Bing Forecast

Thesis BSc. Project IN3405

Roy de Bokx (1515624)
Leendert Gravendeel (1514180)
Maikel Krause (1511475)

Public version

Delft University of Technology
EEMCS Faculty
July 8th, 2011

Exam committee

Bing Technology LLC:

Dan Guy, CEO at Bing Technology

Delft University of Technology:

Ir. B.R. Sodoyer

J. de Vries

Preface

This document is the result of three months of work on our BSc. project, concerning the creation of a stock market forecasting application for Bing Technology, LLC. We consider ourselves very lucky to have found a host allowing us to work on this project for two months on location near Philadelphia, USA.

We would like to thank Laurens Swinkels, of Robeco Investment Solutions, for giving us a head start by providing us with some rudimentary general knowledge of stock market forecasting, and for giving us advice when we needed it.

We would also like to thank Augie Chung, for giving technical advice on difficult Java-related issues.

Finally, we would like to thank Dan Guy in particular for supporting us in many ways.

Summary

Bing Technology, a Philadelphia, USA based software company, seeks to develop a software framework that can be used to create forecasts for a wide range of predictive domains. In particular, they would like to create an application of this framework that is able to perform stock market forecasting.

The goal of our project is to develop an innovative algorithm to perform data prediction, and to apply this algorithm in a stock market forecasting application. The application should generate investment strategies which can be evaluated to output an optimal investment portfolio.

We have achieved this goal by making use of a variant of Genetic Programming to create strategies which are represented internally by a tree containing "modules". Each module performs a specific function; examples include simple numerical parameters, statistical functions and technical indicators.

The backbone of the application is written in Java, communicating with a web-based PHP front-end through a MySQL database. The front-end allows the user to create jobs for the back-end to process, as well as view the resulting strategies and related statistics.

Test runs have shown a correlation between in-sample and out-of-sample performance. Further, we have determined with high certainty that there is a predictable relation between the complexity of a test run and the duration of the run.

There are several features and changes we would like to see in future development of the product. Several program parameters can be optimized further, and there are more modules to implement that could boost the performance of the generated strategies. Further, the data provided to the modules can be improved by including qualitative data about the company, or industry aggregative data. Other improvements could come from implementing support for short selling and more accurate transaction costs. On the user side, we would like to give investors more control over the strategy building process by allowing them to create templates on which new strategies have to be based.

Table of Contents

1. Introduction

Mid January 2011, Leendert received an assignment from Bing Technology, a company Leendert and Roy had worked for in the past. Bing sought to develop a framework that could be used to produce specialized forecasting tools useful across industry verticals. Initially, they were seeking to target two specific domains: stock market prediction and the Heritage Health Prize. Leendert then approached Roy, who was already planning to do his BSc. project together with Maikel. This resulted in a group of three students who were to travel to the US and write software to predict the future, under the supervision of Dan Guy.

We started off doing a lot of research in The Netherlands concerning existing stock market prediction methods as well as the possibilities and facilities we were planning to use in the US. When we were all set to go, we went to Conshohocken, Pennsylvania, where the project began.

This document describes the product that was developed during the project. In chapter 2 we start by describing and analyzing the problem to be solved. Next, the product itself is described according to the design in chapter 3, and the implementation, in chapter 4, followed by several findings in chapter 5. Finally we will make some recommendations for the future development and use of Genetic Programming in the field of stock market forecasting in chapter 6.

2. Problem Definition and Analysis

When Bing Technology approached us, they sought to develop a framework that could be used to produce specialized forecasting tools useful across industry verticals. Initially they were seeking to target stock market prediction and the Heritage Health Prize. Targeting the stock market would come down to developing an application of the forecasting framework which could generate a trading strategy which should perform better than other currently known trading strategies. The Health Heritage Prize is a competition to develop an algorithm to predict and prevent unnecessary hospitalizations in the US, based on patient data of the past. This goal is to create an algorithm that predicts how many days a patient will spend in the hospital in the next year.

Bing sought a framework that could ideally be repurposed for other tasks such as product recommendations and other predictive domains. This would happen in a possible follow-up project. Additionally, Bing would prefer that the back-end be written in Java. PHP would be acceptable for the front-end application. They desired having a functional proof of concept by May 24 2011, with further iterations delivered as needed depending on efficacy of the prototype and future arrangements between the selected vendor(s) and Bing.

Before arriving in the US, a great amount of research was done, including an interview with Laurens Swinkels, vice president of Robeco Investment Solutions. This research helped us get a better view of the environment in which the applications had to run. The next step was to create a design which would give us the stability and flexibility to meet the demands of Bing Technology. After some research was done on the scope of this assignment, followed by some drafts on the design of it, it turned out that it would not be feasible to make a reusable forecasting framework and an application of this framework for the stock market forecasting within the limited time constraints of the Bachelor project. Therefore, it was decided to develop a stock market forecasting application for the proof of concept, which could, if necessary, be transformed into a forecasting application for some other domain later on.

After this was decided, a design had to be made for the stock market forecasting application. Because of the scope and complexity of the existing prediction algorithms, it was decided to create a modular system, which would have a front-end in PHP and a back-end in Java to meet the demands. The back-end would contain all the functionalities to create and evaluate trading strategies. To create these strategies, Genetic Programming would be used to combine the different modules. These modules would be obtained by breaking down existing trading strategies to their granular functionalities, which would then be implemented in Java. These modules would be represented by a node which can specify constraints on its child nodes. By genetically constructing a tree out of these different nodes not only existing strategies but also new strategies can be constructed.

The collection of all these node types should facilitate the creation of modules that implement all functionalities that are represented in currently known trading strategies. It should also facilitate the creation of new trading strategies, with additional modules that implement new functionalities if required.

However, the danger of this approach is overfitting. The ability to create new functionalities which are to be tested on historical data, in order to perform well in the future, increases the chance that a strategy will be constructed which does very well on the data which it will be tested on, however it will perform bad on future data. It is therefore important to only implement a module if there are good reasons to do so, instead of implementing functionalities just for the sake of it. This way, overfitting is prevented by design.

However, it is also very important to use in-sample and out-of-sample testing, because this is actually the most commonly used method to measure overfitting. If the constructed strategy performs

very well on the in-sample test data, but very poor on the out-of-sample test data, chances are high that the constructed strategy suffers from overfitting.

3. Design

To achieve the goal of outperforming all other known trading strategies, a solid design is needed. A design that is solid, however not rigid, since the application will likely need to be altered in the future to adapt to new market trends. Therefore, it was decided to design the core of the application as a modular system, which would consist of a part that would genetically generate strategies, a part that would evaluate them and a part that would contain all the modules that could be used for constructing the trading strategies.

3.1 Data Design

First of all, it is useful to begin by looking at the data objects implicitly mentioned in the previous chapter. This is to prevent any misunderstandings or ambiguities when describing the design and implementation of the application.

3.1.1 Strategies

From a high-level point of view, the main task of the application is to create new investment strategies. Such a strategy is essentially an algorithm which can tell an investor which stocks to buy or sell, and how much. Said another way, a strategy should be able to predict the optimal stock portfolio composition for the next day.

Most stock traders use investment strategies intuitively all the time., however it is needed to turn these intuitive notions of a "strategy" into some data structure that can be manipulated by software. This data structure needs to facilitate the following operations:

- Constructing new strategies completely automatically, based on any sample data we can provide, such as historical market data, company profit results, or even Twitter feeds.
- Evaluate a strategy for the current date, telling the user how to compose his portfolio in order to maximize the portfolio's value.

The most logical representation of a strategy – which is actually an algorithm – is a syntax tree, since the field of Machine Learning has much experience in creating optimal algorithms represented as trees by the technique of Genetic Programming. A slight twist in our approach to Genetic Programming is that the basic elements that our strategies will consist of will not be elementary operations which all require and return values of the same type. This could cause too much risk of data-fitting, and further would result in very complex, black box algorithms.

Instead, our approach was to break down existing, proven strategies in such a way that they would be reusable for constructing new strategies, where overfitting would be prevented by design. Each of these basic block were implemented as *modules* in the application, instances of which can be placed in a strategy's tree, resulting in a strategy that could be evaluated on a day-by-day basis.

3.1.2 Modules, Nodes and Node Types

Modules are the basic primitives with which the strategies will be constructed. These modules are specific to the problem domain, and allow the forecasting algorithm to be easily reused for other domains or other types of input which could vary from historical data of stock markets to Twitter feeds.

Modules often require some parameters, which will then be obtained by their children. Each module can therefore put constraints on the amount, order and types of its children. These node types are, as mentioned before, obtained during the breakdown of existing strategies and can be found in appendix D, followed by a graphical representation in appendix E.

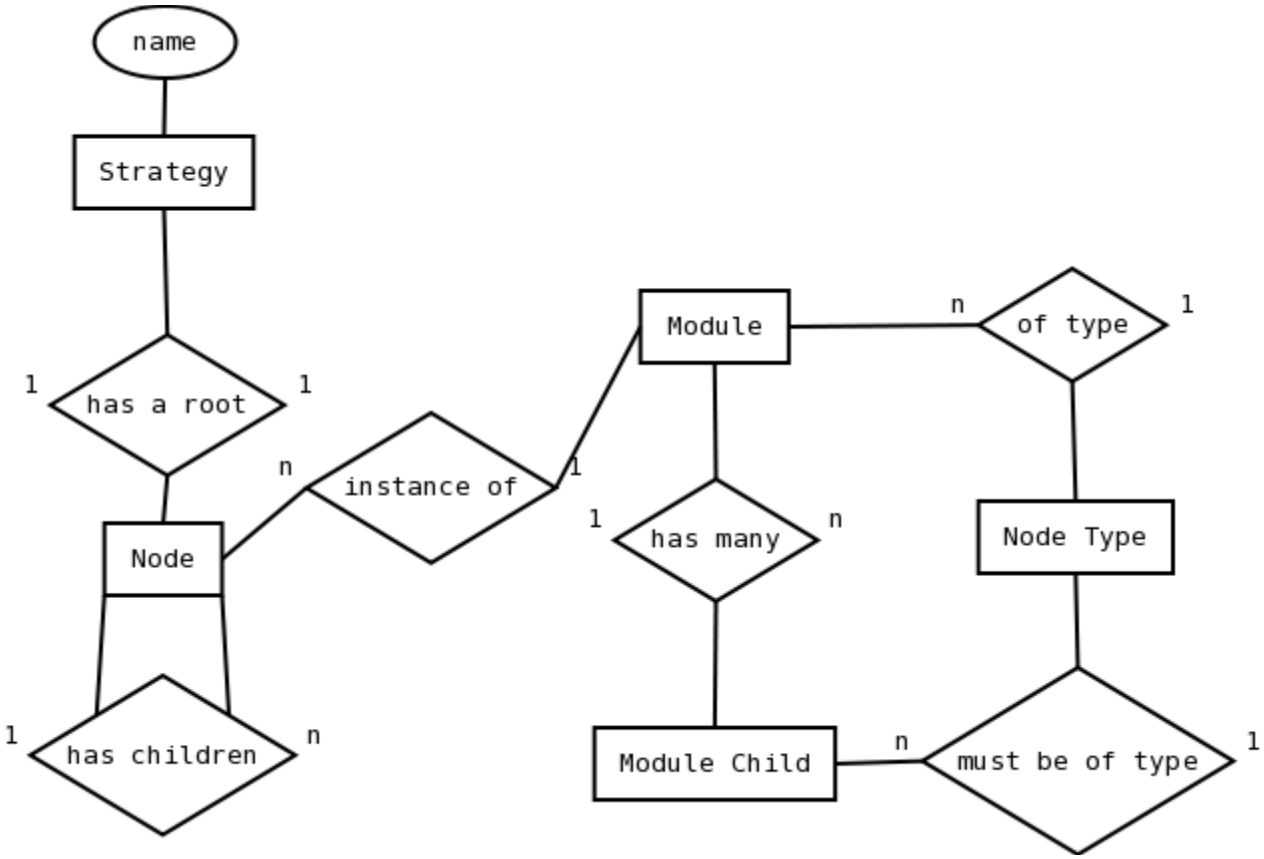


Figure 3.1 Relationships between strategy-related objects.

3.2 The Forecasting Algorithm

As stated before, our approach to forecasting is to use an innovative variation on Genetic Programming. The algorithm builds a program that can be evaluated through time to ultimately propose a portfolio composition for every single day. Using Genetic Programming to forecast stock markets is not really something new, however so far most approaches used Genetic Programming to combine several Buy/Sell/Hold (BSH) indicator nodes. Our approach is different, combining different functionalities instead which have different input and output data types. The exact implementation of our approach will be described in paragraph 4.2.

3.3 Architecture

The application is split up into two parts: the back-end, written in Java, and a web-based front-end, written in PHP. This is done because Java is more suitable for developing complex and heavy duty applications. However, for displaying the results through a web interface, which is the standard at Bing Technology, PHP is more suitable. All communication between the front-end and the back-end flows through a shared database. This setup will also allow us to execute all CPU intensive jobs on a big server while the system can still be controlled from a portable interface.

The back-end is a command line Java program that will run continuously on the server. The back-end will first poll for any jobs that are submitted into the database by the front-end. Next, it will execute all calculations necessary to provide the front-end with the desired information such as generated strategies and results of the evaluations of these strategies. Note that the evaluation of strategies will be done by an evaluator which will continuously evaluate strategies that are not

evaluated during the last 24 hours. These results will then again be submitted to the database so they can be retrieved by the front-end. All these different tasks are assigned to separate packages which can be found in appendix E.

The front-end will serve as the user interface to the entire application. The user will be able to submit new jobs for tasks such as creating new strategies and instantiating generic strategies. After these jobs are executed by the back-end, the user will be able to review the generated results such as the structure of the generated strategies, as well as the performance of these strategies.

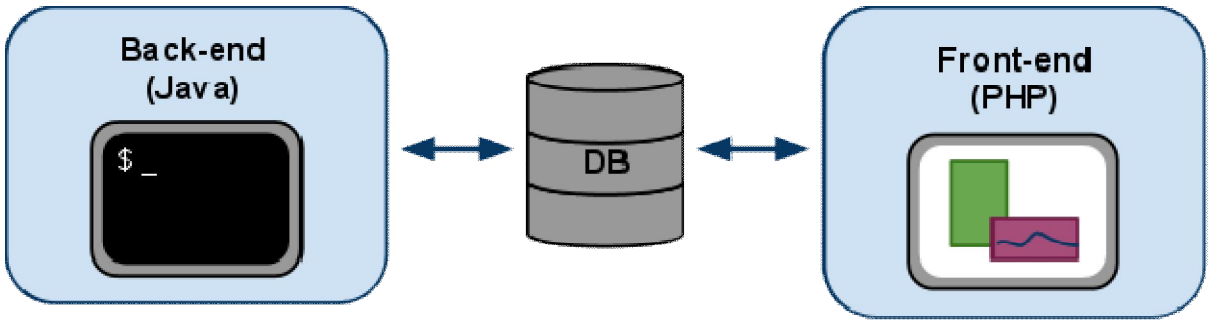


Figure 3.2 Back and Front-end architecture

3.4 Database Design

The application needs a somewhat extensive database. First of all, it needs to facilitate submitting tasks for generating and instantiating strategies at the front-end. These tasks have to be executed by the back-end which then needs to store the genetically generated strategies in the database so they can be executed every day.

Second, the results of the strategies have to be calculated by the back-end, stored in the database and viewed by the front-end. To make all these functions possible, a database design was composed which can be found in appendix G.

3.5 Interface Design

On implementing an application, an interface is also to be designed, so the application can actually be executed and used by the user. In this paragraph, the back-end and front-end interface is discussed. For the back-end this is done by describing how to start the application on the server, as the actual interaction with the back-end will from there on be done by the front-end interface. The front-end interface will then be described by mock-ups, showing what functionalities can be found on the several pages the web interface provides.

3.5.1 Back-end

The back-end is entirely command line based, and does not provide a graphical user interface. The program accepts the following command line arguments:

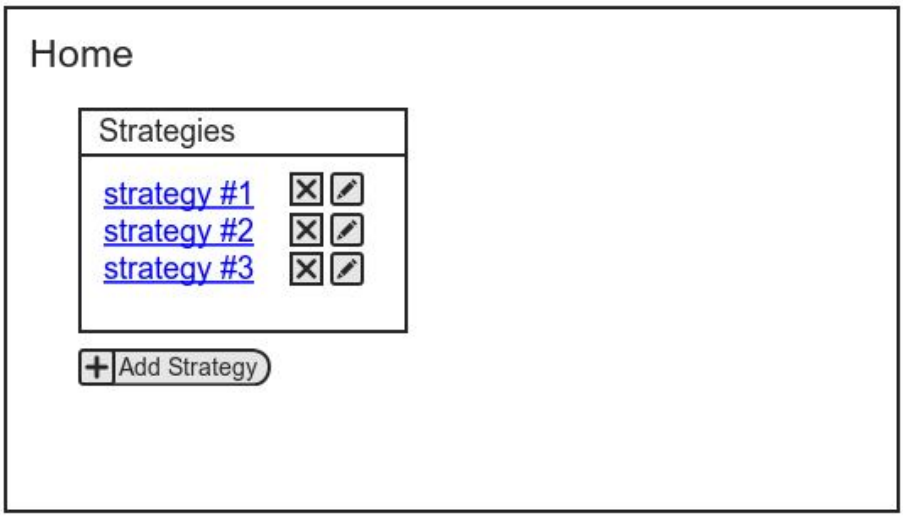
- Database name
- Database user
- Database password
- Number of threads

The program will run in the background and automatically polls for new jobs to perform.

3.5.2 Front-end

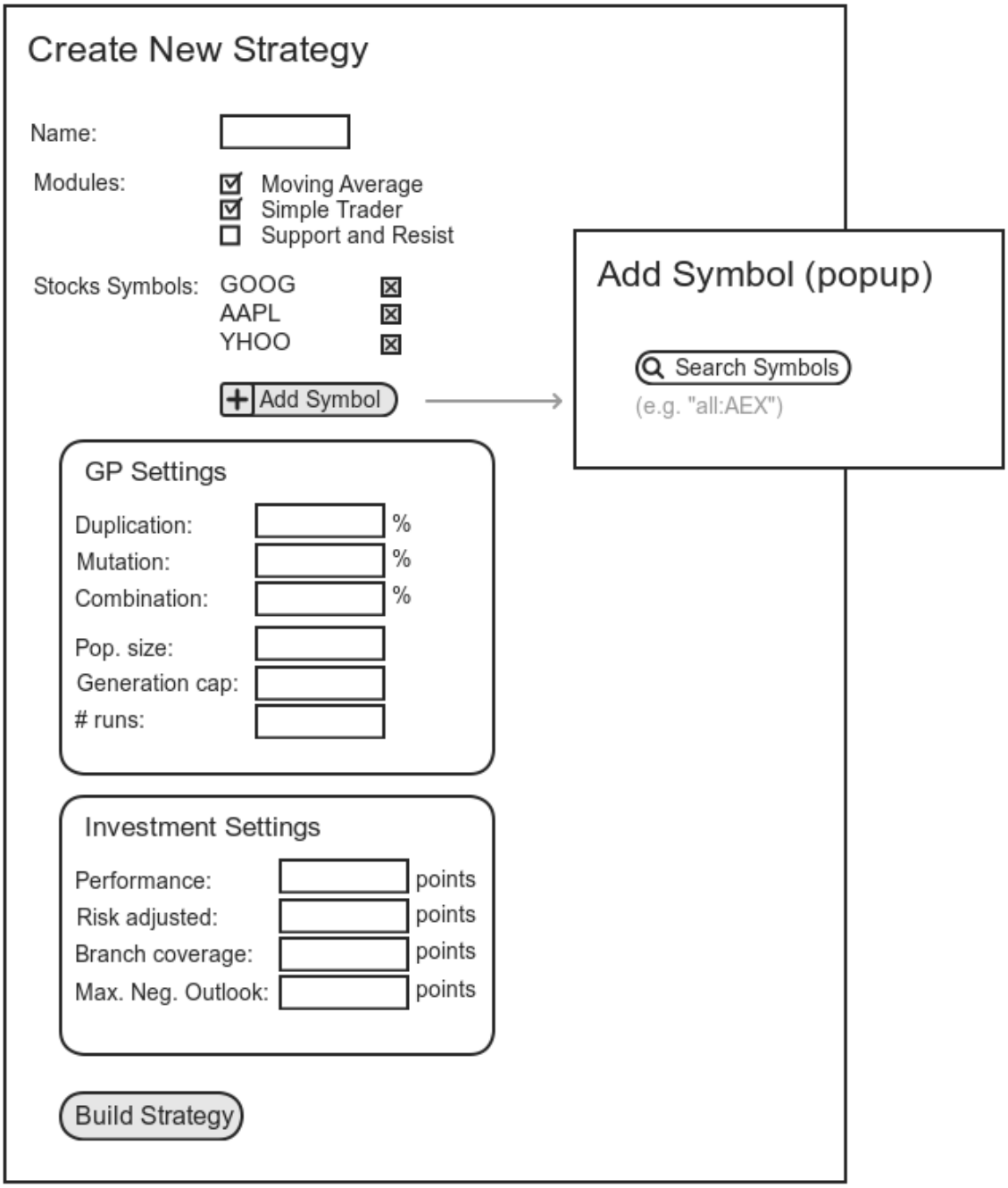
The front-end is a web-based application that allows users to request the generation of new strategies and then run these for the current date. It also allows users to view the details of a generated strategy and get its performance data.

Home



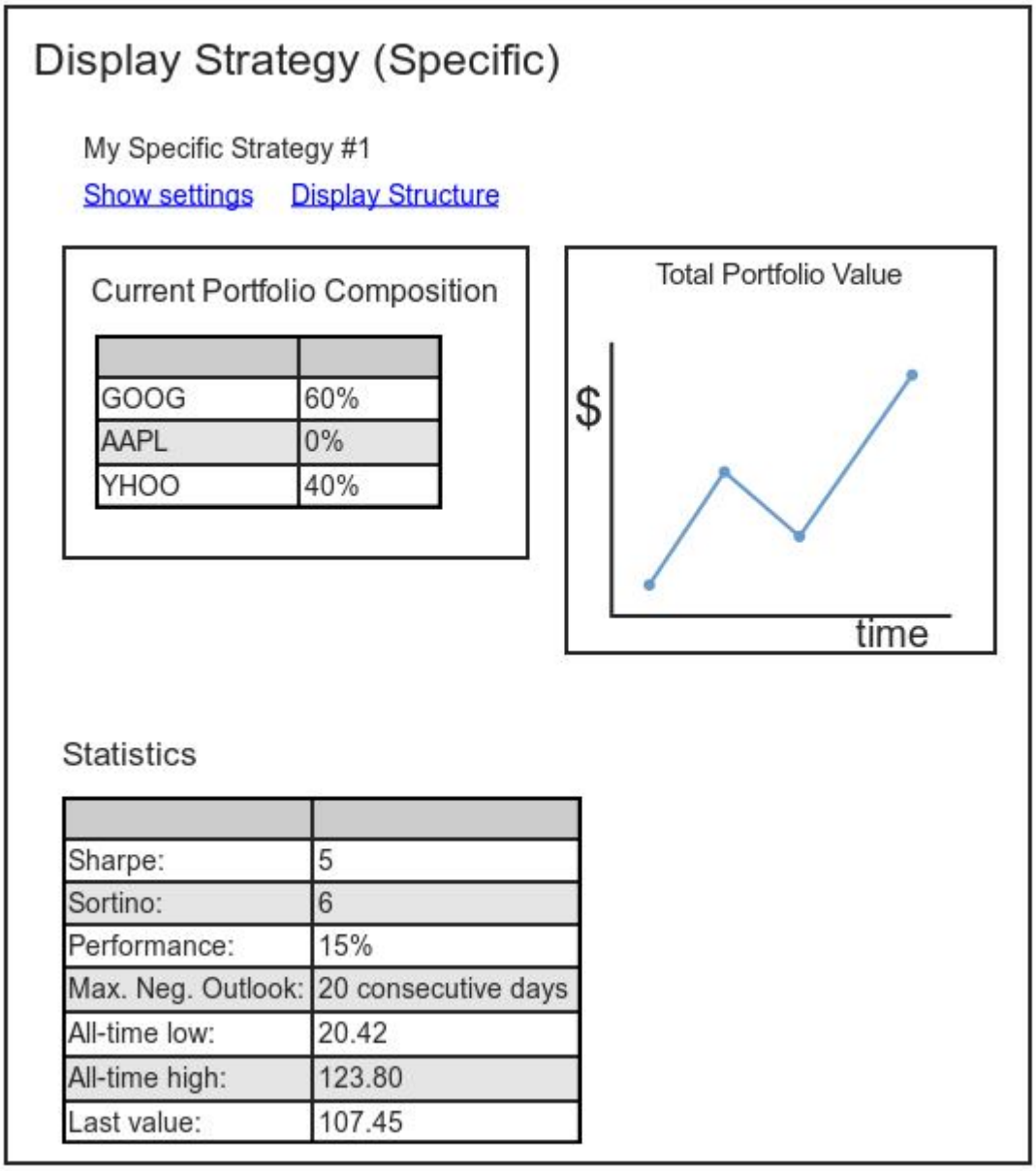
The Home screen is very simple, and simply gives the user an overview of his strategies, as well as providing standard CRUD (Create, Read, Update, Delete) actions.

Create Strategy



Clicking the "Add Strategy" button on the Home screen takes the user to the Create Strategy screen, which is a form where the user can input various settings, such as GP parameters and investment preferences.

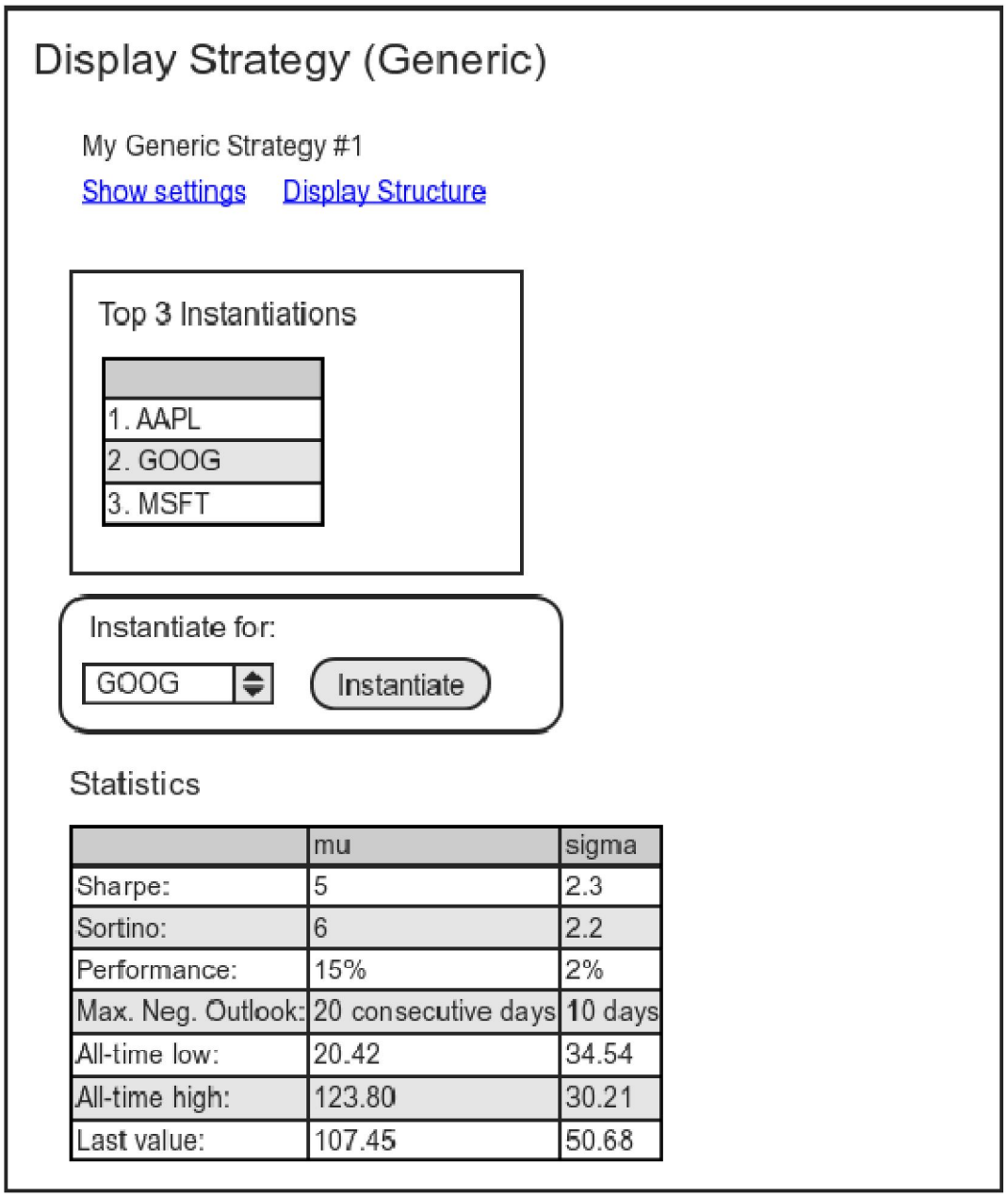
Display Strategy (Specific)



Clicking the name of a specific strategy on the home screen takes the user to the Display Strategy (Specific) screen. Specific and generic strategies have different properties, and the corresponding "display" screens are therefore separate screens.

This screen shows basic information about the strategy such as its settings as well as the current portfolio composition advice and the strategy results over time. It also displays some statistics such as the Sharpe ratio of the strategy.

Display Strategy (Generic)



The generic version of the Display Strategy screen is similar to the specific one, but shows averaged data over all allowed instantiations. The amount of allowed instantiations is equal to the amount of allowed symbols to be used in the job; for each allowed symbol one specific strategy is created and simulated.

4. Implementation

In this chapter, the implementation process of the forecasting project is described, as well as the decisions that were made during the implementation phase. As mentioned in the Plan of Approach, a flat hierarchy was used during the project, assigning tasks to each other on the fly. This resulted in a division of labor in which, roughly, Maikel took care of the front-end, and Roy and Leendert implemented the majority of the back-end.

4.1 Strategy Modules

As mentioned in the previous chapter, the strength of the application is in the breakdown of existing strategies. The node types which were mentioned in this chapter should facilitate the breakdown and modelling of existing strategies in modules which can then be used for the construction of new strategies. By breaking down several existing strategies, as well as doing research on existing indicators and heuristics, many modules were obtained and implemented, which can all be found in appendix A.

The process of breaking down existing strategies is focused around trading strategy interfaces, both internal and external. All trading strategies have some form of inputs and outputs, which can be used to categorize them. In turn trading strategies also use different data types internally. For instance, a trading strategy that takes a long position in the stock with the highest value of the closing price multiplied by the daily traded volume can be seen as an object that takes a date as input and returns one selected symbol as output.

Internally, the strategy could be broken down as well. First of all, the strategy contains a component that is a set of symbols from which the best one is selected. For this sample strategy we assume the stock set remains the same through time, meaning that particular component does not have any inputs and returns a set of stocks as output.

A second component in the sample strategy could be an object that takes a stock and date as input and returns a "score" for that particular stock on that particular date. In the case of the sample strategy, it could be calculated by looking up the closing price and daily traded volume on the specified date and multiplying both to get a resulting score. This component could be seen as a stock evaluator, of which more similar components can exist throughout the program, with different formulas to calculate the score for a given stock on a given date.

The last component in the sample strategy is the top object, connecting both the set of stocks and the stock evaluator. It takes a date, but also takes two or more components as inputs, namely an element that can return a set of stocks without any input and an element that can return a score when given a stock and a date. This top node can now be used to construct the sample strategy, but besides that it can also be used to create similar but slightly modified strategies.

Alternative strategies could now be generated by swapping out the required components for different implementations that use the same interfaces. If another stock evaluator component were to come along having the same exact signature as the stock evaluator used in the sample strategy, it could be used by the top node to create an entirely new strategy with a very similar structure. An example of such a different stock evaluator component would be a component that calculated the difference between the open and close prices of a certain stock, and multiplied that by the traded volume of that day.

This method of analysing strategies, breaking them down into components, grouping all created components by their interface and having all components of the same interface select their own set of required child interfaces was used on many existing strategies. The amount of different interfaces used was relatively small, allowing for great substitutability within the created strategies, in turn allowing for more diversity in possible output strategies.

While a greater amount of modules results seems better, it is also important not to overdo it as this allows for too much overfitting in the strategies. An example of this can be found in the described sample strategy, of which the module that multiplied the stock price by the volume could still be broken down further. A possible way to break it up would be to create a top node that multiplies two numerical values and two more modules that can get the stock price and the volume. This way of breaking up the module into smaller components would be a good example of going too far in the process of modular decomposition. The reason why this would go too far is that it would allow for substitution of any number used in existing strategies by a multiplication of that number with any other number. If the amount of modules that return a number becomes great enough, this multiplication module would become the gateway for overfitted strategies.

To summarize, using this method the interfaces, referred to as "node types", specify the output of strategy components and their required inputs for the evaluate function. The constraints that are put on the children of such a node type depends on the implementation of this evaluation function however. This means any component of a certain node type has the same signature, which for instance could always return an integer if a date is given as input. The components, referred to as "modules", could then specify the amount of children they require and their node types.

4.2 Genetic Programming

As mentioned in chapter 3, Genetic Programming is used to construct trees that represent trading strategies. With the basic ingredients such as node types and modules, as discussed above, a start could be made on actually constructing an algorithm that would genetically construct these trees. In this paragraph our implementation of Genetic Programming is described, as it differs from the usual concept of Genetic Programming on some aspects.

First of all, Genetic Programming has already been used in this field in several scenarios. However, usually the goal of this genetic process was to find the ultimate combinations of indicators. This would mean that all nodes in the constructed trees would have the same input and output except for the leaves which would base their output on historical data. In our implementation of Genetic Programming, we use different node types of which the implementations, also known as modules, may put different constraints on their children, as described in chapter 3. The construction of these trees was done by starting with a generation of random generated trees, which are then duplicated, mutated and combined towards the optimal trading strategy.

Second, our genetic process has a different stop criterion than common genetic processes. Usually, a genetic process would stop when the overall improvement of the last x generations does not meet a certain boundary value. In our implementation, the amount of generations is fixed because of the limited time that was available for this project. The advantage of this approach is that for the performance breakdown, the required time for a certain job, considering the hardware of the machine it is running on, can quite easily and accurately be predicted. However the disadvantage is that a genetic process might be stopped while still making significant improvements between consecutive generations, which means that some testing needs to be done on the ideal amount of generations.

4.2.1 Parameters

Constructing a strategy is initiated by the front-end, where the user inserts the parameters used for the construction. These parameters include the following:

- Input parameters
  - Modules: the modules that are allowed to be used.
  - Symbols: the symbols on which the performance of the constructed trees are verified.
  - Transaction costs ratio: the ratio of the transfer price that a transaction on the stock market costs.
- GP parameters
  - Duplication Rate: the chance that during the genetic construction of new generations any tree will be duplicated.
  - Mutation Rate: the chance that during the genetic construction of new generations any tree will me mutated.
  - Combination Rate: the chance that during the genetic construction of new generations any tree will be combined with a part of another tree.
  - Population Size: the size of the generations during the genetic construction.
  - Generation Amount: the amount of generations that were constructed during the genetic process.
  - Run Amount: the amount of separate runs that should be executed, based on these settings.
  - In Sample Start Date: the start date of the period on which the constructed trees can base their fitness.
  - In Sample End Date: the end date of the period on which the constructed trees can base their fitness.
- Output parameters
  - Performance Weight: the weight of the performance (excess returns) in the fitness function
  - Sharpe Weight: the weight of the Sharpe ratio in the fitness function
  - Sortino Weight: the weight of the Sortino ratio in the fitness function
  - Branch Coverage Weight: the weight of the branch coverage in the fitness function
  - Maximum Negative Outlook Weight: the weight of the Maximum Negative Outlook (a quantification of the duration of a period in which a strategy does not perform well) in the fitness function.
  - Max Nodes: the maximum amount of nodes. If a tree was constructed with more nodes, it will not be discarded, however it will get a penalty on its fitness, which decreases its chances of survival.
  - Max Symbols: the maximum amount of symbols. If a tree was constructed with more symbols, it will not be discarded, however it will get a penalty on its fitness, which decreases its chances of survival.

4.2.2 Process

Next, these settings, submitted as a job in the database by the front-end, will be retrieved by the Poller of the back-end, after which this Poller will create a JobRunner that will actually execute the job.

However, before the genetic process will start, the application will first check whether the job is feasible. During this check all symbols that do not have historic data for the specified in-sample period will be excluded from the process. After this, a check is performed to check if it is actually possible to construct trees with the given parameters. This means that there has to be at least one stock symbol left to base the tree on and for all selected modules of the type Portfolio, a complete and correct tree can be constructed with the help of the other modules that were allowed for this job.

After the JobRunner has determined that the job is feasible, the genetic process begins. According to the principle of Genetic Programming, the process starts with the construction of a randomly generated generation. This is done by the GenerationFactory by acquiring strategies that were produced by the StrategyFactory. This StrategyFactory constructs these strategies by randomly picking a portfolio module from the provided pool of allowed modules. From here on, a tree was constructed by selecting random children for this portfolio, the children of the portfolio and so on, according to the constraints each node puts on its children. This generation of strategies is repeated until a collection of trees was constructed that has a certain size, defined by the job parameters.

After this first generation was made, all strategies in this generation are evaluated and ranked. This is done according to an evaluation on historic data, based on the fitness function.
After the construction and ranking of this first generation, an iterative process starts that is repeated a certain amount of times, defined by the job parameters. In this process, new generations are constructed based on the previous ones to converge towards the ultimate trading strategy.

To construct a generation based on the previous generation, it is a requisite that this previous generation was ranked according to the fitness function. Then, a tree was randomly picked from this previous generation, according to its fitness. This random selection of strategies was implemented by a WeightedRandomPool, of which you can get a random element in O(log(n)) time, where elements with a bigger weight have a higher chance to be picked.

4.2.3 Reproductive Operations

When a tree is picked from this WeightedRandomPool, it is either duplicated, mutated or combined by the StrategyFactory. Chances for each genetic operation are determined by the job parameters. These steps are repeated until a new generation is created with duplicated, mutated and combined strategies that has a certain size that was specified by the job parameters, after which all the strategies in the new generation are evaluated and ranked according to the fitness function.

When a tree is duplicated, a duplicate of the tree is made where all nodes are reinitialized, since most module implementations are stateful. This duplicate is then placed in the next generation.

When a tree is mutated, however, a duplicate is created on which a mutation is performed. This is done by randomly picking a node from the tree, which is then deleted, together with its children. Next, a random tree is constructed of which the root node must have the same type as the deleted node. This tree is then placed in the duplicated tree at the position of the deleted node, which results in a new strategy which is then placed in the next generation.

Finally, there is a chance that the picked tree will be combined. This is done by randomly picking a node from the duplicate of the picked tree. This node will be deleted together with its children, after which a second strategy was picked from the previous generation, of which a random node will be chosen that has the same node type as the deleted node in the first tree. This node from the second tree, will then be duplicated, together with its children, and inserted in the first tree on the place where the randomly picked node was deleted. The result will then be added to the next generation.

4.2.4 Stop Criterion

The creation of new generation based on the previous generation is repeated for a certain amount of times, as defined by the job parameters. Usually a genetic process stops when the overall improvement of the last x generations does not succeed a certain boundary. In our implementation however, the amount of generations is fixed because it is quicker to implement and fine-tune. Considering the limited time that was available for this project, this seemed justifiable.

As mentioned before the advantage is that the required time for performing a job can be estimated and easily regulated, however the disadvantage is that several tests might be needed to determine the ideal amount of generations that are needed to get optimal results. If not enough generations are constructed it is possible that the process would be stopped while significant improvement was made on consecutive generations. That would mean that if more generations were allowed, the outcome of the process would quite likely be better. However if too many generations were used, it would be a waste of resources, which are quite important as will be discussed in paragraph 5.2.

After the iterative process is finished and the last generation was ranked, the best performing strategy of the last generation will be picked as the actual result of the genetic process. This strategy will then be stored in the database and evaluated for the out-of-sample period. Finally, the strategy can be showed to the user by the front-end, together with any information on the performance of the strategy during the in- and out-of-sample period. A graphical overview of this entire process can be found in appendix H.

4.3 Representation and Interpretation of Numeric Values

Many modules have one or more numeric children, which for instance can be used as the amount of days to look back for a moving average. The amount of days a moving average module is allowed to look back is not just any random number between 0 and the maximum amount of days of which data is available. If this were to be allowed, the risk of overfitting would increase drastically. For instance, if both a 123-day and a 124-day lookback period are allowed, the one with the best performance on the in-sample data would likely be selected eventually in the GP process. However, there is absolutely no guarantee that this choice will also yield the best results on the out-of-sample data.

To get around this issue of overfitting, we wanted to make sure only those lookback periods for a moving average could be selected that were commonly used or at least referenced by some academic papers. Lookback periods that were more popular got greater chance of being selected. We did not do this just for the moving average module, but for all modules that used numeric children on which references could be found.

To implement this functionality all numbers are represented as numbers between 0 and Java's maximum integer value. For each number that was required as a child of a module, a probability distribution function of all possible desired outcomes was constructed as an instance of the WeightedRandomPool class. We then used a form of inverse transform sampling to map the random number to a selection of this WeightedRandomPool.

As a result of this approach, the same input number will always resolve to the same interpretation of the number. Also, if the same WeightedRandomPool contains an interpretation of a number with a weight twice as high as another interpretation, the first interpretation will be twice as likely to be selected when a randomly generated strategy is interpreted.

4.4 Multithreading

The entire GP process is very expensive to run and is generally executed on big servers with many available cores. Without any further optimization it would not be possible to spread out the process to more than one thread, thus not fully optimizing the process to fully utilize all available cores. On the

other hand it would be much quicker to implement the program without multi-threading, as thread setup and possible race conditions would not have to be taken into consideration. We therefore chose to initially design the program in such a way that it would be easy to later extend it to use multi-threading but at first just use a single thread, in compliance with our incremental design pattern.

A new "Parallel" package was created for all tasks that could be parallelized, starting out with single thread implementations. The reason for this choice was the ease with which it could later be swapped for a package that used multi-threading. Inside this package is the class Dispatcher, containing the link between a single thread and multiple threads. Calls to its methods can be made from a single thread and it will ensure new threads are started, executed and joined. After the threads are joined the Dispatcher method will also ensure the results are being passed back to the single thread as if everything was executed in the same thread.

Three possible places of parallelization were found in the back-end. First of all the creation of a new generation consisting of randomly created strategies, secondly the creation of new generations based on previous generations and lastly the evaluation of a generation. Out of those three possibilities the last two were chosen to be implemented. The reason for not implementing parallelization of the creation of the first generation is that it is only called once per GP job and can run relatively quickly on one thread as well. On top of that our timeframe was rather limited forcing us to only focus on the most important aspects.

4.4.1 Creating Generations from Previous Generations

When creating a new generation from an old generation, a WeightedRandomPool containing all strategies from the previous generation is used. The actual creation of the required amount of new strategies is a task that can then easily be parallelized. On top of that, this task takes quite some time to execute for each job as it is responsible for the creation of (amount of generations - 1) * (population size) new strategies, which for an average job of 200 generations and a population size of 2500 already makes for 497,500 strategies.

The strategies that are being created for a new generation can be created independent of each other, meaning it can efficiently be parallelized by creating N new tasks for the creation of (population size / N) strategies each. After all of these tasks are then finished, a single thread can join all strategies into the new generation.

For this particular parallelization task, a new method was created with the following syntax:

```
public static Generation getGenerationFromPrevious(Job job,
      Generation previousGeneration)
```

The method will start out by creating N StrategyCreator objects, which each create (population size / n) new strategies. After the objects have been initialized a new Thread is created for each of them, after which all threads are executed. When all threads are finished, a new Generation object is created and filled with the results from each of the StrategyCreator objects.

4.4.2 Evaluation of Generations

Once a new generation has been constructed it needs to be evaluated so a new generation can be constructed based on it, but also so the current best strategy from the generation can be selected. When evaluating a generation all strategies are first evaluated individually, after which the mean, standard deviation, maximum and minimum performance of all strategies in the generation is calculated. The process of evaluating all individual strategies is the part that takes a lot of time and it is

also a part that can be parallelized efficiently because separate strategies can be evaluated independently of each other.

To evaluate a generation, the following method of the Dispatcher can be called:

```
public static void evaluateStrategies(List<Strategy> strategies,
      Date begin, Date end)
```

Once a new request has been received by the Dispatcher in this way, it will create N new StrategyEvaluator objects and corresponding Thread objects, with each (population size / N) strategies to evaluate. Each StrategyEvaluator object will then ensure the performance of each of its Strategies is calculated and stored. To account for rounding errors, the exact amount of strategies per StrategyEvaluator object can vary by one.

After all StrategyEvaluator threads are finished the Dispatcher is finished as well, as the method manipulates the input list of strategies.

4.4.3 Race Conditions

Because the originally single-threaded program had to be modified to allow for multi-threading, there were some newly introduced race conditions which had to be taken into consideration. Problems can occur when multiple threads are accessing data on the same location on the same time and both start manipulating it. The problems start occurring when shared data is not only being read but also written.

The first instance of these problems arises with the database access. All of the database access in the system runs through DAO Singletons, which means the getInstance methods of their classes need to be thread-safe. This could be simply achieved by making the getInstance methods synchronized, but that will introduce locking between threads for every call to getInstance, making the parallelization less effective than possible. To get around this issue, the instance attribute was marked as volatile and the getInstance methods were implemented as such:

```
public static SomeDAO getInstance() {
      if(instance == null) {
            synchronized(DataAccessObject.class) {
                  if(instance == null) {
                        instance = new SomeDAO();
                  }
            }
      }
      return instance;
}
```

By only making the creation of a new instance synchronized, the getInstance call will generally not lock up, making the process more efficient.

Another issue with thread safety occurs with the introduction of cache. It is important that cache access is thread-safe, but at the same time it is important that reading from a cache does not lock it up for other threads. To achieve this the same design pattern from the DAO Singletons was used. All read methods that hit a cache are non-blocking, but will use a synchronized section when the cache is empty and the results have to be fetched and stored.

To summarize, multithreading was implemented to boost the performance of the program to the maximum. This was done by parallelizing the creation and evaluation of strategy generations. However, multithreading does not come without any risks. Some design patterns were applied to prevent unacceptable risks to the stability of the program, according to the race conditions that were discussed. The result of this parallelization is that the workload of the program can be spread equally over all available cores, as one core will be reserved for the main thread and MySQL, where the rest of the cores will execute the generation and evaluation of the generations. This means that the performance of the program can be improved linearly to the hardware it will be run on.

4.5 MyBatis

A database is used extensively, since all strategies are dynamically created and are therefore stored in the database. Besides this, a great amount of historic data will facilitate the evaluation of these strategies. It would therefore be useful to use a library that could increase the ease of coding and decrease the chance of bugs by providing a framework for database manipulation.

In Java, there is a broad collection on database libraries, as mentioned in the Orientation Report. However, because of the huge amount of database transactions, even when historic values would be cached, the library should be light-weight. Therefore, it was decided to use MyBatis.
MyBatis is actually rather an ORM library than a framework, however it makes the interaction with the database much easier, since it will take care of JDBC connections and SQL queries each time the back-end would interact with the database. It beats other ORM libraries for Java in it's simplicity and configurability, however during the development of the application, this turned out to also be its disadvantage, since the integration and maintenance took more time than should be necessary. Nevertheless, we integrated MyBatis into our application. The integration mainly consists of implementing three different classes for each table in the database, of which a graphical representation can be found in appendix C.

4.5.1 Database Classes

POJOs

POJOs are Plain Old Java Objects, which basically contain several private attributes, together with a public getter and setter function for each of these attributes. When integrating MyBatis, for eacht table a POJO has to be implemented, containing an attribute for each field of the table. These POJO's were used throughout the program and therefore, some POJO's were extended with some extra attributes and functionalities rather than getters and setters, as these functionalities do not conflict with MyBatis.

DataMappers

DataMappers are interfaces that contain the mapping between the database and the application. This means that for each database transaction needed by the program, this interface contains a corresponding method. This method was linked with an SQL statement containing variables which would be determined by the parameters of the function, if necessary.
To return a usable result, each function that should return a POJO, was joined by a collection of tuples which map the column names of the table to the attributes of the POJO. The actual execution of these transactions is taken care of by MyBatis.

DAOs

DAOs are DataAccessObjects, which actually are the interface of the database towards the rest of the program. If some transaction has to be made in the database during the execution of the application,

the application should do this by calling the corresponding function of the right DAO. This DAO will then use a corresponding DataMapper to perform this transaction and will, if necessary, return the result of the transaction.

4.5.2 Integration

MyBatis supports two ways of integration. The first option is to use XML files. These files are used to construct the DataMapper configurations such as the SQL statements and the collections that map the table names to the POJO attributes. Besides this XML files, an additional XML file is needed for configuring the SqlSessionFactory, which will take care of the actual connection between the application and the database.

The second option is to use java code with annotations to configure the DataMapper and SqlSessionFactory. In our opinion this would create cleaner code and therefore it was decided to use MyBatis with annotations.

However, during the development process some maintenance issues surfaced considering MyBatis. On integrating MyBatis in the application, for each table a POJO has to be made, with a corresponding DataMapper and DAO. Later on, during the development process, for each transaction a function in the DAO has to be made, together with a function in the DataMapper, linked to an SQL statement and a collection of tuples. In this scenario, the maintainability of this framework was already not meeting our initial expectations.

While implementing the application, some fields were added to tables in the database. Here the maintainability issue really surfaced, since the corresponding POJO had to be updated together with the corresponding DataMapper. Despite the fact that updating the POJO is not a lot of effort, updating the DataMapper would mean that for every implemented function, the SQL statement and the mapping of field names to POJO attributes had to be updated.

However, all other expectations on this framework were met, the maintainability is not perfect and would be something to work on in the future.

4.6 Caching

As mentioned earlier it is very expensive to run Genetic Programming jobs because of the great amount of strategies that have to be created and evaluated. A possible way to reduce the processing cost was already mentioned in chapter 4.4, by multithreading. Another way to reduce the execution time is caching. Results that already have been calculated or retrieved from an external data source can be stored so any subsequent request for that particular data is much quicker.

There are a few places where the system can benefit from caching, first of all when accessing the database. The database is used quite frequently, especially the tables that store historic stock value information. Secondly all the created strategies in the GP process can be very similar. Evaluating the exact same tree twice for the exact same In-sample period will result in the same strategy performance, thus allowing for strategy performance caching as well. Lastly, components of strategies could be cached as well, because many strategy components are likely to be shared across different strategies, also because of the used GP process.

4.6.1 Database Caching

The database is frequently accessed when creating and evaluating strategies. The most common requests to the database are the retrieval of historic stock value information from HistoricValueDAO, retrieval of a Symbol object by its id and lookups in the Module table.

To deal with caching of recurring queries Ehcache[8] was chosen as it is known for its high performance and can easily be integrated with MyBatis. However, this was not sufficient for the retrieval of historic values. Ehcache works by caching any query it executes, but historic values are

requested for each separate symbol and date combination. For a test run with 100 symbols and 30 years of stock value data with 250 trading days per year, this comes down to 750,000 separate queries that need to be executed by MySQL before the cache is completely filled. To increase this performance it was decided to replace the standard integration of Ehcache within MyBatis with our own implementation that we could preload with one single MySQL query per symbol.

When preloading the historic value cache, an additional problem was the absence of data for some symbols on some trading days. The trading strategies are executed for every single known trading day and if a stock is not traded on a certain trading day, its last known value should be used. To get around this list synchronization issue as efficiently as possible, two custom iterators were implemented to iterate over both the trading days and the known historic values for a stock, being DateIterator and HistoricValueListIterator. The DateIterator simply iterates linearly through all dates, while the HistoricValueListIterator is used to update the list pointer to a new date, which in case of missing historic values means the pointer is not updated.

All the added database caching significantly improves the overall system performance, but Ehcache still has quite some overhead. To reduce this overhead a new type of custom caching was introduced with the HighPerformanceCache class. This class can be used in a very similar way to Ehcache, except it uses a simple Java `HashMap<String, Object>` to store its values and does not use any cache expiration policies. Without this HighPerformanceCache the system could perform roughly one `evaluate()` function call in 1.04 microseconds, after customizing the cache this time was reduced to 0.588 microseconds per call, indicating a 43% speed-up.

4.6.2 Strategy Caching

One of the reproductive operations within the GP process is duplication, meaning a new generation has a high chance of containing strategies that had already been evaluated for the same in-sample period for the previous generation. If the exact same strategy is evaluated for the exact same period of time with the same fitness function, the results will also be the same, meaning the results could be cached in these cases.

To implement this every generation writes the results of all its strategies to a cache in the job scope. To accomplish this strategies first generate a 'hash' which is unique for each strategy structure but is the same for two strategies with the same structure. This hash is generated by creating a textual representation of the strategy, which had already been implemented for debugging purposes. The reason the cache is stored in job scope and not in application scope is the fact that settings as the in-sample period and the fitness function can vary per job.

One downside of this method of caching is that writing all strategy performance objects to cache needs to be executed in a single thread, making the system somewhat less efficient on systems with many available cores. However, the performance increase seems to outweigh the cache writes, with a total speed-up of roughly 25% on an 8-core server.

4.6.3 Strategy Component Caching

Many of the components of strategies are also being reused in the GP process, as the reproductive operations often leave parts of strategies in tact. In theory caching these parts of strategies should provide a huge performance increase, but the downside is the enormous variety of strategy components. The first problem would be how to break up strategies, as the power set of the set of nodes in a strategy is quite large. As cache writes are relatively expensive and have to happen in a single thread it was decided not to cache separate components of strategies.

4.7 Historical Data

Key to building good strategies is a large amount of sample data. For modules that are mainly based on the chart of a stock, this means we need historical stock market values. In the database the following values are stored for a given symbol on a given date: open, close, high, low and volume.

There are many sources of data available to pull these values from. Early in the project, the application pulled its data from Yahoo! Finance, a high-quality free service with a public API. Unfortunately, the Yahoo! API only allows you to place a limited number of requests until the user is blocked from further access.

For the test runs, Bing Technology was able to provide an archive of stock market data from all the big US indices such as the NASDAQ, NYSE and S&P, going back to 1981. This archive contains text files for each stock with open, close, high, low and volume values.

There was a bit of a challenge getting this large amount of data in the MySQL database. The first approach was to use a Python script to parse the data files and generate an SQL script containing `INSERT` statements for each historic value. This proved to be suboptimal, as even the historic values of a collection of 100 symbols (the S&P 100 index) from 1981 to 2011 would take roughly 5 hours to insert into the database.

Instead of using pure `INSERTS`, a better approach was to use MySQL's `LOAD DATA INFILE` command to insert the data directly from the text files. This allows the same 100 symbol insertion as above to load in about 3 minutes.

4.8 Front-end

The front-end was built as a PHP web application. Its primary task is to facilitate user friendly CRUD (Create, Read, Update, Delete) operations for strategies, jobs and modules. In addition, it allows the user to analyze a strategy's internal tree structure, and can display interactive charts of a strategy's performance history.

As the application had little special needs besides the standard CRUD operations, we decided to use a web application framework. In particular, the front-end uses a custom MVC[1] framework called Roy[5], built by Maikel for a previous project. In addition, the front-end uses the Flourish[6] and PHP ActiveRecord[7] libraries for functionality such as session handling and object-relational mapping.

Each of the tables in the database has its own model class, and each of strategies, jobs and modules has a controller providing all relevant actions.

Being a data-centric application, the front-end could rely on PHP ActiveRecord for most functions. One function that is not supported is a cascading delete operation that will delete an object along with all its dependencies. This was implemented as a custom method named `delete_completely()`, which is overridden by any object that has dependencies. For a Strategy object for example, this method will delete that strategy along with any strategy values, performance objects, jobs and nodes.

[1] Model-view-controller, 2011-06-15, accessed 2011-06-20, http://en.wikipedia.org/wiki/Model–view–controller

5. Conclusions and Findings

The initial goal was to create a generic forecasting framework that could be used to predict anything. Because this task proved to be too non-specific, the goal was changed. The new task became creating a forecasting system that could create stock trading strategies. Also, the created trading strategies should perform better than other trading strategies and not only in the simulated period but also in the future. In more technical terms, the out-of-sample (OoS) performance should be good in comparison to other existing trading strategies.

5.1 In-sample Versus Out-of-sample Results

It turned out getting good OoS results is more difficult than one would expect, as any usage of OoS data in the construction of trading strategies renders the OoS data as in-sample (IS) data. Even though this might seem obvious, it results in persistent problems while trying to optimize OoS data. For instance, trying out three different input parameters for the forecasting system and selecting the best input parameters for future usage solely by comparing the OoS results of the three generated strategies would not be a good idea. Because this comparison looks at the OoS results, the OoS period would become an IS period and the results in the true OoS period would still be unknown.

This problem with IS and OoS results was resolved in two different ways. First of all, the used modules and maximum allowed nodes in created strategies were chosen in such a way that the chance of overfitting was small. Secondly all tests executed on real data were thoroughly documented and no tests were executed without proper argumentation to support the notion that the tests did not turn any OoS data into IS data. The result is a strong correlation between IS and OoS performance.

To test the correlation between the IS and OoS performance, the forecasting system was used to create 40 trading strategies of which the OoS performance / IS performance ratio was calculated. It would have been better to get even more test results but each test result was quite expensive to obtain, with individual runtimes of at least 3 hours. The small amount of sample data should therefore be taken into consideration when interpreting the results.

The 95% confidence interval of the OoS performance / IS performance ratio is shown in figure 5.1. The returns shown are annual factors of the initial investment, meaning a factor of 1.1 corresponds with profits of 10% per year. The confidence interval was calculated based on the null hypothesis that the results from the different created trading strategies were normally distributed. The basis for this hypothesis was a combination of a visual interpretation of a histogram of the results, shown in figure 5.2, and an interpretation of the generated strategies. All results were generated completely independent of each other, with only the initial settings for the GP process in common. Even though this might lead to the suspicion that the results would be very similar, the resulting trading strategies from individual test runs did not appear to have much in common.

To back up this hypothesis of a normally distributed dataset, a Shapiro Wilk[2] normality test was performed on the resulting ratios to try and reject the null hypothesis. It could not reject this hypothesis with a used significance level (alpha) of 5%. It is therefore likely that the results are normally distributed.

[2] Shapiro-Wilk test, 2011-02-15, accessed 2011-06-21, http://en.wikipedia.org/wiki/Shapiro%E2%80%93Wilk_test

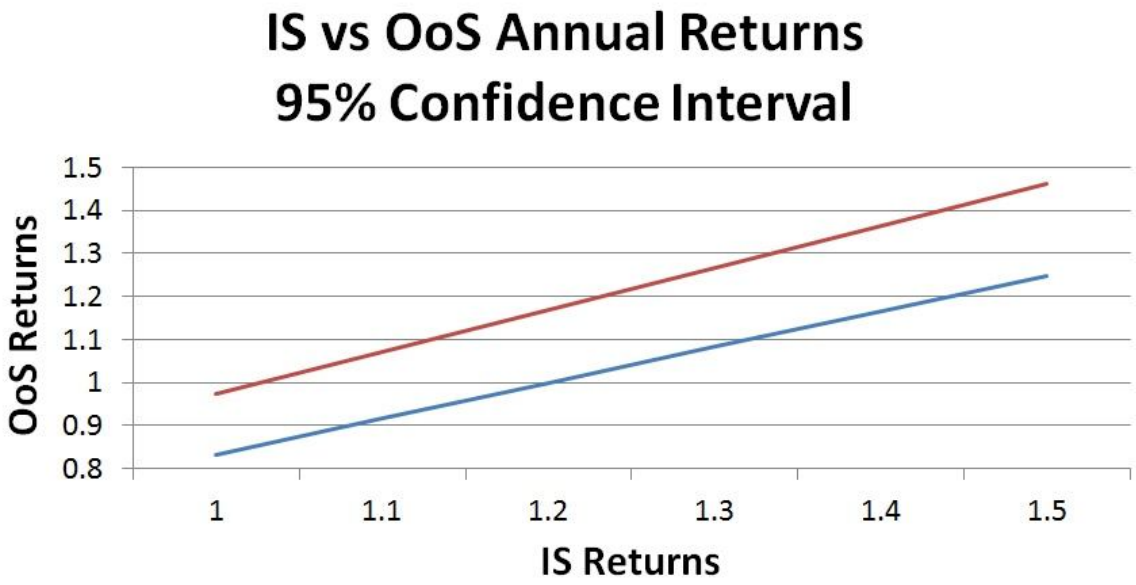


Figure 5.1.

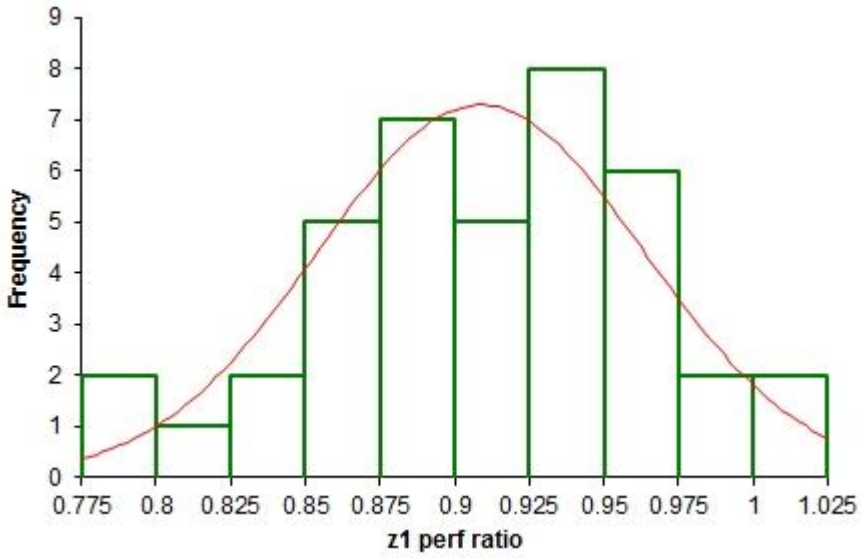


Figure 5.2.

5.2 Execution Time

Another interesting conclusion is the relatively long time it takes to execute tests. Each test run can easily take a few hours on the regular Bing test server, making the execution of a large set of test runs a very time consuming process. An average set of tests needs a large amount of runs to get more stable results. Also, the tests seem to show that tests with larger population sizes or more generations are usually more likely to yield better in-sample results. It was initially expected that test runs could not run instantaneous but the final length of test runs was much longer than expected. The main reason for this underestimation of execution time was an underestimation of the required population size and the amount of runs.

To get a better understanding of the time it takes for strategies to be created, the size of a GP job first had to be normalized. A way to measure the total complexity of a job is to multiply the average amount of nodes of all constructed strategies by the amount of generations, the population size and the amount of evaluated trading days. If the system scales linearly to those inputs, the complexity number multiplied by a certain constant C should be fairly close to the time it takes to execute a job, with the same C for all jobs. The time it took to execute the first 49 test runs, all executed on the same server, is shown in figure 5.3.

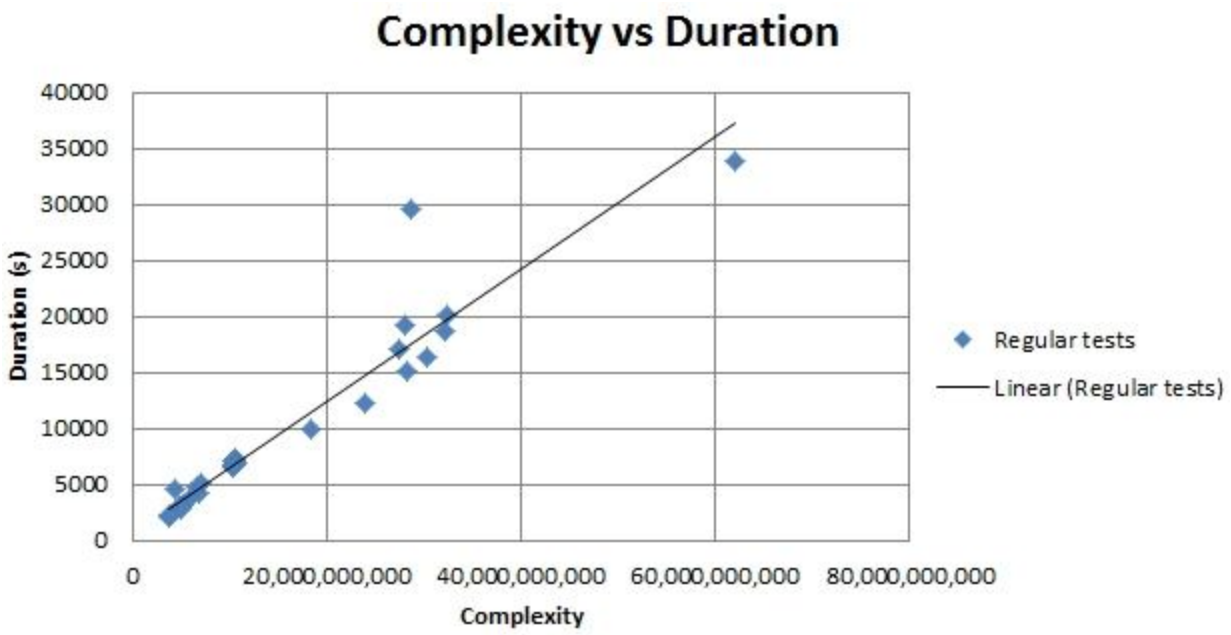


Figure 5.3.

Also shown in the chart is the complexity multiplied by a certain constant C, indicating an average processing speed of roughly 1.7 million complexity units per second. All actual execution times seem to be fairly close to the expected time based on the complexity units, meaning it is likely that the execution time in fact scales linearly to the amount of complexity units of a job. There is one result that stands out from the rest, which could be explained by the fact that the server used in the tests was running other processes as well, including periodically executed cron jobs.

6. Recommendations

From the outset of this project it was clear that we had an idea with a lot of potential and a very large scope. Not only did we want to take on the ambitious – some would say foolish – task of predicting the stock market, but we also wanted to apply these ideas to other forecasting domains.

It is therefore no surprise that we have possible improvements in mind for future development of the product. This chapter gives on overview of recommendations to future teams (possibly including ourselves), to improve and expand upon the forecasting application.

6.1 Parameter Tweaking

As a result of our Genetic Programming approach, the application contains a lot of parameters. Fitness function, population size, GP operation rates such as the duplication rate, just to name a few. Although we've extensively tested the program to optimize these parameters, we feel there is room for improvement.

An example of possible further improvement is tweaking the used generation size for different fitness functions. It is possible to change the fitness function to get better results, but it is not certain what the minimum required generation size should be to get stable results using that new fitness function. As a result of this, every change in fitness function should also result in a thorough retesting of the minimum required generation size, the downside of which is the relatively long time it takes to run a new batch of tests.

Tweaking the required generation size is just an example of the dependencies between parameters that needs to be tested. In theory every single combination of parameters needs to be tested, but that is not feasible given the amount of time it takes to run tests. However, it is recommended to create a test plan for further parameter tweaking, researching known and expected dependencies and suggesting a minimal set of required further tests.

6.2 Modules

In order to be able to construct powerful and flexible strategies, we have implemented a wide variety of technical indicators as modules. However – as a quick look at a reference such as StockCharts' list of Technical Indicators and Overlays[2] will tell you – there is much more ground to cover. A list of additional technical indicators that are suggested to be implemented is shown in appendix B1.

6.3 Industry Aggregation Data

At the moment, all strategies are based purely on data from individual stock symbols. The algorithm may perform better when we take into account aggregative data from entire financial sectors or industries. For example, we would want to answer questions such as: "Which financial industry should I invest in, Technologies or Consumer Goods?"

Sources of financial data such as Yahoo! Finance can provide this kind of data[3]. For example, Yahoo! Finance can tell you the total market cap and P/E ratio on a per-sector or per-industry basis.

6.4 Qualitative Data

As mentioned in the Orientation Report, there are multiple ways to approach investment strategies. One is Technical Analysis, or the chartist approach, which looks only at historical stock price data. Another is Fundamental Analysis, which bases its decisions on information about a company, such as its earnings reports.

So far, most modules we have implemented are chartist in nature. We would like to incorporate modules using fundamentalist data as well. There are data sources available for fundamentalist data, we

would need to look into these sources to see exactly what sort of data they provide and which source is the best.

We may also want to include more unconventional sources of data such as Twitter. The Orientation Report contains more information about the potential of Twitter as a source of investment data.

6.5 Short Selling

So far all we have allowed trading strategies to do is taking long positions in stocks. This basically means the strategy can select any of the provided stocks and buy as much of it as it has available cash or sell as many of the stock as the strategy owns. In addition to these long positions, the system could also be adjusted to allow for covered short selling as well. When allowing strategies to take advantage of this trading method, they can benefit from drops in the price of stocks, rather than rises.

One possible downside of allowing short selling is that some implementations enable the loss of more assets than available to the trading strategy, resulting in effective bankruptcy of a trading strategy. This situation could be resolved by limiting the possibilities for short selling available to the system to relatively safe ones, such as purchasing, but not selling, put or call options.

There are different types of available options as well, of which American-style options seem to be the most practical because of their flexibility, making them easy to connect to the existing system. However, more research is needed to find out if any other option styles might be more fit for usage in the forecasting system.

6.6 More Advanced Option Trading

Besides the introduction of options to enable short selling, the system could also benefit from the introduction of other combinations of options. Some investors might prefer to reduce their exposure by limiting the downside risk on investments, which could be ensured by wrapping an entire strategy by a stabilizing option strategy. A possible stabilizing strategy that could be implemented is a collar, which buys put options and sells call options for the stocks owned.

The advantage of this strategy would be that possible losses per transaction are limited, providing more certainty for the investor. A disadvantage of this collar strategy is the limit it imposes on the possible gains per transaction, although that can also be seen as a possible advantage for the resulting Sharpe ratio.

Other option strategies should be investigated as well, such as fence, straddle, strangle, iron butterfly and iron condor.

6.7 More Accurate Transaction Costs

An important factor in creating strategies is how transaction costs are handled. A broker will charge a certain amount for every buy/sell transaction, limiting the amount of transactions an investor can perform without making too much losses.

At the moment, transaction costs are handled rather conservatively. In reality, broker policies for transaction costs can be complex. We would like to take the utmost advantage of transaction policies to improve our strategies.

6.8 Cloud Computing

Creating and evaluating strategies is very computationally intensive task. Although optimizing the software has been a priority, pure software optimization can only get you so far. In order to improve our results, we would need a lot of hardware.

Instead of buying and maintaining this hardware ourselves, we would like to utilize one of the many cloud computing services available. In particular, we'd like to deploy the application to Amazon's Web Services (AWS) platform[4].

Given our back-end/front-end architecture, it would make sense to deploy only the back-end to AWS, and have the front-end talk to each of these instances. This would require some rethinking in how we communicate between back-end(s) and front-end, either by re-engineering the front-end to talk to several (remote) databases, or by abstracting – from the front-end's point of view – the remote databases into one big master database.

References

1. StockCharts.com – Chartschool, *Money Flow Index (MFI)*, 2011-01-25, accessed 2011-06-16, <http://stockcharts.com/help/doku.php?id=chart_school:technical_indicators:money_flow_index_mf>
2. StockCharts.com – Chartschool, *Technical Indicators and Overlays*, 2011-06-09, accessed 2011-06-17, <http://stockcharts.com/school/doku.php?id=chart_school:technical_indicators>
3. Yahoo! Finance, *Industry Browser – Sector List*, 2011-06-17, accessed 2011-06-17, <http://biz.yahoo.com/p/>
4. Amazon Webservices, accessed 2011-06-17, <http://aws.amazon.com/>
5. Krause, M.R., *Roy project page*, 2011-05-03, accessed 2011-06-20, <https://github.com/mkrause/roy>
6. Flourish, accessed 2011-06-20, <http://flourishlib.com>
7. PHP ActiveRecord, accessed 2011-06-20, <http://www.phpactiverecord.org>
8. Ehcache, Terracotta, 2011-05-23, accessed 2011-06-20, <http://ehcache.org>

Appendix C - MyBatis

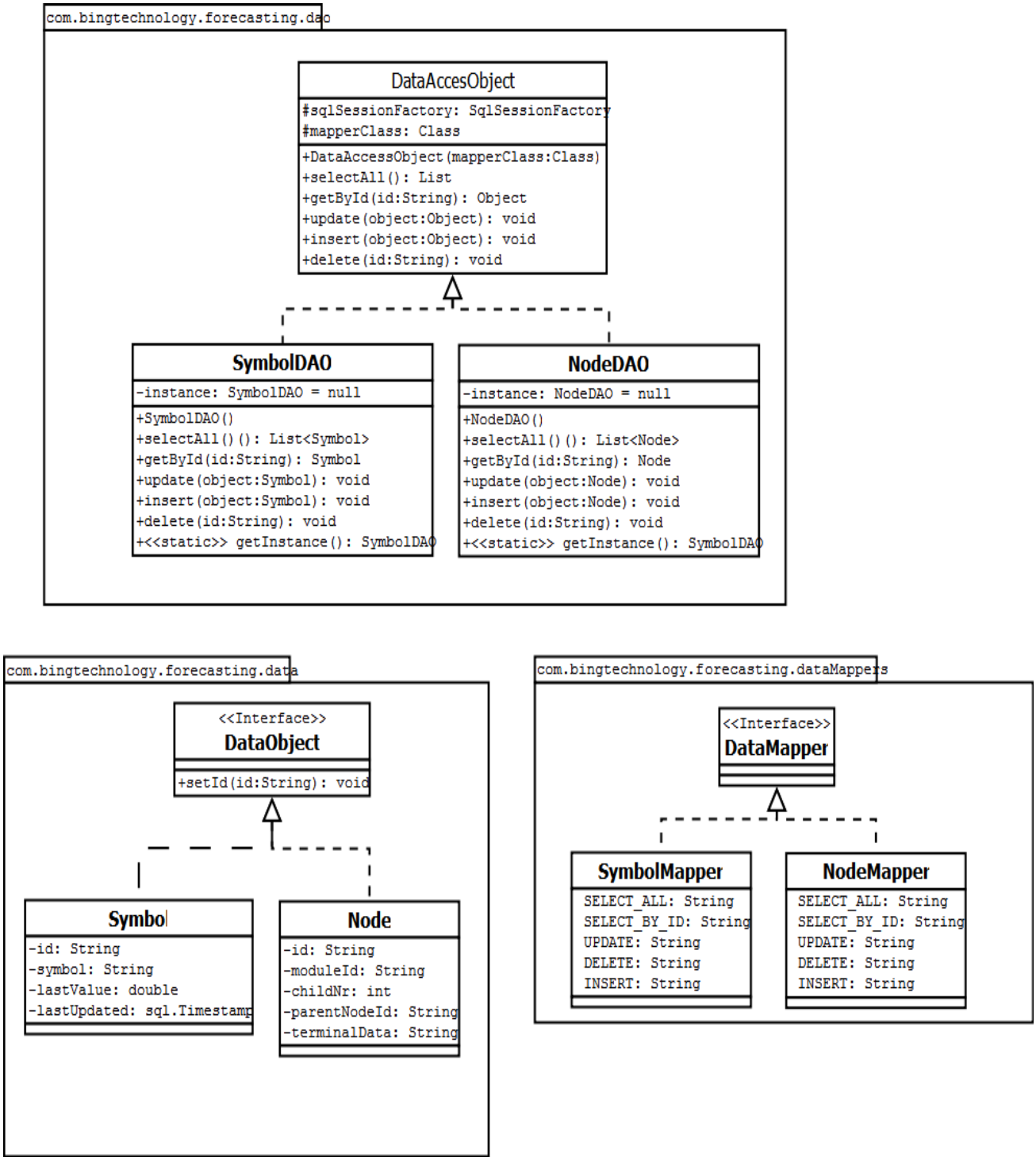


Figure C.1. The design of the MyBatis library. For each table a DAO, DataMapper and Data object exists, however only Symbol and Node are viewed here as an example.

Appendix F - Packages

Data

The Data package contains all the classes for data objects. These are simple POJOs (Plain Old Java Objects), and consist only of some attributes and their getter and setter methods. If necessary, additional functionalities can be added to these objects to increase the ease of coding.

DAO

This package contains all the DAOs (Data Access Objects), which can be seen as model classes (in the MVC sense). These objects use their Data Mappers, in combination with the MyBatis library to manipulate the database.

Strategy

This package contains all the controllers that construct and evaluate trading strategies. This is therefore divided into subpackages, listed below.

Strategy.Nodetypes

This package contains the abstract data classes which represent the different node types a tree can consist of. Note that these abstract classes contain several methods and attributes, however they do not put constraints on the amount and types of their children.

Strategy.Modules

This package actually contains the innovative part of the application. In this package, all implemented modules will be stored which are basically the nodes of which the tree of a strategy will be constructed.

Strategy.Evaluate

This package contains all controllers which are necessary to evaluate the constructed strategies.

Genetic

This package contains all the controllers needed to genetically construct strategies. In fact, this is the real Genetic Programming part of the system.

Parallel

This is a smaller package, however not less important, since it provides the functionalities that are needed for multithreading some parts of the program, so we can really use the full capacity of all available cores on the server. In the future it can be adapted to allow for job distribution across multiple servers.

Poller

This package will contain the part of the program that is continuously running. This means it will retrieve jobs from the database and execute them with the use of the other packages, as well as evaluating all strategies that weren't evaluated in the last 24 hours.

Scraper

This package is a less complex however not less important part of the program, as it scrapes historic values from the web. Initially this will be done from Yahoo Finance, however this package can also

include other scrapers such as a Twitter scraper. These historic values will be stored in the database after which they can be used for the evaluation of constructed strategies.

Test

Last, but certainly not least, the test package. Not all methods will be tested since some are quite trivial to test. For all code that will be tested however, the test suites will be put in this package.

Appendix G – Database Design

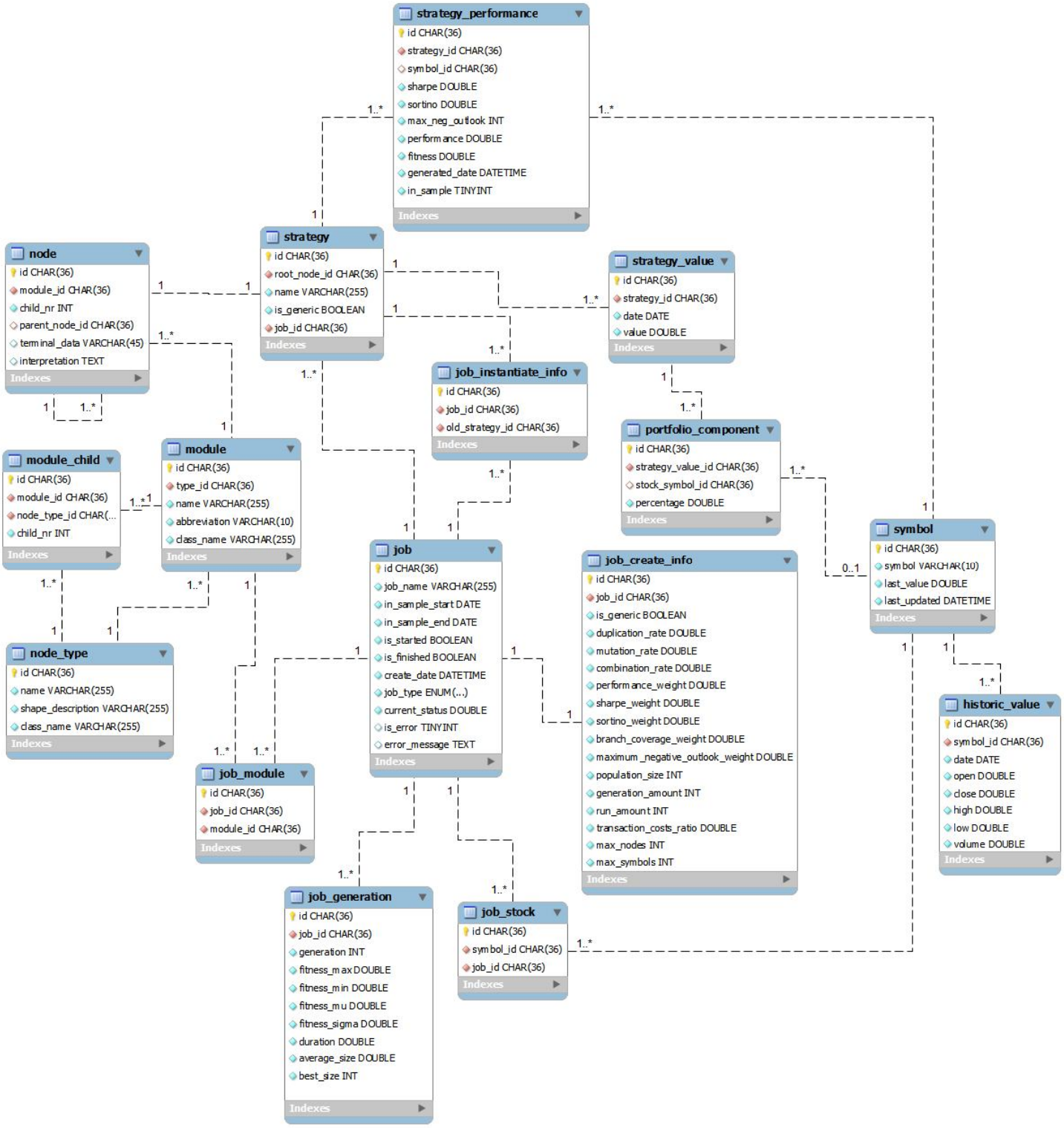


Figure B.1. The database design

Appendix H – Genetic Programming Sequence Diagram

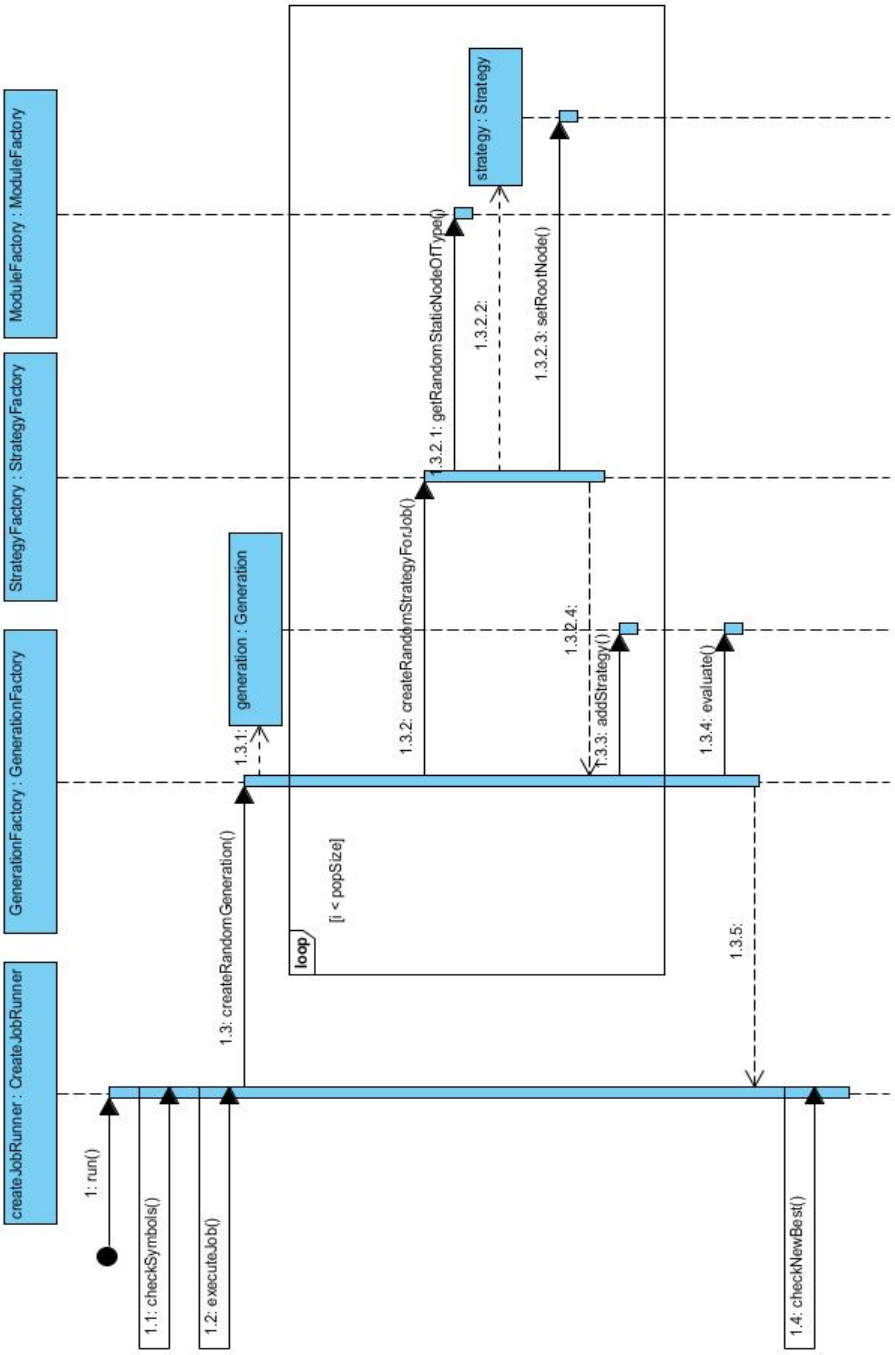


Figure D.1. The first part of genetically creating new strategies.

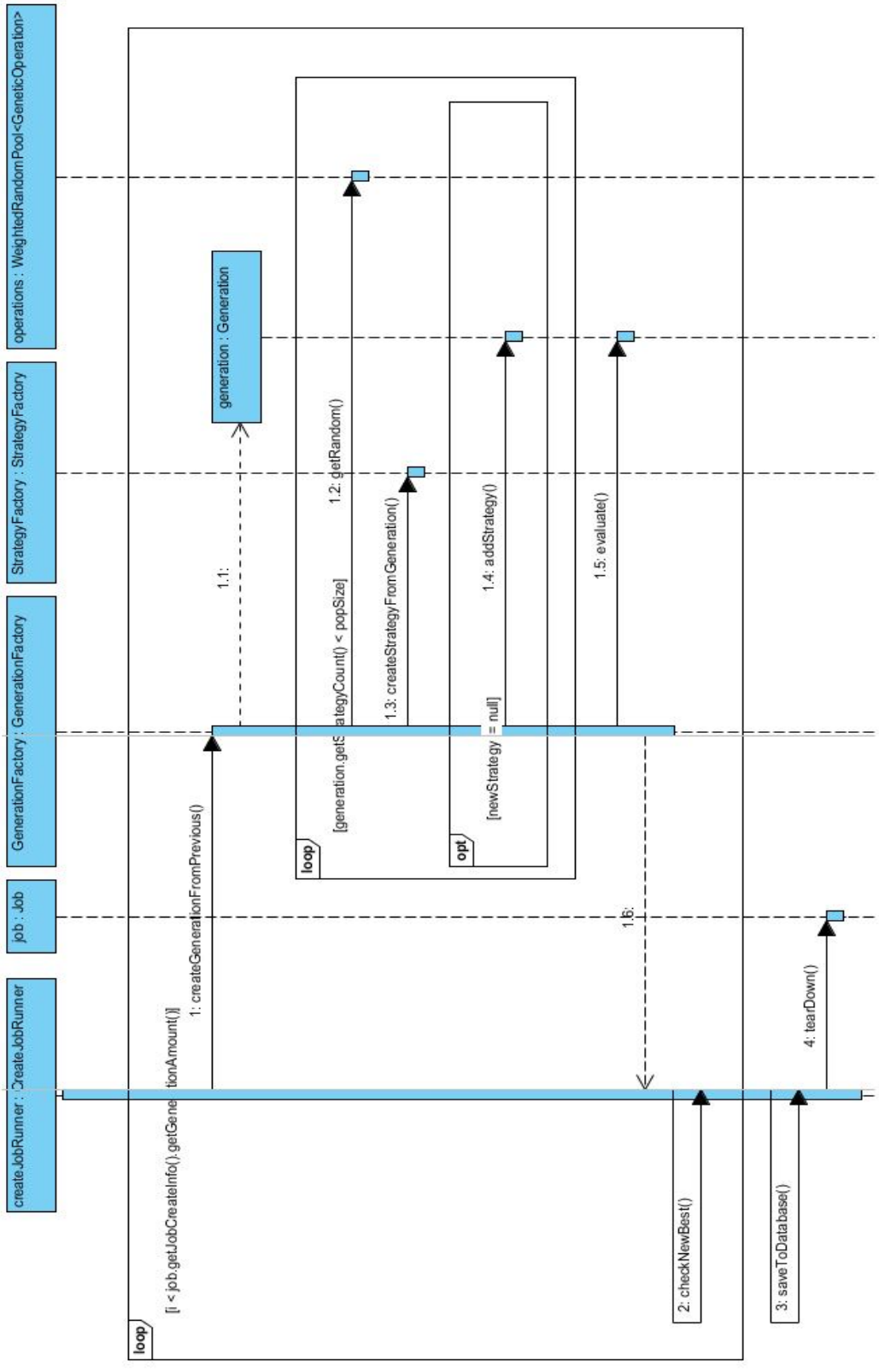


Figure D.2. The second part of genetically generating a strategy.

Appendix I - Orientation Report

Preface

In the fourth quarter of the 2010–2011 academic year, we – a team of three Computer Science students at Delft University of Technology – will be working on our BSc. Project at Bing Technology in the Philadelphia area, USA. The project entails the design and implementation of forecasting software. This Orientation Report contains the results of our initial research into the field of forecasting and some of the available technical solutions – such as software libraries – we might use to build such forecasting software.

1. Introduction

Forecasting is a problem with a long history and many applications. Given a record of past events, can we say anything about future outcomes? In other words, can we predict seemingly random patterns like the weather, the stock market or the pattern of sun spots given historical data? Such predictions are not always possible. For example, some have hypothesized that stock market prices evolve according to a completely random process called a random walk[3]. If this is the case, we cannot say anything useful about the future of such a statistic. If, however, we can discover certain trends and patterns, then we can try to find strategies to accurately predict changes.

We are interested in the problem of forecasting in general, and believe that there is enough overlap in the various techniques for forecasting to merit the making of a general software framework that can be applied to specific problems using a modular approach. However, we will focus on two specific fields of interest to guide us, and the implementations of one or both these will form an important end product for our project. The first field of interest is the stock market. There has been a lot of work in this area to analyze data and find effective strategies. It is also an area where a small algorithmic improvement over current methods can have a big impact. The second field was inspired by a competition set up by the Heritage Provider Network, the goal of which is to predict which patients in America are at high risk to be hospitalized in the next year. An algorithmic solution to this problem has the potential to save billions of dollars in unnecessary hospital admissions.

2. Heritage Health Prize

On April 4th, 2011, the Heritage Provider Network (HPN) launched a competition to find an effective predictive algorithm that can identify patients who will be admitted to a hospital within the next year, using historical claims data[4]. The HPN has offered a prize of USD $3 million for the winner of the competition. The applicant receives a dataset of three years of medical insurance claims, and will be asked to predict the values of the fourth year. The applicant's entry will be evaluated using the following score:

$$\varepsilon = \sqrt{\frac{1}{n}\sum_{i}^{n}[\log(p_i + 1) - \log(a_i + 1)]^2}$$

Figure 2.1. Heritage Health Prize evaluation function.

[3] *Random Walk Hypothesis*, 2011-05-03, accessed 2011-06-21, <http://en.wikipedia.org/wiki/Random_walk_hypothesis>

[4] *Heritage Health Prize*, accessed 2011-06-21, <http://www.heritagehealthprize.com>

Where p_i is the predicted number of days in hospital for member *i*, and a_i is the actual number of days, unknown to the applicant. The Heritage Health Prize is part of the Kaggle platform of data prediction competitions, which will be covered in more detail in chapters 3 and 4.

3. Forecasting Methods

Many techniques have been developed to perform forecasting, many statistical in nature. This chapter reviews some of these techniques.

3.1 Regression Analysis

Regression analysis[5] is a statistical method used to analyze the relationship between a set of variables, some dependent on the others. Using a regression model, one can extrapolate the data to values outside the range of the dataset, making such models very useful to make predictions. Regression models are often combined with machine learning algorithms, which have proven to be efficient at creating a regression model from large amounts of data.

3.2 Stock Market Analysis

Stock market forecasting involves trying to determine the value of company stock at some point in the future. Or, in a less absolute but still useful form, which stocks to buy, sell or hold, and when. This determining "which stocks" is often called *selective skill*, whereas the "when" part is called *timing skill*. Methods to perform forecasting fall into three broad categories:[1]

- Fundamental Analysis
- Technical Analysis
- Technological Methods

These methods can be automated to various degrees, but an investor will generally stay involved in the decision making process.

3.2.1 Fundamental Analysis

Fundamental analysis[6] involves analyzing all relevant information about a business, such as financial statements, competitive advantages and competitors. An important tool for fundamental analysis is a company's financial ratios, such as the Price/Earnings (P/E) ratio.

3.2.2 Technical Analysis

Technical analysis[7], or "charting", is based on the trends of a company's stock price. Technical analysts use *time series analysis* (where a time series is a series of data points in time) to determine future prices. "Trends are your friend," is their motto. Many numerical patterns (such as "head and shoulders" and

[5] *Regression Analysis*, 2011-06-19, accessed 2011-06-21, <http://en.wikipedia.org/wiki/Regression_analysis>
[6] Investopedia , *Fundamental Analysis: Introduction*, accessed 2011-06-21, <http://www.investopedia.com/university/fundamentalanalysis>
[7] Investopedia , *Technical Analysis: Introduction*, accessed 2011-06-21, <http://www.investopedia.com/university/technical>

"cup and saucer") and statistical methods (such as the exponential moving average) are used to predict future trends.

3.2.3 Technological Methods

Making use of the power of modern computers and advents in the science of computational models, technological methods are used to determine future stock prices using techniques such as Artificial Neural Networks (ANN) and Genetic Algorithms/Programming. Moreover, some are using Data Mining (of social media like Twitter for example[1]) to find out what the current sentiments are regarding a certain company.

4. Market Analysis

A number of software solutions already exist to perform forecasting. This chapter presents an overview of the most important such solutions available today.

4.1 Forecasting Platforms

4.1.1 Kaggle

Kaggle[8] is a website that serves as a platform for data prediction. It hosts competitions, like the Heritage Health Prize, with the goal of solving data prediction problems using outsourcing to a large crowd. Kaggle can be useful to us as a source of information, as it gathers a large community of experienced data analysts, and it might even serve as a verification of our software if we manage to solve one of the given problems.

4.1.2 Google Prediction API

Google offers an API to run machine learning algorithms on their powerful servers.[9] A user can upload training data via this API, and have Google perform predictions on new instances of the data. The API can be used to identify, for example, spam, the language of a piece of text, or product recommendations. This API may prove useful to us as a way to run very complex calculations as part of our algorithms, if necessary.

4.2 Stock Market Software

A number of software solutions already exist to perform stock market analysis. Unfortunately, the trading software scene is filled with "get rich quick" schemes. This section attempts to separate the wheat from the chaff and give an overview of the most prominent products available.

4.2.1 Online Stock Screeners

There are various online services providing up to date or even real-time stock screening. Prominent such services include Google Finance and Yahoo Finance, both of which are free to use. These sort of financial services are a fast and easy way to gather a large amount of data, but are generally limited in their ability to analyze this data.

4.2.2 MetaStock

Type: charting tool

Price: $499+

MetaStock[10] is a technical analysis tool by Thomson Reuters. It allows you to view current and historical stock data, and perform statistical methods on this data. The software provides many "expert strategies" advising you when to buy/sell/hold. It also allows you to back-test a strategy on historical data to test its performance.

---

[8] *Kaggle*, accessed 2011-06-21, <http://www.kaggle.com>
[9] *Google Prediction API*, accessed 2011-06-21, <http://code.google.com/apis/predict>
[10] *MetaStock*, accessed 2011-06-21, <http://www.equis.com>

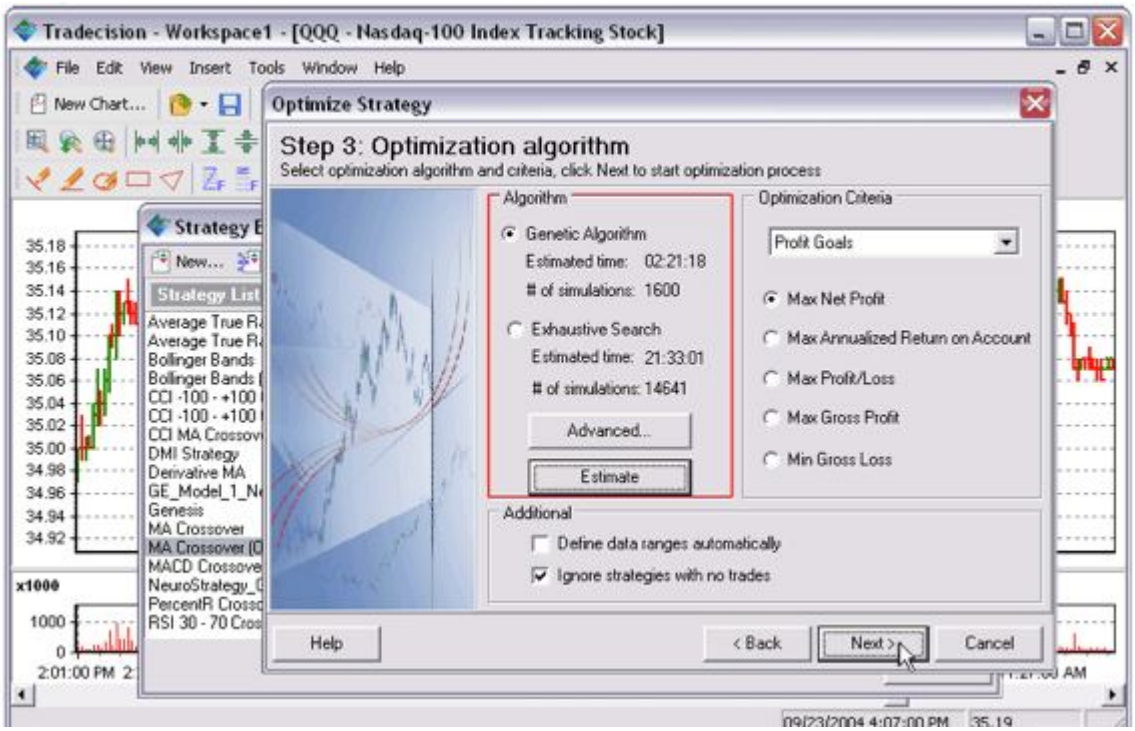
4.2.3 Tradecision

Type: charting tool including technological methods

Price: $1195 – $1995

Tradecision[11] is another technical analysis tool, with the distinguishing feature that it allows you to create Neural Network models to use in trading strategies, as well as using a genetic algorithm for fast optimization. This is an example of using technological methods to optimize a strategy.

[11] *Tradecision*, accessed 2011-06-21, <http://www.tradecision.com>



4.2.4 Tweet Trader

Type: data mining (technological method)

Price: free web service

TweetTrader[12] is a free web service that produces stock market predictions according to tweets concerning the stock market. As mentioned by J. Bollen et al.[2], predictions can be made based on Twitter moods, so a PhD student of the University of Munich, called Timm Sprenger, analyzed 250,000 tweets during six months and found that investors following stock market tweets could have achieved an average return rate of 15%[3]. Consequently, he developed a web service that gathers tweets related to the stock market and asks the community to evaluate whether they think this tweet speaks positively or negatively about one or more stock symbols. These inputs are then used to improve the text analysis algorithm the website uses to make predictions on individual stock symbols.

They are also currently improving their algorithms to detect tweets that consider stock branches, rather than individual stock symbols [12]. All in all this results in a service which can make very accurate predictions (up to 98%) on individual stock symbols [13], which means that it might be interesting to use this service as part of our financial forecasting software.

[12] *TweetTrader*, accessed 2011-06-21, <http://tweettrader.net>

4.3 Forecasting Research Using Innovative Algorithms

During the search for any information on financial forecasting, some research on forecasting algorithms was found related to our approach, using innovative algorithms like genetic programming. In this section, these articles will be summarized and discussed briefly.

Grant V. Farnsworth et. al.[4] describe how genetic programming can be used to combine different buy-sell-hold signals based on different modules like the moving average convergence/divergence and various exponentially weighted moving averages. These strategies are then genetically combined to create an algorithm that uses the best of both to ultimately create one buy-sell-hold signal that is more reliable than all other signals currently known. In this paper, the authors conclude that genetic programming can be used to identify predictable patterns in financial asset prices and outperforms most buy-sell-hold signals on average.

In general, this approach is quite similar to our approach. However, a fundamental difference is that the outcome of our product is more than just a buy-hold-sell signal for one stock symbol. Our output will be a proposed portfolio composition with exact amounts, rather than just a signal which tells you whether you should buy or sell, without any recommended amount.

Another difference is that our algorithm will be far more complex. In the paper the authors use relatively simple operators and literals, as they use buy-sell-hold signals, which are all the same kind of input, and compare these to each other. In the approach proposed to Bing Technology, the generated strategies will consist of different modules which have different output types, connected by different operators, which require various input types. This will therefore be seen as evidence that our approach is first of all feasible and secondly will outperform conventional methods on average. Taking in mind that our algorithm has a much more complex structure, which gives the opportunity to actually act on any time series that are provided to the algorithm, our algorithm is bound to have a result which should be at least as good as the result described in the paper.

5. Metrics

In order to optimize a trading strategy, we need to have some measure of the quality of an investment. This chapter presents an overview of some of the available metrics.

5.1 Sharpe Ratio

The Sharpe Ratio is a measure of the excess return of an investment, with a correction for the risk of an investment. It is defined as:

$$SR = \frac{R - R_f}{\sigma} = \frac{E[R - R_f]}{\sqrt{Var[R - R_f]}}$$

Here, R and Rf are stochastic variables representing the asset return and the return of a benchmark (e.g. a risk-free investment like a savings account) respectively. R - Rf then gives us the excess return over the risk-free rate, and σ is the standard deviation of this excess return. When Rf is replaced by Rb – the performance of a certain benchmark such as the market as a whole – the same formulas can be used to calculate the Information Ratio.

The SR is a dimensionless value indicating the performance of the investment. However, the measurement interval does affect the Sharpe ratio. Due to the central limit theorem, under the assumption that returns per time unit are independent and identically distributed measuring the returns per month will result in a higher standard deviation than measuring the returns per deviation[11]. Usually, Sharpe ratios are measured each month and the results annualized[5]. This is done as follows:

$$SR(q) = \sqrt{q} \cdot SR$$

When applying this formula to monthly measurements, the annualized Sharpe SR(12) becomes the monthly Sharpe ratio SR multiplied by the square root of 12. It should be noted this aggregation into an annualized Sharpe is not always correct, but an excellent approximation for most applications[5].

In some cases the returns per time unit are not independently distributed, in which case an alternative method should be used to create annualized Sharpe ratios from monthly measurements. This method is described as:

$$SR(q) = SR \cdot \frac{q}{\sqrt{q + \frac{2}{Var(R_t)} \cdot \sum_{k=1}^{q-1} (q-k) \cdot Cov(R_t, R_{t-k})}}$$

Because this second method of aggregation will take significantly more time to implement and run than the first method, increase the complexity and only provide more accurate results in some cases[5], we will use the simplified method of Sharpe ratio aggregation.

When comparing two Sharpe ratios to each other, the measurement interval is not important, as long as both ratios were measured in the same way. When looking at Sharpe ratios on their own however, it is important to use one standard measurement interval. This standard measurement interval is usually one year, which is also referred to as annualized Sharpe ratio. Given two investments, if we use the same benchmark, then the investment with a higher Sharpe Ratio has a better risk adjusted performance than the other.

Note that, since S is dimensionless, we cannot say anything useful about the absolute value of the Sharpe Ratio, we can only use it to compare different investments among one another. A related instrument, the Modigliani Risk-Adjusted Performance (RAP), addresses this weakness. This instrument is calculated as follows:

$$M^2 \equiv S \cdot \sigma_B + \overline{R_f}$$

Where σ_B is the standard deviation of a benchmark portfolio and Rf is the Risk-free rate. When comparing individual strategies from one GP job, both the σ_B and the Rf remain the same, making comparison of RAPs redundant when the Sharpe ratios of the same strategies are already being compared.

As a general guideline, we can say that an annualized Sharpe ratio of 1 to 2 is good, 2 to 3 is very good and 3+ is excellent[6], a Sharpe ratio of 5 or higher should not be possible[7].

5.2 Sortino Ratio

While the Sharpe Ratio is a very useful indication of the risk-free performance of an investment, it has a major downside. Because the ratio is divided by the standard deviation of the excess return over the benchmark, it penalizes both upside and downside volatility. It could be argued that upside volatility by itself does not attribute to the total risk involved in an investment. Also, under certain circumstances the overall Sharpe ratio could be improved by only 'discarding' some of the profits that caused the upside volatility. An example:

	Excess return over benchmark	Excess return over benchmark, with some discarded profit
	1	1
	2	2
	3	3
	4	3
Mean	2.5	2.25
Standard deviation	1.29	0.96
Sharpe	1.94	2.35

An alternative ratio that addresses this issue is the Sortino ratio, which is calculated in roughly the same way as the Sharpe ratio, except it divides by the downside risk only. The formula for the Sortino ratio is:

$$Sortino = \frac{R-T}{DR}$$

Where R is the annualized return of the investment, T is the target and DR is the downside risk. If the value for T is determined by Rf, the ratio can be used to calculate the risk-free performance of investments, much like the Sharpe ratio.

The downside risk is calculated as:

$$DR = \sqrt{\int_{-\infty}^{T} (T-x)^2 f(x) dx}$$

where *f(x)* is the probability distribution function of R.

5.3 Calmar ratio / Maximum Drawdown

Another way to measure the performance of an investment strategy is to see how big the biggest sustained losses have been. This is important because the change needed to recover from a loss relative to the loss increases with the loss ($\Delta' = \frac{1}{1-\Delta} - 1$). A measurement for the biggest sustained losses is called the maximum drawdown (MDD), which is specified as the maximum relative drop from the prior maximum. This can be written as the following formula:

$$MDD(T) = Max_{q \in (0,T)}\left[Max_{t \in (0,q)}\left[X(t) - X(q) \right] \right]$$

Where *X(t)* is the total value of the portfolio at time t. Using this MDD, a new ratio, called the Calmar ratio, can be constructed that accounts for both the profits and the greatest sustained losses:

$$Calmar(T) = \frac{X(T)}{X(0) \cdot MDD(T)}$$

Note: A very similar ratio is the Sterling ratio, in which the MDD(T) is replaced by (MDD(T) - 10%).

An interesting aspect of the Calmar ratio is its statistical relation to the Sharpe ratio. For a similar measurement period, the Calmar ratio can be derived from the Sharpe ratio and vice versa[8]. Because the measurement period is the same for all different evaluated strategies within one GP job, usage of the Calmar ratio in the fitness function will not yield any better results than usage of the Sharpe ratio. The same goes for the Sterling ratio[8].

5.4 Bias Ratio

The Bias Ratio (BR) is a ratio that indicates the ratio between the amount of months with a positive performance versus the amount of months with a negative performance. This is calculated by:

$$BR = \frac{PM}{1 + (M - PM)}$$

The amount of months which yielded a profit is described as PM and the amount of months in total is described as M.

The BR can be used in addition to the Sharpe Ratio to detect return smoothing. A Sharpe Ratio can in some cases be artificially high due to excessive smoothing, which can be detected by the BR[9]. Smoothing is a form of 'Financial Engineering' in which excessive returns (returns above the average) are temporarily hidden and resurfaced at a time at which the returns are below average[8]. Because smoothing by itself does not add value, but does increase the Sharpe ratio and some investors base their actions on the Sharpe ratios, an instrument to detect smoothing is required. The BR is one quantitative instrument to detect this. It is also shown not to have a statistical relationship with the Sharpe Ratio, making it a good addition to the set of measurement instruments.

Because all investment strategies constructed by one GP job use the same valuation method, there is no risk of accidental smoothing which therefore does not need to be detected by the fitness function. Therefore the Bias Ratio will not be a part of the fitness function.

5.5 Maximum Negative Outlook / Investor Emotion Quantification

Another important aspect of any investment strategy is the maximum negative outlook it yields. When applying investment strategies in reality, it is important that any failure can be detected quickly so the strategy can be abandoned before too much damage is done. Also, investment strategies that result in long negative outlooks easily lose the trust of potential investors, which should also be avoided.

There are several different methods to calculate the maximum negative outlook and no single method is perfect as its effectiveness is mainly based on human emotion. The different methods to measure the maximum negative outlook are:

- **Longest depression**
  The longest period between any two points in time during which the value at the first point is not exceeded by any other point.
- **Longest underperformance of safe investment**
  The longest period of time the investment strategy performed less than an investment strategy taking a long position in a safe investment from the start.
- **Most consecutive losses**
  The maximum amount of consecutive time-units with negative performance. These time units should probably not be as small as one day, as a single positive day does not emotionally compensate a series of surrounding negative days. The time-units should also not be as large as one year, as the difference between 0 and 1 year with negative performance is too large for the range of emotional stability of investors. Useful options could be weeks or months.

The first option, measuring the longest depression, has one critical flaw. If at any point the investment reaches a high and a recessing market causing a steep decline, a full recovery could be very time consuming. The entire period it takes the investment to return to its old high would be considered as one long depression, while this is likely not to be a good representation of the investor emotion. Therefore this method does not seem to be a valid way to measure the maximum negative outlook.

The second option seems like a fairly reliable way to measure the maximum negative outlook as investors are likely to compare their current returns to a safe investment that started out at the same point. However, investment strategies that start out with very high returns and only lose value after a certain point are also bad for investor emotion, even if the total returns from the start remain above those of a safe investment. This property makes measurement of maximum negative outlook by means of the second option insufficient as well.

Measuring the most consecutive losses seems to be an even more reliable way of measuring investor emotion as it captures long periods of negative returns but does not necessarily yield bad results after a peak in the overall returns. One downside of measuring investor emotion in such a way is the lack of measurement of total returns. Therefore this method is valid for measuring the investor emotion, but only if combined with other metrics that ensure the total returns are solid as well. Combining this method with a Sharpe or Sortino ratio could fulfill this requirement.

5.6 Input stability / 'Data Mining' prevention

It is also important to measure the level of 'data mining' an algorithm executes. An example of a bad algorithm that uses 'data mining' is an algorithm that will base its decisions on the date. Using Genetic Programming such an algorithm would likely develop into a strategy that bases its prediction on date T mainly on T, effectively over-fitting on the period which the fitness function measures in a way that is not likely to maintain success in a future period. This is due to the fact that such an algorithm would be mapping separate events rather than actual behavior.

Testing an algorithm during the time the fitness function uses to measure the performance of algorithms can be described as in-sample testing and testing an algorithm with data from dates outside those used in the fitness function that was used to construct the algorithm is called out-of-sample testing as described by Swinkels[1]. The input stability indicates how closely related the in-sample and out-of-sample results are.

Measuring input stability as part of the fitness function is slightly complicated as comparing results from in-sample and out-of-sample testing basically renders both tests in-sample tests. To cope with this problem, it is important to construct the algorithm in such a way that they are likely to have a high input stability, without actually measuring it.

5.6.1 Prevention

One way to ensure input stability is prevention. If the genetically constructed programs are not able to acquire too much information about events, they will also be unable to use information on separate events and will be much more likely to start mapping behavior instead. It is not possible to eliminate all event information in the inputs given to the algorithms, as all information can be seen as event data in some way. However, by using a form of Genetic Programming with relatively large 'operator' nodes, it is possible to ensure the nodes do not extract too much detailed information of particular events. It should be noted prevention does not provide a good method for comparison of generated algorithms as they were all constructed with the same preventive measures.

5.6.2 Measurement of Branch Coverage

Different generated algorithms could be compared to each based on their input stability by means of branch coverage measurement through time. This could be achieved by creating one large function containing the code from all 'operators' of the genetically created programs as well. The level of branch coverage through time would be a relatively good measurement of input stability. Event-mapping algorithms would be more likely to have branches that are only invoked during certain events, whereas behavior-mapping algorithms would be more likely to mainly consist of branches that are covered more frequently.

5.6.3 Measurement of Diversification Potential

Another method to compare the input stability of two genetically generated financial forecasting algorithms is performance measurement after applying the algorithm on different stocks. Given an algorithm that trades the stocks S1…Sn, the effect swapping Si (1 <= i <= n) for another random stock has on the algorithm performance determines the input stability by means of diversification potential measurement. An algorithm with a high diversification potential measurement is less likely to be an event-mapping algorithm as it could not be mapping events for individual stocks, but only for the entire market or at least a large segment of it. Events that occur in the entire market are already an aggregate of individual stock events and are therefore more likely to be the linked to market behavior instead.

5.6.4 Precognition Interaction Paradox

Another important aspect of input stability that needs to be considered is the way in which usage of the resulting algorithms will affect the future inputs for this algorithm. If an algorithm for financial forecasting turns out to be very effective and starts being used to manage a significant investment, the future stock prices are likely to be influenced by the predictions of this algorithm. This is due to the Precognition Interaction Paradox: "The ability to change the future state of any object at time T is mutually exclusive with the ability to predict the state of the object at time T with certainty". This holds true because if it was not, a machine could be constructed that knows the state of the object at time T and would change it to something else, leading to a contradiction. This same effect applies to the usage of forecasts, as soon as they start being used in a way that will influence the future inputs of the algorithm, the quality will decline.

6. Genetic Programming

Genetic Programming (GP) is a machine learning technique from the field of Artificial Intelligence.[13] In GP, operations inspired by natural evolution are used to construct syntax trees representing an entire program. The technique is a specialization of the broader class of Genetic Algorithms.

In GP, there are several steps which have to be repeated to eventually construct an algorithm which performance converges to the best possible performance. First of all, GP works with generations. These generations consist of dynamically constructed programs that can be modeled as syntax trees. To measure the quality of such a program, a fitness function is needed, which can consist of (combinations of) all kinds of measurements, depending on the expected outcome of constructed programs.

When a GP algorithm starts, it first starts with constructing an initial population of M programs, which is done by randomly creating syntax trees. Next, a sequence of executions follow which will be repeated for N generations. First of all, all the programs of the current generation have to be rated using the fitness function and are then ordered accordingly. Secondly, a program is picked out of the current generation, where programs with a higher fitness function are more likely to be picked. This program can then be either duplicated, mutated or combined with another program of the generation. The result is then put into the next generation.

These steps will be repeated until the next generation consists of M programs, after which the same steps will be executed for the next generation and so forth. After N iterations, a generation was constructed which should have been converged to the maximum fitness value possible.

In our situation, this principle is extended in several ways. First of all we have several modules which, combined, should be able to construct any currently known stock trading strategy. Not all modules have the same outcome, nor need the same (numbers of) input, which means the generation, mutation and combination of trees is slightly more complex. Secondly, our fitness function isn't purely based on the total profit that a strategy would make, but also on several other measurements which are mentioned above in chapter 5. These metrics will be calculated by evaluating the generated programs with the use of the stock data that is scraped from the web.

Finally, we will introduce several functionalities to tweak the performance of the GP algorithm, such as penalties for strategies which are too complex. Eventually, in theory, this should lead to the optimal stock trading strategy, depending on the input given such as available stock symbols, which should outperform all other strategies.

[13] *Genetic Algorithms*, 2011-06-20, accessed 2011-06-22, <http://en.wikipedia.org/wiki/Genetic_algorithms>

7. Framework Research

Before implementing the application, some research on Java and PHP frameworks had to be done to investigate if these might give any advantages during the implementation and testing phase. This means research had to be done on frameworks for both coding and testing in Java and PHP. The problem with most frameworks is that they can make the coding much easier, however they are usually full of, in this context, unnecessary features, which will only cause the application to run slower. Therefore the framework has to be selected according the following criteria:

- It has to be light-weight (as few unnecessary features as possible)
- It should have features which makes coding easier
- It should be easy to set up, learn and use

Below, you can find the frameworks we found, followed by a brief evaluation of the reasons why this framework might be of use.

7.1 Java Development Frameworks

7.1.1 Hadoop

Hadoop[14] is a framework which is specially made for processing large quantities of data. This is used by high-profile companies such as Google and Facebook, which process quantities of data that are too large for a regular database. This is done by distributing tasks over several PC's, using the map and fold function, known from functional programming languages. This framework requires a lot of work to set up and extensive knowledge of Java, network structures and Linux. Even though the application might be processing a lot of data, it can still be managed by a regular database, therefore it would contain more features than required, causing the application to run unnecessary slow.

7.1.2 NetBeans Platform

NetBeans Platform[15] is a framework which supports the development of desktop applications in Java. Because the the forecasting application will be executed on a server, without any graphical user interface, this framework is probably not suitable for developing the application.

7.1.3 Spring Framework

Swing[16] is a decent framework for Java programming, which is used by a lot of programmers. It comprises several modules and services such as batch processing, data access (database communication), messaging and testing. However, it seems like a framework that is generally used for smaller applications. The developers state that your application shouldn't be depending on large frameworks with huge configuration files, which only causes the system to run slow. They try to achieve this by means of a structure which consists of one main container doing most of the work such as gathering data and then using several relatively simple classes to do the actual calculations.

---

[14] *Hadoop*, 2011-06-21, accessed 2011-06-22, <http://en.wikipedia.org/wiki/Apache_Hadoop>
[15] *NetBeans*, accessed 2011-06-22, <http://netbeans.org/features/platform>
[16] *Swing Framework*, accessed 2011-06-22, <http://sourceforge.net/project/stats/?group_id=73357&ugn=springframework>

Despite the useful features, this is not suitable for the forecasting application, since it is a modular system where each methodology has its own module and its own responsibilities.

7.1.4 Hibernate

Hibernate[17] is rather more an ORM library than a framework, but this might still be of use. It makes the interaction with the database much easier, since you don't have to worry about JDBC connections and SQL queries each time you want to retrieve data from the database. Since there will be a lot of interaction with the database, an ORM library can be of great use. However, the disadvantage of Hibernate is that it is a very complex system. This means that it can cause the application to run slow and that it is not very flexible. This can be a great disadvantage, since data has to be stored in the database in a format which can also be used by the PHP application, without too much effort.

7.1.5 MyBatis

The MyBatis[18] ORM library is comparable to Hibernate, however it beats Hibernate in simplicity and configurability. This means that it can ease the effort to use a database as well as Hibernate, but in contrast to Hibernate, this library is light-weight. This is also because you have to configure the executed SQL queries yourself in XML. This can be seen as a disadvantage, however in the case of the Forecasting application, it can work as an advantage because it makes it easy to store data in such a format that it can also be easily retrieved by the PHP application.

7.1.6 Spine

Spine[19] is a framework designed to ease the portability of an application, based on multiple data stores. However it might be useful to run the script on multiple servers if it scales to such a size that the execution time runs out of hand, not many servers will be necessary, so the portability is not really an issue here. Therefore, this framework is not suitable for the forecasting application.

7.1.7 Google Guice

Google Guice is an improved version of Spring but with the same underlying structure [14]. One of the improvements is the addition of Java annotations support, however this doesn't make it more suitable to the forecasting project.

When designing the software, it is advised to use an MVC structure. Since the forecasting application doesn't have a view, this comes down to dividing the application in controllers and models. No frameworks were found that are really suited for the Forecasting application, however the ORM MyBatis has some significant benefits. Therefore, models of the application will be done by MyBatis, leaving the controllers to be implemented manually.

[17] *Hibernate*, accessed 2011-06-22, <http://www.hibernate.org>
[18] *MyBatis*, accessed 2011-06-22, <http://www.mybatis.org>
[19] *Spine*, 2008, accessed 2011-06-22, <http://spine.zphinx.co.uk/overview.html>

7.2 PHP Development Frameworks

7.2.1 CakePHP

CakePHP[20] is a popular MVC web framework, in the style of Ruby on Rails. It provides a fixed directory structure, where the user can plug in the relevant application controllers, models, etc. Its distinguishing features are its ease of use, decent documentation and PHP4 support. However, experience among our team members tells us that it is also limited in flexibility, and its PHP4 backwards compatibility makes its architecture somewhat dated.

7.2.2 Kohana

Kohana[21] is a web framework with a very elegant architecture, using a so called HMVC (Hierarchical MVC) pattern, where a page's output is built up from multiple requests to controllers. Nonetheless, Kohana suffers from lack of good documentation, and changes its API so much that there is a lot of incompatible information and broken plugins out in the wild.

7.2.3 Alloy

Alloy[22] is another HMVC framework that is very light-weight and has better documentation than Kohana. It is however a very unknown, immature framework, and none of us has had previous experience with Alloy, making it risky to use.

7.2.4 Flourish

Flourish[23] is not really a framework, but more of a library, even though it provides much of the functionality of a framework, such as an ORM, anti-XSS security features and user authentication. It calls itself an "unframework" for this reason. Flourish is excellently designed, but does not immediately give you a full-fledged application.

7.2.5 Roy

Roy[24] is a minimal HMVC web framework that provides only the essential features an application programmer needs to set up an application. It includes such features as error/exception handling, URL routing and a standard directory structure for a very modular application. Roy makes it easy to integrate various third-party libraries in an MVC environment. Roy is a custom framework written by one of our team members after a frustrating experience with Kohana.

To get the benefits of an HMVC architecture, with enough flexibility and power to suit our needs, we've decided to use the Roy framework along with Flourish and PHP ActiveRecord (an ORM).

7.3 Java Testing Frameworks

During the study of Computer Science, students are frequently taught to test their programs using JUnit. This framework has everything one might need to make unit tests and is therefore one of the most used testing frameworks for testing java applications. Basically, JUnit would be suitable enough to test the

[20] *CakePHP*, accessed 2011-06-22, <http://cakephp.org>
[21] *Kohana Framework*, accessed 2011-06-22, <http://kohanaframework.org>
[22] *Alloy*, accessed 2011-06-22, <http://alloyframework.org>
[23] *Flourish*, accessed 2011-06-22, <http://flourishlib.com>
[24] Krause, M., *Roy*, 2011-05-03, accessed 2011-06-22, <https://github.com/mkrause/roy>

forecast application, however, it might also be interesting to do some research on other Java testing frameworks. Bellow, the different testing frameworks are discussed and compared to each other.

7.3.1 JUnit

JUnit[25] has more than once proven its benefit when implementing applications in Java. During the course "Software Quality and Testing" this language has been extensively investigated, giving experience and insight in all functionalities JUnit offers to test every aspect of the implemented software. This testing framework can therefore be used as a benchmark when investigating other testing frameworks.

7.3.2 TestNG

TestNG[26] is inspired by JUnit, however with several extensions. While still giving the possibility of writing ordinary JUnit unit tests, this testing framework also has several features that facilitate support for parameters, annotations, integration testing, dependent methods for application server testing and the testing of multi-threaded applications.

7.3.3 JWalk

JWalk is a framework that automatically generates tests according the feedback of the developer. When testing a class, JWalk creates several tests according to the code and presents the outcome to the developer. The developer then gives feedback (pass or fail), which will be used by JWalk to construct an oracle. After giving feedback to several tests, JWalk claims to be able to construct an oracle which scores over 90%, which means that 90% of the generated test cases will be evaluated correctly[10]. Although this might save a lot of time, this testing framework might not be suitable. This is because this framework is especially aimed at testing the underlying state machine of a class and it can in some cases also generate an oracle with a score of only 40%[10]. This might have catastrophic consequences for our application.

7.3.4 DbUnit

DbUnit[27] is a testing framework, or rather just a library, that resets your database to a known state between tests so tests concerning the database won't influence each other by leaving a corrupt or messy database. However this can be a valuable functionality, this is not very necessary, since this is actually one function which you can easily implement yourself.

7.3.4 Other Testing Frameworks

A lot of other testing frameworks surfaced during the research, which were all inspired by JUnit, but didn't contain all the features that TestNG contains. Some of the testing frameworks that were found are Cactus, MultiThread TC, Fit, JTiger, Arquillian, JUnit EE, GroboUtil and Unitils.

[25] *JUnit*, accessed 2011-06-22, <http://www.junit.org>
[26] *TestNG*, accessed 2011-06-22, <http://testng.org>
[27] *DbUnit*, accessed 2011-06-22, <http://www.dbunit.org>

7.4 PHP Testing Frameworks

7.4.1 PhpUnit

PhpUnit[28] is a testing framework in the style of JUnit. It is the most popular PHP testing framework, and is well documented. It provides all the basic testing functionality you'd expect, as well as some nice-to-haves such as mocking and automated test generation using annotations.

7.4.2 SimpleTest

Another popular testing framework for PHP is SimpleTest[29]. This framework does not seem to be as stable and well-documented as PhpUnit, but does include some useful features such as a "Web tester", which allows you to test generated HTML output very easily.

[28] *PhpUnit*, 2011-06-16, accessed 2011-06-22, <https://github.com/sebastianbergmann/phpunit>
[29] *SimpleTest*, accessed 2011-06-22, <http://www.simpletest.org>

Literature

1. Interview with Laurens Swinkels of Robeco Investment Solutions, held 2011-03-02.
2. Bollen, J. et al., *Twitter mood predicts the stock market*, 2010-10-14, accessed 2011-06-22 <http://arxiv.org/abs/1010.3003>
3. Sprenger, T.O., Welpe, I.M., *The Information Content of Stock Micoblogs*, Tweets and Trades, 2010-11-01, accessed 2011-04-20.
4. Grant V. Farnsworth et. al., *Successful Technical Trading Agents Using Genetic Programming*, October 2004, accessed 2011-04-30
5. Getmansky, M., Lo, A.W., Makarov, I., An Econometric Model of Serial Correlation and Illiquidity In Hedge Fund Returns, 2003-04-28, accessed 2011-05-15.
6. Investopedia, *Understanding The Sharpe Ratio*, 2010-07-28, accessed 2011-06-21 <http://www.investopedia.com/articles/07/sharpe_ratio.asp>
7. Le Marios, O. et al., *Return smoothing practices: a potential threat for alternative investment growth*, September 2007, accessed 2011-04-20, < http://www.thehedgefundjournal.com/magazine/200709/technical/a-potential-threat-for-alternative-investment-growth.php>
8. Magdon-Ismail, M., Atiya, A., *Maximum drawdown*, Risk magazine, 2004-10-01, accessed 2011-04-20.
9. Abdulali, A., *The Bias Ratio Measuring the Shape of Fraud*, 2001, accessed 2011-06-10.
10. Simons, A.J.H., *JWalk: a tool for lazy, systematic testing of java classes by design introspection and user interaction*, 2007, accessed 2011-04-20 <www.springerlink.com/index/n2l78k8777262351.pdf>
11. Sharpe, W.F., *The Sharpe Ratio*, Stanford University, Reprinted from The Journal of Portfolio Management, Fall 1994, accessed 2011-04-10.
12. *Followers and Foes: Defining Industry Groups with Twitter*, 2011-03-15, accessed 2011-06-21 <http://tweettrader.blogspot.com/2011/02/followers-and-foes-defining-industry.html>
13. Deiters, E., *Kun je echt rijk worden door Twitter te volgen?*, 2011-04-21, accessed 2011-06-21 <http://www.depers.nl/economie/563206/Kun-je-echt-rijk-worden-door-Twitter-te-volgen.html>
14. *Google Guice – Spring Comparison*, 2010-02-04, accessed 2011-06-22 <http://code.google.com/p/google-guice/wiki/SpringComparison>

Appendix J – Requirements Document

Summary

This document documents the requirements analysis of the Bing Forecast project. This is done by first evaluating the current system, which consists of stock trading according to intuition or tools. While some traders listen to their gut feelings, professionals trade according to tools which give information about the current stock market or the companies behind the stock market. These two trends are called chartism and fundamentalism.

Both these trading strategies have to be either used or outperformed by the application that will be developed. This application has several functional requirements, such as the possibility to provide additional analysis for the generated strategy, and non-functional requirements, such as the requirement to outperform all other forecasting algorithms. These requirements are then prioritized according to a MOSCOW table.

To illustrate the actual usage of the application, some use cases are described in section 3.4, which should, in combination with the functional and non-functional requirements, cover all needs of Bing Technology, concerning the implementation of the application.

1. Introduction

Bing Technology seeks to develop a general framework that can be used to produce specialized forecasting tools useful across industry verticals. Initially this will be specified to two subject domains: stock market forecasting and the Heritage Health Prize.

Bing seeks a framework that can ideally be repurposed for other tasks such as product recommendations and other predictive domains. This would happen in a possible followup phase 2 product, as the initial product will be specialized to stock market predictions.
After consulting Dan Guy, several requirements were obtained, which will be presented in this document. These requirements will ensure that the product will meet the demands of Bing Technology, which will be done according to the following chapters.

First, methods currently used for forecasting will be analyzed. Accordingly, the proposed system will be treated, as well as why this is an improvement on the current system. This will be done according to functional and non-functional requirements, followed by several use cases to illustrate the behaviour of the application.

2. Current system

The current “systems” that are used for forecasting are usually not digital systems, but systems based on personal experience and analytical tools. This comes down to gut feelings and intuition, which can differ per person. For instance in the case of stock market forecasting, a lot of traders trade according to their personal intuition and interpretation of the stock price.

In contrast, some stock traders use various tools to analyze and quantify the current stock prize and trade according to these tools, which is how professionals usually earn their living. Here, a distinction can be made between two different kind of traders: fundamentalists and chartists. Fundamentalists analyze the actual company and determine the value of the stocks accordingly. If this value is above the actual stock price, they would advise you to invest in this company, and vice versa.

In contrast, chartists rather analyze the chart of a stock. They do this according to different tools like Sharpe ratios, extrapolation, etc.

In the new proposed system, which will be described in the next chapter, all of these techniques that are currently used should be used by the application, if this enhances the performance of the forecasting algorithm.

3. Proposed system

3.1 Overview

As mentioned before, Bing Technology seeks to develop a general framework that can be used to produce stock market forecasting tools, which outperforms conventional forecasting algorithms. This can be adapted to the needs of future customers, so they can use it for their business. The general idea is that a customer submits a task to the application to generate a strategy which should outperform all other strategies. After the strategy has been created by the application on the server, the user can access this strategy and its statistics, advice and structure via the web interface.

The different requirements are viewed below in a MOSCOW-table, together with any dependencies to prioritize the different requirements. Additional information on these requirements can be found in sections 3.2 and 3.3.

	Must have	Should have	Could have	Won't have	Dependencies
1. Generate strategies	X				6
2. Give advice for portfolio composition from generated strategy	X				1, 10
3. Generated strategy should be acceptable to investor		X			1, 2, 9
4. Multiple inputs	X				10
5. Usage of a stateful algorithm		X			
6. Ability to put constraints on the results	X				
7. Run on server, accessed by web interface		X			
8. Provide additional analysis		X			1, 2
9. Analyze trading strategy for all / individual stocks		X			8
10. Standard input and output	X				

11. Use the world as input			X		
12. Make generated strategies transparent			X		1
13. Outperform all other traders			X		1, 2,1 3.1, 13.2
13.1 Generate accurate predictions		X			1
13.2 Reliability		X			
14. Innovative algorithms	X				
15. Ease of use		X			7

3.2 Functional requirements

1. Generate strategies. First of all, the application should provide the functionality to generate a strategy according to the stock symbols and the allowed modules the user provides.

2. Give advice for portfolio composition from generated strategy. Based on the presented result of the forecasting algorithm, the application should be able to propose a portfolio composition to the trader.

3. Generated strategy should be acceptable to investor. The strategy generated by the application should be acceptable to investors, which means that it will generate a strategy which they are confident about. This means that the strategy for instance must have a Sharpe ratio that is acceptable to the investor, high input stability and the maximum negative outlook should be limited.

4. Multiple inputs. The forecasting algorithm can base its results on multiple time series and can trade with composed portfolios consisting of more than one stock symbol.

5. Use of a stateful algorithm. As the application could base its generated algorithm on, for instance, the current value of the portfolio, the algorithm generated by the application should also take the results of its previous forecasts into account.

6. Ability to put constraints on the results. When the application generates an algorithm, it is possible it produces a value which can not be used to give an advice to the trader. It is therefore necessary to implement the ability to put constraints on the result of the algorithm.

7. Run on server, accessed by web interface. The application can get very complex and therefore heavy weight to run. In the worst case scenario it might need a lot of computational power, which might cause a heavy load on the machine of potential customers. Therefore, the forecasting framework should be

executed on a central server that can manage the application and generate strategies in a reasonable time without any negative side effects. The generated strategies can then be accessed and reviewed through a web interface. Further advantages of this approach is that the application is portable to different machines, light-weight for the customer to use and easy to install.

8. Provide additional analysis. When the forecasting framework has generated a strategy, which is then accessed by the customer, the customer might want additional analysis to get an indication of the quality of the generated strategy.

9. Analyze trading strategy for all / individual stocks. Depending on the customers needs, the application needs to generate a trading strategy that performs well for one or more stocks. It is therefore needed to analyze the performance of the generated strategy on all stocks and on any specific stocks.

10. Standard input and output. The application should give a certain type of output when a certain type of input is given. This means that the application should always be able to generate a portfolio composition based on the generated strategy, as long as stock value time series, the current state and possibly other time series such as weather reports are given as input.

11. Use the world as input. The application could take advantage of the information that is provided on the internet nowadays. This means it can make use of time series concerning for instance the weather, stock quantities of stores and maybe even social media like Twitter and Facebook.

12. Make generated strategies transparent. To gain the confidence of the investor, it is important to make the generated strategy transparent, so the investor can see why certain advice is given.

3.3 Non-functional requirements

13. Outperform all other traders. In order to develop a product which can be sold by Bing Technology, our application should outperform all other traders, or at least enough to become a viable product. This can be done by making use of the same techniques that traders use, but can also be done by using or generating algorithms that generate better results than these algorithms

13.1 Generate accurate predictions. In order to outperform all other algorithms, accurate predictions have to be made by the algorithm. This can be defined as the difference between the prediction generated by the algorithm and the actual portfolio composition of the best case scenario.

13.2 Reliability. To outperform all other algorithms, it is necessary to develop a reliable application. If for instance the application only performs well in 25% of the time, it will not be of any use for customers, since they will then be better of by asking a conventional trader to trade for them.

14. Innovative algorithms. Since this is a research project for as well Bing Technology as ourselves, innovative algorithms have to be used to investigate what the state of the art algorithms in the field of computer science are capable of.

15. Ease of use. The application should be simple to use so all potential customers can understand it and use it to their needs to actually save effort.

3.4 Use case models

To give a general idea of the required functionalities of the forecasting application, different use cases are described in this section. All use cases will start with a short scenario description, followed by a

systematic representation of the actions of the user and the executions made by the system. In all of these use cases there is only one actor: the investor, also known as “user”.

Create new strategy
To start trading, the investor first needs to submit a task to the application to generate a new strategy. This has to be done by using the “add strategy” functionality in the web interface, in which the user has to select the stock symbols of interest and the modules that the application is allowed to use to generate a suitable strategy. After the task has been submitted, the user can close the application and check if the server has constructed a strategy later.

User	**System**
Start application	Show welcome screen
Navigate to strategy overview screen	View all generated strategies (if any)
Click on “add strategy” button	View form for generating a new strategy
Fill out form Strategy name Stock symbols of interest and sets of stock symbols of interest Allowed modules	
Submit form	Generate and execute task

View advice of a strategy
When an investor is planning to trade his stocks, he first starts the application to view the advice that was generated by a created strategy. This is done by opening the strategy overview screen and clicking the strategy he wants to use.

User	**System**
Start application	Show welcome screen
Navigate to strategy overview screen	View all generated strategies
Click a strategy of interest	If generated strategy is a broadly applicable strategy, ask for stock symbols to apply the strategy on
If necessary, provide stock symbols to provide the strategy on	Generate and view proposed portfolio composition

View strategy performance

After a strategy has been generated, the performance of the strategy can be viewed. This is done by opening the strategy overview screen, clicking the strategy the investor is interested in and navigate to the performance tab. The performance is based on the scenario in which the trader would always change his portfolio to the portfolio composition the system proposes.

User	**System**
Start application	Show welcome screen
Navigate to strategy overview screen	View all generated strategies
Click a strategy of interest	View the advice for this strategy
Click the “performance” tab	View statistics of the strategy, such as average sharpe, best sharpe, profit, etc...

View structure of generated strategy
After a strategy is generated, the investor might want to know the logic behind the strategy, so he knows if he can trust the algorithm. This can be done by opening the strategy overview page and navigating to the “structure” tab of the strategy of interest.

User	**System**
Start application	Show welcome screen
Navigate to strategy overview screen	View all generated strategies
Click a strategy of interest	View the advice for this strategy
Click the “structure” tab	View the structure of the generated strategy.

Appendix K – Plan of Approach

1. Introduction

In the fourth quarter of the 2010-2011 academic year, we – a group of three Computer Science students at Delft University of Technology (DUT) – have decided to work on our BSc. Project at Bing Technology, a software company located in the Philadelphia area in USA.

One of our team members has had previous communication with Bing Technology, and they have generously agreed to host us for 9 weeks in Conshohocken, PA as we work on the project on location.

The project itself has been decided on, and will entail the construction of a stock market forecasting application using an innovative algorithm we will need to develop. This stock market application will serve as a proof of concept for a general forecasting framework that can be widely applied to predictive domains.

This document functions as the plan of approach for our project. The sample Plan of Approach (PoA) provided by the Delft University of Technology (DUT) was used as the base of this document, except that sections were modified or omitted where they did apply to this project.

2. The Problem

2.1 Problem Description as Given by Bing Technology

The project will be performed for the software company Bing Technology, LLC. They have provided us with the following problem description:

Bing Technology seeks to develop a framework that can be used to produce specialized forecasting tools useful across industry verticals. Initially, we are seeking to target two specific subject domains:

1) Stock Market Prediction
2) Heritage Health Prize

Bing seeks a framework that can ideally be repurposed for other tasks such as product recommendations and other predictive domains. This would happen in a possible follow up phase 2 project.

Note that there are two different kinds of products mentioned in this problem description. Two specific forecasting *applications* and a generalization of these applications in the form of a software *framework*. Implicit in this description is the creation of the underlying *algorithm*.

Note that the Heritage Health Prize (HHP) application mentioned in the description has – after deliberation with Bing Technology – been dropped after reconsideration, due to time constraints and unacceptably strict licensing terms enforced by the hosts of the HHP.

2.2 The Goals

The end goal is to produce an innovative forecasting algorithm that can be used to forecast a wide range of predictive domains, along with an application of this algorithm in the form of a stock market forecaster.

The generalization into a general framework is a secondary goal, which would be nice to have but is not essential to the project.

To summarize, the main focus of our project will be to develop the algorithm in the context of an application that can forecast the stock market with high accuracy. This application will serve as a proof of concept that can potentially be generalized into a software framework.

2.3 Deliverables for Delft University of Technology

Delft University of Technology's (DUT) main interests in our project relate to the process itself. To this end, DUT will receive the following documents:

- Plan of Approach (April 21st, 2011)
- Orientation Report (April 21st, 2011)
- Announcement of Commencement (May 2nd, 2011)
- Two-weekly progress reports (one every two weeks)
- Requirements Document (July 8th, 2011)
- Design Document (July 8th, 2011)
- Final Report + own evaluation (July 8th, 2011)
- Final Presentation (July 8th, 2011)
- Evaluation by Bing Technology (July 8th 2011, possibly later)

In addition, the SIG will twice receive a copy of our code for review, as arranged by DUT.

2.4 Deliverables for Bing Technology

The deliverables for Bing Technology are:

- Proof of Concept (May 24th, 2011)
- Finished product (July 8th 2011)
- Documentation in the form of a User Manual (July 8th 2011)

The Proof of Concept (POC) will need to be able to show that our algorithm works well enough in forecasting the stock market, such that Bing Technology is convinced it will be able to sell this product. The POC need not be 100% complete, it may lack some features or interface elements. It is mainly intended as a point of evaluation.

After the POC has been delivered to Bing Technology and evaluated, we will make the final planning for the finished product. Depending on the results, Bing Technology may decide to have us continue work on the stock market application, or direct us to work on another application or the general forecasting framework.

2.5 Technical Constraints

The technical constraints for this project are:

- The back-end must be written in Java
- The front-end must be written in PHP

Server requirements are not a big issue, in principle. Bing Technology will provide us a development server with generous hardware, and the actual production server(s) may have higher performance if necessary. The algorithm will not need to run in real-time, it may take several hours to run if needed. Of

course, efficient performance remains an important requirement, but high accuracy takes priority in our case.

2.6 Schedule Constraints

The schedule constraints for this project are:

- Proof of Concept, by May 24th 2011
- Completely finished, by July 8th 2011

2.7 Risk Factors

The entire project has several risk factors which should be taken into account. Many of these risks have been pointed out to us by Laurens Swinkels, an econometrist we interviewed prior to the start of our project.

'Data mining' effect / overfitting

The first risk factor is the 'data mining' effect, as described by Swinkels [1]. This effect occurs when too much incidental input data is used, causing any predictions made in the testing period to become mere mappings of events rather than derivatives of underlying behaviour.

'Black swan' effect

Another risk factor mentioned by Swinkels is the 'black swan' effect, meaning a completely unexpected event occurs. An example of this effect in stock market forecasting could be the sudden total collapse of a company which used to yield relatively stable results. Bankruptcy as such could easily be overlooked as a possible event influencing the stock prices, but many other unexpected events could happen as well. It is important to take this into calculation while coming up with reliable forecasting strategies.

Inability to achieve sufficient quality of forecasts

Another risk factor, which is in fact more of a challenge than a risk, is the quality of the results from forecasting tools based on the constructed application. The main target will be to create forecasts which are more reliable than any competitors, thereby making them more valuable. However, achieving such a quality of forecasts is difficult and there is no guarantee of this beforehand.

Inability to meet the planning

One of the toughest issues in software engineering projects, experience shows, is sticking to the original schedule. This can be caused by wrong time estimates or unpredictable factors. We try to avoid this by using a Scrum workflow, which is specially designed for getting projects done in time. Another way we tried to counter this problem is the introduction of some slack, allowing for small delays without real consequences. Despite all this, it is still possible that some phases will take more time than estimated. Consequently, it is important to set priorities in case this happens besides working harder of course.

As discussed in chapter 3 most phases build on the previous phase. This means that if we get behind on schedule, we cannot drop the current phase we are working on and move to the next phase. This is only the case for Phase 5 and 6, since these phases mostly consist of adding and implementing new strategies, which will be implemented as modules. After completing the implementation and testing of the current module, it is possible to move to the next phase. However, this will have to be discussed with Bing Technology.

If we get behind on schedule during any other phase we have to analyze why this happens and check if we can do anything about it.

3. Approach

3.1 File Sharing

We will use a shared Dropbox account to sync files such as documents, proceedings, research papers and design source files. Dropbox is free up to 2GB of data, and provides live syncing and even some version control.

3.2 Documentation

All documentation shall be done using Google Docs. This free tool allows us to easily collaborate on documents online, and also provides a limited form of version control.

At the end of each phase, all documents should be in a readable state. That is, it should contain no "to do's" and be reviewed for spelling and content by all team members. All documents will then be exported and archived as the end result of that phase.

3.3 Methodology

We will use iterative development for the entire project, building and releasing the product in several phases. For each phase we will use the V-Model to construct the final product, starting out by defining the global requirements, working our way down to module requirements and the final implementation. During the explication of the requirements of all stages, we will also define the test designs and if possible test specifications.

As far as the workflow is concerned we will use a variation of Scrum, with a brief status meeting at the beginning of each day and a finalization of a Phase roughly every week or so, allowing us to finish up all phases in time for the final deadline.

The specific forecasting applications we will be working on require a lot of knowledge about the particular subjects, which might not always be present within our team. To overcome this issue we will conduct interviews with experts on subjects such as stock market forecasting to gain further insights and find answers.

3.4 Techniques

The techniques we will use for the application are Genetic Algorithms (GA) and Genetic Programming (GP). We will use GA to estimate optimal values for strategy parameters for our software in Phase 3. In Phase 4 we will switch to a more elaborate form of evolutionary algorithms with an adapted version of GP. This adapted version of GP will be used to construct trading strategies that can dynamically adjusted.

3.5 Activities

The activities for this project will consist of:

- Research, consulting with experts, desk research and preliminary tests
- Talking with Bing Technology
- Gathering requirements
- Writing Plan of Approach
- Writing Orientation Report
- Writing Final Report
- Writing Requirements Document
- Writing Design Document
- Data mining, setting up sanitized databases for particular applications (stock prices, hospitalization data)

- Building general back-end in Java
- Testing, all frameworks, adjustments will be tested

3.6 Planning

Following the Scrum methodology, we have split up the project into several *sprints* of two weeks in length. Each sprint corresponds to one of the following phases:

- Phase 1: Orientation (week 1+2, April 18th - April 29th)
- Phase 2: Initial Prototype (week 3+4, May 2nd - May 13th)
- Phase 3: Proof of Concept (week 5+6, May 16th - May 27th)
- Phase 4: Elaboration (week 7+8, May 30th - June 10th)
- Phase 5: Final Product (week 9+10, June 13 - June 24th)
- Phase 6: Final Report (week 11+12, June 27th - July 8th)

Note that phase 4 is highly dependent on the results of the proof of concept. It may involve elaborating on the stock market application, or we may start to work on other applications or even generalize the software into a general forecasting framework.

Each phase should result in some product, something that is functional and able to be tested and evaluated. This product can consist of documents (an export of all current documents, which should be reviewed by all team members) and code (should successfully compile and be able to run and produce some valid output).

4. Project Design

In this paragraph we will specify how the project will be organized to meet the aforementioned requirements, constraints and planning. This will be done by specifying different aspects of the project such as organization, staff and finance.

4.1 Involved Parties

During the project, different parties are involved with different interests. First of all there is Bing Technology, who would like to see some application which they can sell. However, they think it is also important to document the program well, so further development and extension of the program can be done by other developers if needed. Besides that a proof of concept is desired by the end of May so Bing Technology will know what to expect of us and help steer us in the right direction if needed.

Secondly, the Delft University of Technology also has certain interests in our project because they also benefit if we succeed. Besides a possible contribution to their reputation, they might also gain a new partner to send future students for other Bachelor Projects.

Last but not least, we are involved ourselves. Besides passing our Bachelor Project for the Delft University of Technology and gaining the experience of working at a company abroad, we also got some certain interests and obligations towards Bing Technology during this project. However we don't get paid according to our performance, we still got a moral obligation to deliver a decent application, or at least a firm basis for future developers to finish this product, since they were so kind to give us this great opportunity and even took care of several facilities. Besides that, there is also a chance that Bing Technology would contact us for future work if they are satisfied with our work.

4.2 Organization

The following organization structure is derived from the interests of the different parties that were described above.

First of all we will keep a flat hierarchies within our group, because there is only the three of us who will develop this project. During the project we will set up the tasks together and will divide them equally. This will be done according to personal preferences and experience so a task will be done by someone who is preferably both motivated and qualified, since this will increase the overall efficiency.

During the project we can count on assistance of Dan Guy and Augie Chung. Dan Guy is the CEO of Bing Technology and has much experience in project management and Java development. Besides helping us with overcoming obstacles, he also keeps a close eye on our progress to help us in the right direction if necessary. Augie Chung has a lot of experience in the stock market, so he can give us some advise about, for instance, different strategies.

Besides mr. Sodoyer we can also count on the assistance of mr. Wiggers from the Delft University of Technology for certain technical questions relating to Genetic Programming.

4.3 Staff

During the project, the Delft University of Technology and Bing Technology both impose some requirements towards us. As far as the Delft University of Technology is concerned, we have to complete a project that is “BSc-Project-worthy”, which means that the content of the project is related to our study, the required skill-level for the project is high enough, the amount of work has to be at 15 ECTS, including writing the reports, which comes down to 12 weeks of 40 hours a week. Besides that, we all have to have finished all the subjects of our Propedeuse (1st year of our study), at least 30 ECTS of subjects of our second year and the subjects “Software Engineering” and “ST4 project”.

As far as Bing Technology is concerned, we are required to have some experience in project management and software engineering. This necessary to get the project done in time and meet the standard of coding quality of the company. One of the most important demands though, is to retain the secrecy of the code and algorithm we will develop. A minor requirement is that we deal professionally with possible clients.

4.4 Administrative procedures

Before, during and after the project different administrative procedures have to be done.

Before the project

Disable public transport traveling card

By disabling our traveling card for the public transport in Holland, we can get a compensation from the Dutch government. This is only possible for the period we are staying abroad.

Check the worldwide coverage of your insurance

If your insurance doesn’t cover any calamities during our stay in the VS, you should get an additional insurance that covers this.

Fill out ESTA

To use the visa waiver commitment of the USA, we need to fill out the ESTA form online and wait for approval. This cost us 14 dollars per person, which has to be paid by credit card.

Get a credit card

If you don’t already have a credit card, you should get one before you go to the US, since it is not guaranteed that your ordinary bank card can be used over there.

Book a flight
Obviously, we need to book a flight to actually get to the US. We will fly from Schiphol (AMS) to New York JFK (JFK) and our trip back will fly from Newark airport (EWR) to Schiphol (AMS).

Apply for the FIS scholarship
The Delft University of Technology gives scholarships to students who do an internship at non-European companies. We have to fill out a set of forms and hand them in at the Central Student Administration, together with an affirmation of the internship coordinator of the TU, the agreements with Bing Technology and a copy of the invoice of our flights.

Get an international drives license
In theory, we should be allowed to drive any car in Pennsylvania with our Dutch drivers license, however there are rumours that some police officers might take you into custody if don't have an American drivers license, until everything is sorted out. To avoid this, an international drivers license can be bought at the "ANWB".

Passport
You are not allowed to travel to the USA if you only have an Identity Card, so if you don't have a passport yet, you should get one.

Aside from these administrative procedures, several facilities were arranged by Bing Technology, listed below.

Apartment
Bing Technology arranged an apartment at Shery Lake, which contains all facilities we might need during our stay, including gym, pool and more.

Car
Bing Technology will get us a car so we can travel to and from the office. We can also use the car for any other purpose, such as trips in the weekend. We only have to pay the fuel we use.

Internet
To work on the project and communicate with, for instance, Dan when he is not around, we will need mobile internet. The benefit of mobile internet is that it is relatively cheap and gives us internet access everywhere, such as in the apartment and at the pool. We have to pay for this ourselves.

Laptops
To work on the project, Bing Technology provides laptops which we can use during and after the project.

Office
Last but not least we need a place to work.

During the project
During the project the only administrative work that has to be done are the reports for the Delft University of Technology. Next to this document, the following documents have to be created:

Orientation Report

This report will contain all the research we've done beforehand, but also the research we will be doing during the project.

Announcement of commencement
As soon as we start working at the office, we have to fill out the announcement of commencement and send this to the Delft University of Technology. This has to be done within one week after commencement.

Two weekly progress report
Every two weeks, we have to make a little report about our progress. This report then has to be sent to our internship coordinator at the Delft University of Technology (B.R. Sodoyer).

Besides these documents, we are planning to make a little accounting system to keep track of all the costs we have to divide such as food supplies and mobile internet.

After the project
Final report
After we get back, we have to finish the final report. We are planning to write a part of it while we are developing in the US, however most of the report has to be written in Holland.

Evaluation of BSc. Project
When we finished the project, the Delft University of Technology would like us to fill out an evaluation form to improve the organization of the BSc Projects of the future.

Evaluation by Bing Technology
When we will be back in Holland, Dan has to fill out a form to evaluate the project.

Presentation
After we finished and submitted the final report, we have to present our project for the committee. This includes the client (Bing Technology), the Delft University of Technology mentor and the BSc Project coordinator.

Submit approved final report
After our final report is approved, we have to submit it to the Delft University of Technology repository.

These are all procedures that have to be done for the Delft University of Technology, however we might have to do some administrative work for ourselves as well when we get back. The following issues are dependent on your own situation and preferences.

Subscribe for subjects for next year
If you are going to do a minor track or maybe a master, you have to subscribe for these subjects.

Disable your credit card
If you don't like the idea of a credit card and don't think you might use it for some time, you could disable your credit card.

Stop the world coverage of your insurance

If you don't think you are going to travel to a country outside Europe for a while, you might want to stop the world coverage of your insurance.

4.5 Reporting

Communication is essential to the success of a project. We will therefore meet frequently with Dan to check if we are still heading in the right direction and if he has any tips for us. Besides that we will, despite the agile workflow, document thoroughly so other developers of Bing Technology are also able to understand the design and implementation of the application. This will in general be done by JavaDoc, while any information gained by research will be included in the orientation report.

4.6 Facilities

For the development of this application, several facilities are needed, which will all be provided by Bing Technology. First of all we will need a working space with laptops and internet access. Bing Technology has rent an office with internet access where we can work at and will provide laptops with extra external monitors on which we can implement the software. Besides that, they provide an SVN server on which the code will be placed and a dedicated server on which the program will be run when it is finished. As mentioned before they will also provide a car, an apartment and mobile internet.

5. Quality Assurance

Last but not least, the quality of application, as well as the underlying code, has to be assured in some way. In chapter 2 we already described the criteria and risks of this project, however in this chapter we will describe how this criteria will be met and how risks can be avoided.

5.1 Product Quality

First of all, what do we define as quality? Quality consist of different aspects, with different criteria and different priorities. The following aspects define the quality of this application.

Maintainability

Obviously, it is necessary to write clean and maintainable code so possible bugs can easily be traced and further development is enabled. Even though this might not seem very important for the end-user, it is very important for ourselves and Bing Technology because this influences the time spent on development and maintenance of the application. It therefore has a high priority. The criteria for writing clean code are not really clear, since writing clean code can save time at the end, however if you take it to the maximum, it will cost more time than it will save. The criterion is therefore defined at a level on which every developer would be able to understand and maintain the applications underlying code and design, though with as less effort required from the developer as possible.

Application performance

Secondly the application performance is very important. This might actually have the highest priority, since eventually Bing Technology wants to sell this application with as much profit as possible. The better the performance of the application is, the higher the value of program will be. However, performance is quite a general term. It can be specified into sub-terms.

Accuracy is of great importance to the overall performance of the application. This means the rate at which the application will overrule other predictions made by men or computers. The criterion for this

property is very high: our goal is to develop an application that will overrule all other human and computer made predictions.

Besides accuracy, response time is important. Despite the fact that this property has a high priority, the criterion is not very high. If we take for instance the forecasting specification of stock markets, it would be sufficient if you post a request for several stocks at the end of the day and get the results by the next morning. However if you have to wait a week for your result, the application would be fairly useless for the customer, not to mention that the result would be outdated.

Application usability
The usability of the application is also quite important, however not as important as aforementioned aspects. If the application gives an accurate outcome and contains all the features the customer needs, however isn't really user friendly, the application will still be very valuable. The criterion for this aspect is therefore that as long as the costumer can use the application according to it's needs, possibly with the help of a manual and with a reasonable response time, the application is considered as usable enough.

Application reliability
Finally, the reliability of the application is important. With reliability we don't mean if a stock trader can rely on the results of application to earn a living, this is considered as the accuracy of the program, mentioned above. With the application reliability we mean at what rate the application does what it is supposed to do. For instance, if the application is supposed to produce a result within an hour and the program still hasn't produced a result after a day, the program is considered as unreliable. The criterion of this aspect depends on the response time of the application. For instance, if the application will produce a result in several seconds, it is not a big problem if the problem sometimes fails to calculate a result, as long as this doesn't happen too often. However if the response time of the application is several hours, it can be very inconvenient if the application fails to produce a result.

5.2 Process Quality

Besides product quality, process quality has to be taken into consideration. A high process quality means time is efficiently used, leaving more time to extend the application, resulting in a higher product quality. In short this actually comes down to efficiency of resources such as time and material.

5.3 Evaluation

To achieve the criteria of the aforementioned quality aspects, we took several precautions. First of all Dan Guy will guide us throughout the entire project and keep a close eye on our progress as well as the produced code. Besides that we will have different people wit experiences in stock markets have a look at the produced result for their opinion on the performance and usability. Finally we will run several acceptance tests to test the applications reliability.

To ensure the process quality we will, next to asking feedback from Dan, frequently reflect on our work, attitude and use of resources.

References

[1] Interview with Laurens Swinkels

Appendix L – Request for Proposal

Objective
Bing Technology seeks to develop a framework that can be used to produce specialized forecasting tools useful across industry verticals. Initially, we are seeking to target two specific subject domains:

1) Stock Market Prediction
2) Heritage Health Prize

Bing seeks a framework that can ideally be repurposed for other tasks such as product recommendations and other predictive domains. This would happen in a possible followup phase 2 project.

Technologies
Bing would prefer that the back-end at a minimum be written in Java. PHP is acceptable for the front-end application(s).

Schedules
Bing desires having a proof of concept (POC) functional by May 24, with further iterations deliverable as needed depending on efficacy of the prototype and future arrangements between the selected vendor(s) and Bing.

Next Steps
Any interested vendors should submit a proposed methodology for accomplishing the aforementioned goals as well as a proposed cost and time schedule. This information can be forwarded to dan@bingtechnology.com