. To some extent he or she would be right. But on the other hand Watchmaker framework itself would not be designed in such a way in order to give the possibility to distribute its calculations over a Hadoop cluster, if this framework would be really useless. Moreover, CPU cores of each node being in such a cluster are utilized in 100 % when performing CPU-intensive calculations and it is yet another reason why it is reasonable to utilize Hadoop for optimizing trading strategies as well.

6 Possible improvements and further work

6.1 Possible improvements

Finding of the best Hadoop cluster settings could be also realized with the help of GA. Nevertheless, it would be expensive and time consuming job. On the other hand, if more tests would have been completed during the process of cluster testing, probably the cluster would work better and execution time of experiments would be shorter. However, already found settings made the cluster fast enough to obtain results in reasonable time.

Framework could be programmed in more versatile way, in order to make it possible to optimize trading strategies within other markets, for example the one with persistent orders. Now, if one would like to use different market simulation, he or she would have to code it from scratch. There are no classes which could be just extended and used to implement such a different market fast. Moreover, this versatility could be also included into adding new features to whole framework, such as new ones for visualizing obtained results or collecting more statistics. New class representing such a visualization feature could be implemented, making it easier to extend existing functionality easier and faster for a user of the optimization framework.

6.2 Further work

Work for the future developers of this optimization framework could include dealing with following issues. First of all, one could explore other optimization methods in order to check if they are better in optimizing trading strategies than GA.

Another thing could be testing Cluster Compute Instance. It gives large computing power to its users, maybe it could work better than a nine node cluster. Following this direction of exploring other features of AWS, one may as well try to build whole cluster using SQS, EC2 and S3 without any other frameworks designed to distribute calculations.

Moreover, exploring new settings which could be used within optimization, such as number of market simulations and population size, can also be considered as a further work. New hardware was used when compared with previous experiments, so it may reveal that increasing the number of candidate solutions or market simulations could improve final result, while the execution time would not increase significantly. It was tested however to some extent during the framework development, nevertheless, it could be checked with different settings values as well.

One may also attempt to implement AA trading strategy within the optimization framework. It could be done with just some minor code alterations, since the framework is designed in such a way, so it is easy to add new trading strategies for optimization. When work with implementing AA would be completed, it could be optimized and a question which one, after being optimized, is better - AA or ZIP60. Moreover, in the future, there will be more trading strategies implemented and they exist already. This optimization framework could be used in order to find the best settings for them as well.

The designed and produced optimization framework itself is believed to be changeable in such a way, that with some minor code adjustments and implementations, it could be able to optimize any system. Such a system should be using some input settings and it should be possible to simulate it. It could be something completely different than a trading strategy, thus extending the possible usages of the optimization framework even further.

References

[1] Nicholas Carr, **The Big Switch: Rewiring the world, from Edison to Google**, W. W. Norton & Company, Inc., 2009

[2] Dave Cliff, **ZIP60: Further Explorations in the Evolutionary Design of Trader Agents and Online Auction-Market Mechanisms**, Foreign Exchange Complex Risk Group, Deutsche Bank, 2006

[3] Dave Cliff, **Minimal-intelligence agents for bargaining behaviours in market environments**, Hewlett-Packard Laboratories Technical Report HPL-97-91, 1997

[4] Perukrishnen Vytelingum, **The Structure and Behaviour of the Continuous Double Auction**, Faculty of Engineering, Science and Mathematics School of Electronics and Computer Science Intelligence, Agents, Multimedia Group, University of Southampton, 2006

[5] IBM, **Datacenter Services**, 2010
http://www-935.ibm.com/services/us/index.wss/offerfamily/gts/a1027717

[6] Google, **Google Apps**, 2010
http://www.google.com/services/

[7] Microsoft, **Windows Azure Platform**, 2010
http://msdn.microsoft.com/en-gb/ee514245.aspx?WT.srch=1

[8] Amazon, **Amazon Web Services**, 2010
http://aws.amazon.com/

[9] Mitch Garnaat, **Monster Muck Mashup - Mass Video Conversion Using AWS**, 2007
http://developer.amazonwebservices.com/connect/entry.jspa?externalID=691

[10] Fabio Bellifemine, Giovanni Caire, Dominic Greenwood, **Developing Multi-Agent Systems with JADE**, John Wiley & Sons, Ltd, 2007

[11] Friedman, D., Rust, J., **The Double Auction Market: Institution, Theories, and Evidence**, Addison Wesley, 1993

[12] Rajarshi Das, James E. Hanson, Jeffrey O. Kephart and Gerald Tesauro, **Agent-Human Interactions in the Continuous Double Auction**, Institute for Advanced Commerce, IBM T.J. Watson Research Center, 2001
http://www.research.ibm.com/infoecon/paps/AgentHuman.pdf

[13] Graham-Rowe, D., **How Bots Can Earn More Than You Do**, New Scientist (20 Aug., 2005) 187(2513):26-27

[14] Pritchard, S., **Zippy Agents Going for Brokers**, Financial Times (London) FT-IT Review. July 13, 2005: p.2.

[15] Economist, **The March of the Robo-Traders**, The Economist Technology Quarterly, pp.23-24, 15 September 2005. Note: The Economist policy is to not iden- tify the authors of their articles.

[16] Richard Monson-Haefel, PJ Cabrera, **Introduction to AWS for Java Developers**, Amazon, 2007
http://developer.amazonwebservices.com/connect/entry.jspa?externalID=848

[17] Martin Gudgin, Microsoft, Marc Hadley, Sun Microsystems, Noah Mendelsohn, IBM, Jean-Jacques Moreau, Canon, Henrik, Frystyk Nielsen, Microsoft, Anish Karmarkar, Oracle, Yves Lafon, W3C, **SOAP Version 1.2 Part 1: Messaging Framework (Second Edition)**, W3C, 2007
http://www.w3.org/TR/soap12-part1/#intro

[18] Alex Rodriguez, **RESTful Web services: The basics**, IBM, 2008
https://www.ibm.com/developerworks/webservices/library/ws-restful/

[19] JetS3t, 2010
http://bitbucket.org/jmurty/jets3t/wiki/Home

[20] Amazon Web Services, **Amazon Simple Storage Service**, 2010
http://docs.amazonwebservices.com/AmazonS3/2006-03-01/

[21] Amazon Web Services, **Amazon Simple Queue Service**, 2010
http://docs.amazonwebservices.com/AWSSimpleQueueService/latest/SQSGettingStartedGuide/

[22] Amazon Web Services, **Amazon Elastic Compute Cloud**, 2010
http://aws.amazon.com/ec2/

[23] Amazon Web Services, **Amazon Virtual Private Cloud**, 2010
http://aws.amazon.com/vpc/

[24] Amazon Web Services, **Amazon SimpleDB**, 2010
http://aws.amazon.com/simpledb/

[25] Amazon Web Services, **Amazon Machine Images (AMIs)**, 2010
http://developer.amazonwebservices.com/connect/kbcategory.jspa?categoryID=171

[26] Mitch Garnaat, **Son of Monster Muck Mashup - Mass Video Conversion Using AWS**, Amazon Web Services, 2008
http://developer.amazonwebservices.com/connect/entry.jspa?externalID=1404&categoryID=152

[27] FIPA, **FIPA2000**, 2005
http://www.fipa.org/specifications/index.html

[28] Garson G. Quantification in Modal Logic. **In Handbook of Philosophical Logic**, Vol. II: **Extensions of Classical Logic**, pp. 249 - 307, D. Reidel Publishing Company, 1984

[29] Rao, A.S. and Georgeff, M. BDI Agents: from Theory to Practice. In **Proceedings of the 1st International Conference on Multi-Agent Systems**, pp. 312 - 319, San Francisco, CA, 1995

[30] Smith, V., **Experimental study of competitive market behavior**, Journal of Political Economy, 70:111-137, 1962

[31] Gode, D., Sunder, S., **Allocative efficiency of markets with zero-intelligence traders**, Journal of Political Economy, 101:119-137, 1993

[32] Clearwater, S., ed., **Market-Based Control**, World Scientific Press, 1995

[33] Dave Cliff, Bruten, J., **Animat Market-Trading Interactions as Collective Social Adaptive Behavior**, Adaptive Behavior 7(3&4):385-414, 1999

[34] Tesauro, G., Bredin, J., **Strategic Sequential Bidding in Auctions using Dynamic Programming**, Proceedings of the First International Conference on Autonomous Agents and Multiagent Systems (AAMAS-02), Bologna, Italy, 2002

[35] Tesauro, G., Das, R., **High-Performance Bidding Agents for the Continuous Double Auction**, presented at Economic Agents, Models, and Mechanisms Workshop, International Joint Conference on Artificial Intelligence (IJCAI), 2001

[36] Cliff, D., **Genetic optimization of adaptive trading agents for double-auction markets** in Proceedings of Computational Intelligence in Financial Engineering (CIFEr), New York, 1998. IEEE/IAFE/Informs (preprint proceedings), pp.252-258, 1998

[37] Cliff, D., **Evolutionary optimization of parameter sets for adaptive software-agent traders in continuous double-auction markets**. Presented at the Artificial Societies and Computational Markets (ASCMA98) workshop at the Second International Conference on Autonomous Agents, Minneapolis/St. Paul, May 1998. Also available as HP Labs Technical Report HPL-2001-99, 2001

[38] Rumelhart, D., Hinton, G., Williams, R. (1986), **Learning internal representations by error propagation, in Parallel Distributed Processing: Explorations in the Microstructures of Cognition**, Volume 1, edited by D. Rumelhart and J. McClelland (Cambridge, MA: MIT Press), pp. 318-362, 1986

[39] Wilson, S., **Classifier Fitness Based on Accuracy**, Evolutionary Computation, 3(2):149-175, 1995

[40] Goldberg, D., **Genetic Algorithms: In Search, Optimization, and Machine Learning**, Reading, MA: Addison Wesley, 1989

[41] Mitchell, M., **An Introduction to Genetic Algorithms**, MIT Press, 1998

[42] Cliff, D., **Evolution of market mechanism through a continuous space of auction-types**, Presented at Computational Intelligence in Financial Engineering (CIFEr) session at Congress on Evolutionary Computation (CEC2002), Hawaii, May 2002

[43] Cliff, D., **Evolution of market mechanism through a continuous space of auction-types II: Two-sided auction mechanisms evolve in response to market shocks**, Presented at Agents for Business Automation session at IC2002, Las Vegas, June 2002, In: Proceedings of the International Conference on Internet Computing IC02, Volume III, edited by H.R. Arabnia and Y. Mun. CSREA Press, pp.682-688.

[44] Cliff, D., **Visualizing Search-Spaces for Evolved Hybrid Auction Mechanisms**, Presented at the Beyond Fitness: Visualizing Evolution workshop at the 8th International Conference on the Simulation and Synthesis of Living Systems (ALifeVIII) conference, Sydney, December 2002. Also available as Hewlett-Packard Laboratories Technical Report HPL-2002-291.

[45] Cliff, D., **Evolution of Market Mechanism Through a Continuous Space of Auction-Types III: Multiple Market Shocks Give Convergence Toward CDA**, Hewlett-Packard Laboratories Technical Report HPL-2002-312.

[46] Cliff, D., **Explorations in evolutionary design of online auction market mechanisms**, Electronic Commerce Research and Applications 2(2):162-175, 2003

[47] Grant Ingersoll, **Mahout on Amazon EC2**, Mahout, 2010
https://cwiki.apache.org/confluence/display/MAHOUT/Mahout+on+Amazon+EC2

[48] The Apache Software Foundation, **Hadoop-common API**, 2010
http://hadoop.apache.org/common/docs/current/api/overview-summary.html

[49] Daniel W. Dyer, **Watchmaker Framework for Evolutionary Computation**, 2010
http://watchmaker.uncommons.org/

[50] The Apache Software Foundation, **Apache Mahout**, 2010
http://lucene.apache.org/mahout/

[51] The Apache Software Foundation, **Apache Hadoop**, 2010
http://hadoop.apache.org/

[52] Hadoop Wiki, **PoweredBy**, 2010
http://wiki.apache.org/hadoop/PoweredBy

[53] The Apache Software Foundation, **Hadoop Cluster Setup**, 2010
http://hadoop.apache.org/common/docs/r0.20.2/cluster_setup.html

[54] All Things Hadoop, **Tips, Tricks And Pointers When Setting Up Your First Hadoop Cluster To Run Map Reduce Jobs**, 2010
http://allthingshadoop.com/2010/04/28/map-reduce-tips-tricks-your-first-real-cluster/

[55] Omer Trajman, Alex Loddengaard, John Kreisa, **Top 10 Tips & Tricks for Hadoop Success**, SlideShare Inc., 2010
http://www.slideshare.net/cloudera/top-ten-tips-tricks-for-hadoop-success-r9?src=related_normal&rel=3346153

[56] Srigurunath Chakravarthi, **Tuning Hadoop for Performance**, SlideShare Inc., 2010
http://www.slideshare.net/ydn/hadoop-summit-2010-tuning-hadoop-to-deliver-performance-to-your-application

Appendix - Source Code

```
package basega;
(...)
//starts Genetic Algorithm optimization
public class EvolutionaryATS {
    public StoringDataS3 sds3;
    private AgentEvaluator eee;
    public double[] createEvolution(int popsize, int elite, int gencount, ExpCtl expctl,
          StoringDataS3 sds3t, int n_exps, BaseAgentUtils bau, AbstractCrossover ac,
          EvolutionaryOperator am, AbstractCandidateFactory acf, int execType) {
        sds3 = sds3t;
        Random rng = new MersenneTwisterRNG();
        List<EvolutionaryOperator<double[]>> operators = new ArrayList<
              EvolutionaryOperator<double[]>>(2);
        operators.add(ac);
        operators.add(am);
        EvolutionaryOperator<double[]> pipeline = new EvolutionPipeline<double[]>(
              operators);
        GenerationalEvolutionEngine<double[]> engine = null;
        if (execType == 0) {
            eee = new AgentEvaluator(expctl, n_exps, bau);
            MahoutFitnessEvaluator<double[]> eval = new MahoutFitnessEvaluator<double[]>(
                  eee);
            engine = new GenerationalEvolutionEngine<double[]>(acf,
                    pipeline,
                    eval,
                    new RankSelection(new TournamentSelection(new Probability(0.8))),
                    rng);
            engine.setSingleThreaded(true);
        }
        if (execType == 1) {
            eee = new AgentEvaluator(expctl, n_exps, bau);
            engine = new GenerationalEvolutionEngine<double[]>(acf,
                    pipeline,
                    eee,
                    new RankSelection(new TournamentSelection(new Probability(0.8))),
                    rng);
        }
        engine.addEvolutionObserver(new EvolutionLogger());
        return engine.evolve(popsize, elite, new GenerationCount(gencount), new
              TargetFitness(0, false));
    }
    private class EvolutionLogger implements EvolutionObserver<double[]> {
        public void populationUpdate(PopulationData<? extends double[]> data) {
            sds3.collectBestCandidatesFitness(data.getBestCandidateFitness());
            System.out.println("Generation = " + data.getGenerationNumber() + " Best
                  candidate fitness = " + data.getBestCandidateFitness());
        }
    }
}
```

Source code 1: EvolutionaryATS.java

```
package basega;
(...)
//evaluates fitness of each candidate solution
public class AgentEvaluator implements FitnessEvaluator<double[]>, CtlAgentInterface {
    private ExpCtl expctl;
    private BaseAgentUtils bau;
    private int n_exps;
    public AgentEvaluator(ExpCtl expc, int n_exps, BaseAgentUtils bau) {
        this.bau = bau;
        expctl = expc;
        this.n_exps = n_exps;
    }
    public double getFitness(double[] candidate, List<? extends double[]> population) {
        Vector<Double> dummyVec = new Vector<Double>();
        double fitness = ExpExecution.Exps(candidate, n_exps, expctl, bau, dummyVec);
        return fitness;
    }
    public boolean isNatural() {
        return false;
    }
}
```

Source code 2: AgentEvaluator.java

```
package ZIP60;
(...)
//implements ZIP60 trading strategy
public class ZIP60Agent extends BaseAgent {
    public double[] beta = new double[MAX_N_CASES];
    public double[] momntm = new double[MAX_N_CASES];
    public ZIP60Agent() {
    }
    public ZIP60Agent(double[] betamin, double[] betaspread,
            double[] gammamin, double[] gammaspread,
            double[] camin, double[] caspread,
            double[] crmin, double[] crspread) {
        bank = 0;
        n = 0;
        sum = 0;
        last_d = 0;
        for (int c = 0; c < MAX_N_CASES; c++) {
            beta[c] = randomNumber(betamin[c], betaspread[c]);
            if (beta[c] > 1) {
                beta[c] = 1;
            }
            momntm[c] = randomNumber(gammamin[c], gammaspread[c]);
            if (momntm[c] > 1) {
                momntm[c] = 1;
            }
            c_a[c] = randomNumber(camin[c], caspread[c]);
            c_r[c] = randomNumber(crmin[c], crspread[c]);
        }
        active = 1;
    }

    public void profitAlter(double price, int c) {
        double diff, change, newprofit, gamma;
        diff = price - this.price;
        gamma = momntm[c];
        change = ((1 - (gamma)) * (beta[c]) * diff) + gamma * last_d;
```

```
        last_d = change;
        newprofit = ((this.price + change) / limit) - 1.0;
        if (job == SELL) {
            if (newprofit > 0.0) {
                profit = newprofit;
            }
        } else {
            if (newprofit < 0.0) {
                profit = newprofit;
            }
        }
        setPrice();
    }
    public void profitAlter(double price) {
    }
}
```

Source code 3: ZIP60Agent.java

```
package ZIP60;
(...)
//implements ZIP60 trading strategy
public class ZIP60AgentUtils extends BaseAgentUtils {
(...)
    public void shout_update(int deal_type, int status, int n_sell, BaseAgent[] sellers,
          int n_buy, BaseAgent[] buyers, double price) {
        double target_price;
        int c;
        Random generator = new Random();
        for (int s = 0; s < n_sell; s++) {
            if (status == DEAL) {
                if (sellers[s].price <= price) {
                    c = 0;
                    target_price = (price * (1.0 + generator.nextDouble() * sellers[s].c_r
                          [c])) + generator.nextDouble() * sellers[s].c_a[c];
                    sellers[s].profitAlter(target_price, c);
                } else {
                    if (deal_type == BID && sellers[s].willingTrade(price) == 0 && sellers
                          [s].active == 1) {
                        c = 1;
                        target_price = (price * (1.0 - generator.nextDouble() * sellers[s
                              ].c_r[c])) - generator.nextDouble() * sellers[s].c_a[c];
                        sellers[s].profitAlter(target_price,c);
                    }
                }
            } else {
                if (deal_type == OFFER) {
                    if (sellers[s].price >= price && sellers[s].active == 1) {
                        c = 2;
                        target_price = (price * (1.0 - generator.nextDouble() * sellers[s
                              ].c_r[c])) - generator.nextDouble() * sellers[s].c_a[c];
                        sellers[s].profitAlter(target_price,c);
                    }
                }
            }
        }
        for (int b = 0; b < n_buy; b++) {
            if (status == DEAL) {
                if (buyers[b].price >= price) {
                    c = 3;
                    target_price = (price * (1.0 - generator.nextDouble() * buyers[b].c_r[
                          c])) - generator.nextDouble() * buyers[b].c_a[c];
                    buyers[b].profitAlter(target_price, c);
                } else {
                    if (deal_type == OFFER && buyers[b].willingTrade(price) == 0 && buyers
                          [b].active == 1) {
                        c = 4;
                        target_price = (price * (1.0 + generator.nextDouble() * buyers[b].
                              c_r[c])) + generator.nextDouble() * buyers[b].c_a[c];
                        buyers[b].profitAlter(target_price, c);
                    }
                }
            } else {
                if (deal_type == BID) {
                    if (buyers[b].price <= price && buyers[b].active == 1) {
                        c = 5;
                        target_price = (price * (1.0 + generator.nextDouble() * buyers[b].
                              c_r[c])) + generator.nextDouble() * buyers[b].c_a[c];
                        buyers[b].profitAlter(target_price, c);
                    }
                }
            }
        }
    }
}
```

Source code 4: ZIP60AgentUtils.java

```
package ZIP60;
(...)
//maintains crossover between two parents
public class ZIP60Crossover extends AbstractCrossover<double[]>
{
    public ZIP60Crossover()
    {
        this(1);
    }
    public ZIP60Crossover(int crossoverPoints)
    {
        super(crossoverPoints);
    }
    public ZIP60Crossover(int crossoverPoints, Probability crossoverProbability)
    {
        super(crossoverPoints, crossoverProbability);
    }
    public ZIP60Crossover(NumberGenerator<Integer> crossoverPointsVariable)
    {
        super(crossoverPointsVariable);
    }
    public ZIP60Crossover(NumberGenerator<Integer> crossoverPointsVariable,
                          NumberGenerator<Probability> crossoverProbabilityVariable)
    {
        super(crossoverPointsVariable, crossoverProbabilityVariable);
    }
    @Override
    protected List<double[]> mate(double[] parent1,
```

```
                                double[] parent2,
                                int numberOfCrossoverPoints,
                                Random rng)
    {
        if (parent1.length != parent2.length)
        {
            throw new IllegalArgumentException("Cannot perform cross-over with different
                length parents.");
        }
        double[] offspring1 = new double[parent1.length];
        System.arraycopy(parent1, 0, offspring1, 0, parent1.length);
        double[] offspring2 = new double[parent2.length];
        System.arraycopy(parent2, 0, offspring2, 0, parent2.length);
        double[] temp = new double[parent1.length];
        for (int i = 0; i < numberOfCrossoverPoints; i++)
        {
            int crossoverIndex = (1 + rng.nextInt(parent1.length - 1));
            System.arraycopy(offspring1, 0, temp, 0, crossoverIndex);
            System.arraycopy(offspring2, 0, offspring1, 0, crossoverIndex);
            System.arraycopy(temp, 0, offspring2, 0, crossoverIndex);
        }
        for (int i = 0; i < 59; i++) {
            if (i % 2 == 0) {
                if (offspring1[i] + offspring1[i + 1] > 1.0) {
                    offspring1[i + 1] = 1 - offspring1[i];
                }
            }
        }
        for (int i = 0; i < 59; i++) {
            if (i % 2 == 0) {
                if (offspring2[i] + offspring2[i + 1] > 1.0) {
                    offspring2[i + 1] = 1 - offspring2[i];
                }
            }
        }
        List<double[]> result = new ArrayList<double[]>(2);
        result.add(offspring1);
        result.add(offspring2);
        return result;
    }
}
```

Source code 5: ZIP60Crossover.java

```
package ZIP60;
(...)
//maintains mutation of offspring
public class ZIP60Mutation implements EvolutionaryOperator<double[]> {
    private final NumberGenerator<Probability> mutationProbability;
    private int generation = 0;
    private double ms = 0.05;
    private double me = 0.0005;
    private double log10(double x) {
        return Math.log(x) / Math.log(10);
    }
    public ZIP60Mutation(Probability mutationProbability) {
        this(new ConstantGenerator<Probability>(mutationProbability));
    }
    public ZIP60Mutation(
            NumberGenerator<Probability> mutationProbability) {
        this.mutationProbability = mutationProbability;
    }
    public List<double[]> apply(List<double[]> selectedCandidates, Random rng) {
        List<double[]> mutatedPopulation = new ArrayList<double[]>(selectedCandidates.size
            ());
        for (double[] s : selectedCandidates) {
            mutatedPopulation.add(mutateArray(s, rng));
        }
        generation++;
        return mutatedPopulation;
    }
    private double[] mutateArray(double[] s, Random rng) {
        double[] temp = s.clone();
        double mg = Math.pow(10.0, (log10(ms) -
                ((generation * log10(ms / me)) / (999))));
        double range = mg - (-mg);
        for (int i = 0; i < s.length; i++) {
            if (mutationProbability.nextValue().nextEvent(rng)) {
                double fraction = (range * rng.nextDouble());
                double randomNumber = (fraction + (-mg));
                temp[i] += randomNumber;
                if (temp[i] < 0) {
                    temp[i] = 0;
                }
            }
        }
        for (int i = 0; i < 59; i++) {
            if (i % 2 == 0) {
                if (temp[i] + temp[i + 1] > 1.0) {
                    temp[i + 1] = 1 - temp[i];
                }
            }
        }
        for (int j = 0; j < temp.length; j++) {
            if (temp[j] > 1.0) {
                temp[j] = 1.0;
            }
            if (temp[j] < 0.0) {
                temp[j] = 0.0;
            }
        }
        return temp;
    }
}
```

Source code 6: ZIP60Mutation.java

```
package ZIP60;
(...)
//produces initial population for Genetic Algorithm optimization
public class ZIP60Factory extends AbstractCandidateFactory<double[]> {
    public double[] generateRandomCandidate(Random rng) {
        double[] cand = new double[61];
        int q = 10;
        for (int i = 0; i < 10; i++) {
            cand[i] = rng.nextDouble();
```

```
        }
        for (int i = 0; i < 9; i++) {
            if (i % 2 == 0) {
                if (cand[i] + cand[i + 1] > 1.0) {
                    cand[i + 1] = 1 - cand[i];
                }
            }
        }
        for (int i = 1; i < 6; i++) {
            cand[0 + q] = cand[0];
            cand[1 + q] = cand[1];
            cand[2 + q] = cand[2];
            cand[3 + q] = cand[3];
            cand[4 + q] = cand[4];
            cand[5 + q] = cand[5];
            cand[6 + q] = cand[6];
            cand[7 + q] = cand[7];
            cand[8 + q] = cand[8];
            cand[9 + q] = cand[9];
            q += 10;
        }
        for (int j = 0; j < cand.length; j++) {
            if (cand[j] > 1.0) {
                cand[j] = 1.0;
            }
            if (cand[j] < 0.0) {
                cand[j] = 0.0;
            }
        }
        cand[60] = rng.nextDouble();
        return cand;
    }
}
```

Source code 7: ZIP60Factory.java

```
package baseagent;
(...)
//executes whole experiment
public class ExpExecution implements CtlAgentInterface {
    public static double Exps(double[] agentInput, int n_exps, ExpCtl expctl,
          BaseAgentUtils bau, Vector<Double> prices) {
        double expResult = 0;
        int n_trans, s, b, eq;
        int status = 0;
        int n_buy, n_sell, n_trades;
        int max_trades = 0;
        int ats_n[] = new int[MAX_TRADES];
        int dummy_i = 0;
        double price = 0, p_0 = 0, sigmasum, alpha = 0, ep, last_price = 0, sum_price_diff
            , dummy_r1 = 0, dummy_r2 = 0;
        double sum_price, diff, pd, pdisp, pds, alphatrans;
        double ats[] = new double[MAX_TRADES];
        double bounddata[] = new double[4];
        double max_surplus = 0, surplus, efficiency = 0;
        String fname;
        DayData[] ddat = new DayData[MAX_N_DAYS];
        TradeData[][] tdat = new TradeData[MAX_N_DAYS][MAX_TRADES];
        for (int i = 0; i < MAX_N_DAYS; i++) {
            ddat[i] = new DayData();
            for (int j = 0; j < MAX_TRADES; j++) {
                tdat[i][j] = new TradeData();
            }
        }
        BaseAgent[] buyers = new BaseAgent[MAX_AGENTS];
        BaseAgent[] sellers = new BaseAgent[MAX_AGENTS];
        for (int t = 0; t < MAX_TRADES; t++) {
            ats_n[t] = 0;
            ats[t] = 0.0;
        }
        int tr = 0;
        for (int e = 0; e < n_exps; e++) {
            bau.buy_init(buyers, agentInput);
            bau.sell_init(sellers, agentInput);
            for (int d = 0; d < expctl.n_days; d++) {
                max_trades = expctl.max_trades;
                double[] ars = Smith.day_init(e, d, expctl, sellers, buyers, p_0,
                      max_surplus);
                p_0 = ars[0];
                max_surplus = ars[1];
                n_buy = expctl.dem_sched[expctl.d_sched].n_agents;
                n_sell = expctl.sup_sched[expctl.s_sched].n_agents;
                surplus = 0.0;
                n_trades = 0;
                sigmasum = 0.0;
                sum_price = 0.0;
                sum_price_diff = 0.0;
                int t;
                for (t = 0; t < max_trades; t++) {
                    double[] art = Smith.trade(tdat[d][t],bau, sellers, buyers, expctl,
                          max_surplus, surplus, status, agentInput[agentInput.length-1],
                          prices);
                    surplus = art[0];
                    status = (int) art[1];
                    if (status == DEAL) {
                        if (t > 0) {
                            last_price = price;
                        }
                        price = tdat[d][t].deal_p;
                        tr++;
                        if (t > 0) {
                            sum_price_diff += ((price - last_price) * (price - last_price)
                                  );
                        }
                        pds = ((price - p_0) * (price - p_0));
                        n_trades++;
                        sum_price += price;
                        sigmasum += pds;
                        alpha = (100 * Math.sqrt(sigmasum / n_trades)) / p_0;
                        efficiency = (surplus / max_surplus) * 100;
                    } else {
                        if (status == END_DAY) {
                            t = max_trades;
                        }
                    }
                }
                pd = 0.0;
                for (b = 0; b < n_buy; b++) {
                    diff = (buyers[b].a_gain) - buyers[b].t_gain;
```

```
                    pd += diff * diff;
                }
                for (s = 0; s < n_sell; s++) {
                    diff = sellers[s].a_gain - sellers[s].t_gain;
                    pd += diff * diff;
                }
                pdisp = Math.sqrt((1 / ((double) (n_buy + n_sell))) * pd);
                ddat[d].ddatUpdate(n_trades, sum_price, alpha, pdisp, efficiency,
                    sum_price_diff);
            }
        }
        expResult = DayData.showStats(ddat, expctl.n_days, n_exps);
        return expResult;
    }
}
```

Source code 8: ExpExecution.java

```
package baseagent;
(...)
//stores experiment control data
public class ExpCtl implements CtlAgentInterface{
    public class Agent_sched implements CtlAgentInterface{
        int n_units;
        double limit[] = new double[MAX_UNITS];
    }
    public class SD_sched implements CtlAgentInterface{
        int n_agents;
        int first_day;
        int last_day;
        int can_shout;
        Agent_sched agents[] = new Agent_sched[MAX_AGENTS];
    }
    public String id;
    public int n_days;
    public int min_trades;
    public int max_trades;
    public int random;
    public int nyse;
    public int n_dem_sched;
    public SD_sched[] dem_sched = new SD_sched[MAX_SCHED];
    public int d_sched;
    public int n_sup_sched;
    public SD_sched[] sup_sched = new SD_sched[MAX_SCHED];
    int s_sched;
}
```

Source code 9: ExpCtl.java

```
package optimframework;
(...)
//executes proper number of optimizations
public class gaMain implements CtlAgentInterface {
    public static void gaExec(String[] args) {
        if (args.length > 8) {
            System.out.println("There should be 8 parameters specified.");
            System.exit(1);
        }
        StoringDataS3 sds3 = new StoringDataS3();
        ExpCtl expctl = XMLReader.expctl_in(args[7], 1);
        long start = new GregorianCalendar().getTimeInMillis();
        double[] result = null;
        Random generator = new Random();
        int expBlockID = generator.nextInt(1000);
        if (Integer.parseInt(args[0]) == 1) {
            for (int i = 0; i < Integer.parseInt(args[5]); i++) {
                System.out.println("\nZIP8: Executing experiment number " + i);
                result = new EvolutionaryATS().createEvolution(Integer.parseInt(args[2]),
                        Integer.parseInt(args[3]), Integer.parseInt(args[4]), expctl, sds3
                        ,
                        Integer.parseInt(args[1]), new ZIP8AgentUtils(), new ZIP8Crossover
                        (),
                        new ZIP8Mutation(new Probability(1)), new ZIP8Factory(), Integer.
                        parseInt(args[6]));
                System.out.println("Resulting parameters are: ");
                for (int j = 0; j < result.length; j++) {
                    System.out.println(result[j] + " ");
                }
                sds3.storeExperimentData("ZIP8", result, expBlockID);
            }
        } else if (Integer.parseInt(args[0]) == 2) {
            for (int i = 0; i < Integer.parseInt(args[5]); i++) {
                System.out.println("\nZIP60: Executing experiment number " + i);
                result = new EvolutionaryATS().createEvolution(Integer.parseInt(args[2]),
                        Integer.parseInt(args[3]), Integer.parseInt(args[4]), expctl, sds3
                        ,
                        Integer.parseInt(args[1]), new ZIP60AgentUtils(), new
                        ZIP60Crossover(),
                        new ZIP60Mutation(new Probability(1)), new ZIP60Factory(), Integer
                        .parseInt(args[6]));
                System.out.println("Resulting parameters are: ");
                for (int j = 0; j < result.length; j++) {
                    System.out.println(result[j] + " ");
                }
                sds3.storeExperimentData("ZIP60", result, expBlockID);
            }
        } else if (Integer.parseInt(args[0]) == 3) {
            System.out.println("\nAA: Support under construction.");
        }
        long end = new GregorianCalendar().getTimeInMillis();
        System.out.println("\nTotal execution time = " + (end - start) + " milliseconds.")
            ;
        System.exit(0);
    }
}
```

Source code 10: gaMain.java