Application of the LCS Problem to High Frequency Financial Data

Kevin Ngan
Lady Margaret Hall
University of Oxford

A thesis submitted in partial fulfillment of the MSc in

Mathematical and Computational Finance

June 21, 2013

Contents

List of Tables

Chapter 1

Introduction

High frequency trading is now growing rapidly in the financial markets, and its strategies are widely used by investment banks, hedge funds and proprietary trading firms. In 2009, studies suggested that more than 60% of total trading volume in the US financial market is accounted by HFT and algorithmic trading; in 2011, the HFT firms still contributes to more than 50% of total market volume in the US. [1]

Unlike traditional trading, high frequency trading (HFT) uses advanced technology solutions and computer algorithms to open and close a position within a matter of seconds, or even fractions of a second. Therefore, instead of considering daily or hourly data, tick data is particularly important to the field of HFT. A tick data includes a time stamp, bid price, ask price, trade volume e.t.c., and data vendors such as Reuters transmits more than 275,000 prices per day of foreign exchange spot rates alone. [2]

High frequency trading strategies include market making, statistical arbitrage and low latency strategies, and all require a clean set of high frequency financial data to carry out statistical analysis and build suitable models. With such large data sets from various data vendors, it is likely that most sources contains different forms of "noise" or data errors. There are mainly two types of data errors,

1. Human errors: Errors directly caused by human data contributors, for different reasons,
    - Unintentional errors, such as typing errors
    - Intentional errors, such as dummy ticks produced just for technical testing

2. System errors: Errors caused by computer systems, their interaction and failures.

Further, different data vendors might have a different set of data. For instance, trade X is recorded at time $t = 1$ in data from data vendor A, but at time $t = 1.05$ in data form data vendor B. Tick data also arrives in irregular spaced time, and existing models or statistical apparatus are catered for regularly spaced data.

For example,

$t =$	0	1	2	3	4	5	6	7	8	9	10
Data Vendor A	ϕ	X_1	X_2	ϕ	ϕ	X_3	X_4	ϕ	ϕ	X_5	ϕ
Data Vendor B	ϕ	ϕ	X_1	X_2	ϕ	X_3	ϕ	ϕ	X_4	ϕ	X_5

Table 1.1: Data arrival time example

At time t, ϕ represents that there is no trade occurring. X_i represents the i^{th} trade in the data series. From now on, these notations will be used though out the paper.

One unit of time could represent only 0.01 seconds in real time, and clearly, trades X_i do not arrive regularly. Data vendors might not also provide synchronised data; this could be caused by the difference in speeds of servers.

Other than irregularly spaced data, human error also account a significant proportional of "noise".

For example,

	Bid	Ask
True value	\$1.3498	\$1.3505
Data	\$1.3498	\$1.3405

Table 1.2: Data errors

It is particularly important that the algorithms can compare the data of the same trade from different data vendors, and reject if data are not similar.

Computer algorithms are also required to be implemented to run in a matter of fractions of a second, and the filter needs to be computationally fast. In this thesis, several algorithms are proposed, and are both sequential and iterative, which uses the existing information base when a new tick arrives, with a minimum amount of updating.

Typically, before an execution of a high frequency trading order, a forecasting model is used to determine the execution position and size. However, such forecasting model requires a clean set of high frequency data in order to deduce the most reliable trading decision. Therefore, the data cleaning algorithm must work in real time and in an "online" manner such that as raw tick data streams in, it passes through the data cleaning algorithm before reaching the forecasting model; and such procedure is shown in the following table,

Data Stream
↓↓
Data Cleaning Algorithm
↓
Forecasting Model
↓
Trade and Risk Quantifier
↓
Execution
↓↓
Market, Profit/Loss

Table 1.3: Trade cycle in high frequency trading

L. Zhao et al. proposed a data cleaning method in 2002, "A New Efficient Data Cleansing Method" [3]. In this paper, L. Zhao et al. proposed an algorithm to eliminate duplicate records in a data set, and uses rolling window to improve the efficiency of the algorithm. For instance, suppose a data set D has 100 entries, and there are d duplicates in D, i.e. only $100 - d$ entries are distinct. Instead of conducting pairwise comparisons for all 100 entries, the algorithm is modified as follow,

1. Define a rolling window r, and let $r = 10$ in this case.

2. Conduct pairwise comparison for the first r entries in the data set D, and set the $(r+1)^{th}$ entry to be the *anchor record*, and conduct pairwise comparison

between the *anchor record* and the first r entries, and create a list of similarity scores; such similarity score $s \in [0, 1]$ is calculated using a function defined in the paper.

3. Then the rolling window moves up 1 entry, such that the window r covers from the second entry to the $(r+1)^{th}$ entry.

4. New entries moving into the rolling window will conduct pairwise comparison and compute similarity score s with the *anchor record*, and using such score and the previous list of scores, one can be showed that whether such record is a duplicate

By introducing the anchor record and similarity score, this data cleaning algorithm is much more efficient than traditional pairwise comparison. However, the setting of the problem in L.Zhao's paper is different from the one proposed in this paper.

Building on the work of L.Zhao et al., and using the idea of the rolling window and the similarity function, which will be introduced in Chapter 2, this paper translates and modifies similar algorithms in order to compensate for high frequency financial data. In L.Zhao's paper, such algorithm works with a static database such that the size D is fixed and the algorithm is implemented "offline". However, high frequency data should be cleaned in an "online" fashion; as data streaming in, this paper uses the idea of the rolling window to clean a data set which the size is strictly increasing for every unit of time, so that a cleaned data set could be fed into the forecasting model and hence an execution strategy in real time.

Instead of eliminating duplicates in data sets, one should compute the longest common subsequence and optimal alignments of two different high frequency financial data sets so that the forecasting model is only fed data confirmed by another data source. Moreover, the window used in Zhao et al.'s paper is based on the number of entries in the data set D, in order to apply this idea into high frequency data, the window will be based on time rather than length of data.

Let α and β be sequences with length m and n. A longest common subsequence γ is a common subsequence of α and β *iff* γ is a subsequence of α and also a subsequence of β. In this paper, the idea of computing the longest common subsequence, or LCS, is used to clean the data and obtain common data points for two difference

data sets.

There are several approaches to LCS-like Algorithms, in the literature including the algorithm by Needleman, Saul B. and Wunsch, Christian D in 1970 [4]. This algorithm performs a global alignment on two sequences, and it is commonly used in bioinformatics where the two sequences could be a DNA sequence or protein sequence. There are also other variants to compute the optimal alignment, including Hirschberg's Algorithm, named after Dan Hirschberg in 1975 [5].

However, these algorithms focuses on DNA or protein sequences and are generally not very efficient in computing optimal alignments for high frequency financial data. Most of the literature about optimal alignment or LCS is related to bioinformatics, while research papers of the application of LCS to high frequency finance are sparse and rare. As mentioned before, the main difference between data from bioinformatics and from high frequency finance is that high frequency data is irregularly spaced; the arrival of trade data can be modelled by a *Poisson process*, and the waiting time of each trade is *Exponentially* distributed. Therefore, one can expect a random number of ϕs to be inserted between each pair of trade data.

In this paper, several algorithms are proposed and modified from the previous literature, such as the Needleman-Wunsch Algorithm. The new algorithms in this thesis are particularly applicable to high frequency finance; the structure of the data sequences are exploited, and hence a more efficient algorithm is proposed specifically for high frequency financial data.

Chapter 2

The Longest Common Subsequence Problem

2.1 Longest Common Subsequence

The longest common subsequence is a classic problem in computer science, and has been studied extensively with many applications, including text pattern matching, speech recognition, and most commonly bioinformatics. Within bioinformatics, the LCS-like algorithm is used to extract homologies between different strands of DNA and protein sequence by aligning and comparing them.

We consider *sequences* α, β, γ,... of *symbols* on an *alphabet* $\Sigma = (\sigma_1, \sigma_2, ..., \sigma_s)$ of size s. A sequence is identified by writing $\alpha = a_1a_2...a_m$, with $a_i \in \Sigma(i = 1, 2, ..., m)$. The *length* of α is m. A sequence $\gamma = c_1c_2...c_l$ is a *subsequence* of α if there is a mapping $F : [1, 2, ..., l] \rightarrow [1, 2, ..., m]$ such that $F(i) = k$ only if $c_i = a_k$ and F is monotone and strictly increasing. Thus γ can be obtained from α by deleting a certain number of (not necessarily consecutive) symbols.

Let $\alpha = a_1a_2...a_m$ and $\beta = b_1b_2...b_n$ be two strings on Σ with $m \leq n$. We say that γ is a *common subsequence* of α and β *iff* γ is a subsequence of α and also a subsequence of β. The *longest common subsequence (LCS) problem* for input strings α and β consists of finding a common subsequence γ of α and β of maximal length. Note that γ is not unique in general.

2.1.1 Example

Let's consider two DNA sequences,

$$\alpha = actagtctac$$

and

$$\beta = cgacgtcgata$$

Then the optimal alignment of these two sequences would be

a	c	⊔	t	a	⊔	g	t	c	⊔	⊔	t	a	c
⊔	c	g	⊔	a	c	g	t	c	g	a	t	a	⊔

Table 2.1: The alignment of sequences α and β

Where ⊔ denotes a gap within the sequence. For two different sequences, gaps are inserted so that the two modified sequences are aligned.

There are several kinds of alignments, including global and local alignments.Suppose,

$$\alpha = gtgtactccagag$$

and

$$\beta = gtacccaag$$

,
then the global and local alignments would be

g	t	g	t	a	c	t	c	c	a	g	a	g
g	⊔	⊔	t	a	c	⊔	c	c	a	⊔	a	g

Table 2.2: Global alignment

A global alignment is a sequence alignment that extends the full-length of the sequences that are being compared. Global alignment procedures usually will produce an alignment that includes the entire length of all sequences including regions that are

g	t	g	t	a	c	t	c	c	⊔	a	g	a	g
⊔	⊔	g	t	a	c	⊔	c	c	a	a	g	⊔	⊔

Table 2.3: Local alignment

not similar, and can be made to produce meaningless alignments between unrelated sequences; on the other hand, a local alignment is a high scoring alignment between sub-sequences of two or more longer sequences. Unlike a global alignment, there may be multiple high scoring local alignments between sequences. Since the algorithm for producing local alignments is much more complex, therefore only global alignments are used through this paper.

In the context of cleaning high frequency finance data, the gloal alignmnet is used, along with the special case of optimal alignment, which is the LCS of α and β.

The LCS of these alignments is

$$\gamma = gtacccaag$$

2.2 Needleman-Wunsch Algorithm

The Needleman-Wunch Algorithm [4] was published in 1970. It performs a global alignment of two sequences α and β, and consequently produces their longest common subsequence γ. Following from above, again, α and β are defined as follow

$$\alpha = actagtctac$$

and

$$\beta = cgacgtcgata$$

Scores for aligned letters are specified for every pair (α_i, β_j) from the alphabet by a similarity matrix S. For the case of DNA sequences, S will have the form of the following table.

Where $S_{i,j}$ denotes $Score(i, j)$, and since $Score(i, j) = Score(j, i)$, the matrix S is symmetric; and " " denotes an empty string, such that any elements align with an empty string has a score of 0.

	a	g	c	t	⊔	""
a	$S_{a,a}$	$S_{a,g}$	$\cdots$		$\cdots$	$S_{a,""}$
g	$S_{g,a}$	$\ddots$				$\vdots$
c	$\vdots$					
t						
⊔	$\vdots$				$\ddots$	$\vdots$
""	$S_{"",a}$	$\cdots$			$\cdots$	$S_{"","}$

Table 2.4: General similarity matrix S

In the special case of LCS, the score matrix S is simply an identity matrix; if any alphabets in sequence α and β is matched, a score of 1 is assigned, otherwise it gains a score of 0. More general alignment methods are obtained by a different choice of score matrix S; for instance in the interest of DNA or protein sequences, the score matrix S might be given as follow,

	a	g	c	t	⊔	""
a	10	-1	-3	-4	0	0
g	-1	7	-5	-3	0	0
c	-3	-5	9	0	0	0
t	-4	-3	0	8	0	0
⊔	0	0	0	0	$-\infty$	0
""	0	0	0	0	0	0

Table 2.5: Particular similarity matrix S for DNA sequences

For simplicity, consider the LCS case where the score matrix S equals the identity matrix. An empty string " " is added in front of α and β for initialisation. Then the boundary values of the score matrix M is constructed; $M_{0,j} = \sum_{k=1}^{j} Score(\alpha_j, ⊔)$ and $M_{i,0} = \sum_{k=1}^{i} Score(⊔, \beta_i)$ for $i \in [0, ..., m]$ and $j \in [0, ..., n]$ where $m =$ the length of β and $n =$ length of α.

Then the following recursion is used to compute all the remaining entries in matrix M,

- Match: $M_{i-1,j-1} + Score(\beta_i, \alpha_j)$
- Delete: $M_{i-1,j} + Score(\beta_i, ⊔)$

		α										
		""	a	c	t	a	g	t	c	t	a	c
	""	0	0	0	0	0	0	0	0	0	0	0
	c	0										
	g	0										
	a	0										
	c	0										
β	g	0										
	t	0										
	c	0										
	g	0										
	a	0										
	t	0										
	a	0										

Table 2.6: The initialization of the score matrix M

- Insertion: $M_{i,j-1} + Score(\sqcup, \alpha_j)$

And $M_{i,j} = max(Match, Delete, Insertion)$.

By construction, a score of 1 is assigned if an entry from the two sequence is matched, and 0 otherwise. Hence, in order to compute the optimal alignment, one should optimise the score matrix M such that the last entry, $M_{m,n}$, is at maximum. This optimisation problem can be solved by dynamic programming, resulting in the Needleman-Wunsch Algorithm. Each entry of the matrix M corresponds the solution to a subproblem; for instance, let $\bar{i} < m$ and $\bar{j} < n$, the value of $M_{\bar{i},\bar{j}}$ is the optimal solution for $\bar{\alpha}$ and $\bar{\beta}$ such that $\bar{\alpha}$ is the subsequence of α with length $\bar{j}$ and $\bar{\beta}$ is the subsequence of β with length $\bar{i}$.

Bellman's Principle of Optimality states that a dynamic optimisation problem can be broken down into subproblem; the Needleman-Wunsch Algorithm solves the optimisation subproblem for every $i < m$ and $j < n$, and hence the value of $M_{m,n}$ solves the optimisation problem for α and β.

Using this formula, one can compute the matrix M. The score matrix M contains information for the optimal alignments, and the backtrack function can read out the alignments from the matrix M. Note that the value of $M_{m,n}$ shows the length of the LCS γ.

The backtracking algorithm first starts at entry $M_{m,n}$, and works backwards to determine the actual alignments that result in the maximum score. The following pseudo-code shows the backtracking algorithm,

1. **Initialisation**

   Before running the backtracking algorithm, the similarity matrix S needs to be determined. In this case, the special case, LCS, is chosen, so that the similarity matrix S is an identity matrix.

   $Score(\sigma_i, \sigma_i) \leftarrow 1$
   $Score(\sigma_i, \sigma_j) \leftarrow 0$ for $i \neq j$
   $i \leftarrow m$
   $j \leftarrow n$

   Since the backtracking algorithm starts at the entry $M_{m,n}$, indices (i, j) are set to (m, n), and (i, j) denotes the "position" of the backtracking algorithm.

2. **Backtracking**

while $i > 1$ **or** $j > 1$ **do**
  if $i > 1$ **and** $j > 1$ **and** $M_{i-1,j-1} + Score(\beta_i, \alpha_j) = M_{i,j}$ **then**
    $\alpha\_alignment = \alpha\_alignment + \alpha_j$
    $\beta\_alignment = \beta\_alignment + \beta_i$
    $lcs = lcs + \alpha_j$
    $i = i - 1$
    $j = j - 1$
  else if $j > 1$ **and** $M_{i,j-1} + Score(\alpha_j, \sqcup) = M_{i,j}$ **then**
    $\alpha\_alignment = \alpha\_alignment + \alpha_j$
    $\beta\_alignment = \beta\_alignment + \sqcup$
    $j = j - 1$
  else if $i > 1$ **and** $M_{i-1,j} + Score(\beta_i, \sqcup) = M_{i,j}$ **then**
    $\alpha\_alignment = \alpha\_alignment + \sqcup$
    $\beta\_alignment = \beta\_alignment + \beta_i$
    $i = i - 1$
  end if
end while

Within the *while* loop, there are three cases; each case represents the possible predecessor of $M_{i,j}$. Hence, by looking at the values of $M_{i-1,j-1}$, $M_{i,j-1}$ and $M_{i-1,j}$, one can determine which of the three entries that lead to $M_{i,j}$. Using such recursive algorithm, it eventually returns to the entry $M_{1,1}$.

1. The first case represents that there is a match between β_i and α_j, because the entry $M_{i,j}$ is calculated using $M_{i-1,j-1}$, in other words, $max(Match, Delete, Insertion) = Match$. Therefore, entry β_i is added to $\beta\_alignment$ and entry α_j is added to $\alpha\_alignment$. Further, since $\alpha_j = \beta_i$, the entry also contributes to the *lcs*. The backtracking algorithm hence moves to the entry $M_{i-1,j-1}$ by letting $i = i - 1$ and $j = j - 1$.

2. The second case corresponds to insertion, since the entry $M_{i,j}$ is calculated via $M_{i,j-1}$, i.e. $max(Match, Delete, Insertion) = Insertion$. Therefore if the algorithm reaches this part of the *if* statement, a gap is added to $\beta\_alignment$, and α_j is added to $\alpha\_alignment$.

   The third case is symmetric to the second case, that it corresponds to Delete, $max(Match, Delete, Insertion) = Delete$ at $M_{i,j}$.

The table shows the score matrix M, and the circled entries show all the possible backtrack paths using the backtracking algorithm. Note that there are 3 different paths, it is because all 3 paths lead to an optimal alignment. Using dynamic programming, all 3 paths give the maximum score of $M_{m,n} = 7$, and also optimal at any entries $\bar{i} < m$ and $\bar{j} < n$.

							α					
		ϕ	a	c	t	a	g	t	c	t	a	c
	ϕ	0	(0)	0	0	0	0	0	0	0	0	0
	c	(0)	0	(**1**)	(1)	1	1	1	1	1	1	1
	g	(0)	0	(1)	(1)	1	2	2	2	2	2	2
	a	0	(**1**)	1	1	(**2**)	2	2	2	2	3	3
	c	0	1	(2)	(2)	(2)	2	2	3	3	3	3
β	g	0	1	2	2	2	(**3**)	3	3	3	3	3
	t	0	1	2	3	3	3	(**4**)	4	4	4	4
	c	0	1	2	3	3	4	4	(**5**)	5	5	5
	g	0	1	2	3	3	4	4	(5)	5	5	5
	a	0	1	2	3	4	4	4	(5)	5	6	6
	t	0	1	2	3	4	4	5	5	(**6**)	6	6
	a	0	1	2	3	4	4	5	5	6	(**7**)	(7)

Table 2.7: Score Matrix M

a	c	⊔	t	a	⊔	g	t	c	⊔	⊔	t	a	c
⊔	c	g	⊔	a	c	g	t	c	g	a	t	a	⊔

a	c	t	⊔	a	⊔	g	t	c	⊔	⊔	t	a	c
⊔	c	⊔	g	a	c	g	t	c	g	a	t	a	⊔

⊔	⊔	a	c	t	a	g	t	c	⊔	⊔	t	a	c
c	g	a	c	⊔	⊔	g	t	c	g	a	t	a	⊔

Table 2.8: 3 different alignments from score matrix M

The above table shows all 3 possible optimal alignments. However, the LCS γ is generally not the same for all the alignments.

For the first and second alignment,

$$\gamma_1 = cagtcta$$

for the third alignment,

$$\gamma_2 = acgtcta$$

2.3 Similarity Score

When applying the LCS algorithm to the high frequency financial data, instead of alphabets *agct*, trade information is compared. For instance, a trade X consists of time stamp, bid ask price, trade volume e.t.c, and the uniqueness of each trade is characterised by this information. However, as mentioned above, due to human errors or system errors, the same trade from different data vendors might have different values.

X_i	Time Stamp	Bid price	Ask price	Trade volume
Data Vendor A	1401135	\$1.3498	\$1.35050	1000
Data Vendor B	1401135	\$1.3499	\$1.3505	1000

Table 2.9: Trade data example

The time stamp 1401135 represents a trade occurs at time 14:01:13.5, which means at two o'clock in the afternoon, first minute, and 13.5 second.

As seen from the above table, although the trade X_i is unique, both data vendors provide different information. Instead of \$1.3505 at ask price from data vendor B, the system of data vendor A generates a different number of \$1.35050. Also, for the bid price, there could be server lag from data vendor B, and it produces \$1.3499 instead of \$1.3498 from data vendor A. However, this trade is unique, and should be treated as the same data point in the data sequences such that X_i from data vendor A equals X_i from data vendor B.

Therefore, a similarity score is introduced. This is a comparison method proposed by L. Zhao et al. [3]

First of all, we determine the relative importance of fields, and assign each field F_i a weight W_i. Time stamp is likely the most important field within a trade data X, so the weight $W_{timestamp}$ is the highest. Suppose the trade data X has fields $F_1, F_2, ..., F_n$, and the field weightings are $W_1, W_2, ..., W_n$ respectively, the sum of all weights, $\sum_{i=1}^{n} W_i = 1$.

The similarity score for field F_i is defined as

$$Sim_{F_i}(\alpha, \beta) = lcs_p(\alpha_{F_i}, \beta_{F_i})$$

where,

$$lcs_p(\alpha, \beta) = \frac{lcs(\alpha, \beta)}{max(|\alpha|, |\beta|)}$$

and $lcs(\alpha, \beta)$ is the length of the longest common subsequence γ of α and β.

Hence, the similarity score of a trade data X would be

$$\sum_{i=1}^{n} lcs_p(\alpha_{F_i}, \beta_{F_i}) \times W_i$$

By construction, the similarity score for any two sequences is between 0.0 and 1.0. Any two sequences are treated as the "same" if their similarity score exceeds a certain threshold, denoted as ρ.

2.3.1 Example

Using the example above, Table 2.2, the similarity score can be used to compare trade X from data vendor A and data vendor B. The weightings are as follow,

Fields	Weightings
Time Stamp	0.4
Bid Price	0.2
Ask Price	0.2
Trade Volume	0.2

Table 2.10: Field weightings example

The similarity scores for fields Time Stamp, Bid Price, Ask Price and Trade Volume are 1, $\frac{5}{6}$, $\frac{7}{8}$ and 1 respectively. Note that the characters "$" and "." also count towards the alphabet Σ. Hence the total similarity score for trade X is

$$0.4 \times 1 + 0.2 \times \frac{5}{6} + 0.2 \times \frac{7}{8} + 0.2 \times 1 = 0.94167$$

If the threshold σ is 80%, then trade X from both data vendors is considered as the same data point, since $0.94167 > \rho = 0.8$.

Chapter 3

Application to High Frequency Financial Data

In the field of bioinformatics, a sequence alignment is a way of arranging sequences of DNA or protein to identify regions of similarity. On the other hand, within high frequency finance, since data series from different data vendors are different, regions of similarity are identified in order to carry out statistical analysis.

Previously, the Needleman-Wunsch algorithm is introduced to solve a LCS problem on a DNA sequence. Now, a different score function is used, and by exploiting the structure of high frequency financial data series, more efficient algorithms can also be derived.

As mentioned in Chapter 1, pairs of trade data arrive in irregular spaced time. For instance,

Data Vendor A	$X_{1,\tau=1}$	$X_{2,\tau=2}$	$X_{3,\tau=5}$	$X_{4,\tau=6}$	$X_{5,\tau=9}$
Data Vendor B	$X_{1,\tau=2}$	$X_{2,\tau=3}$	$X_{3,\tau=5}$	$X_{4,\tau=8}$	$X_{5,\tau=10}$

$X_{i,\tau=\hat{\tau}}$ represents the i^{th} trade occurs at time $\hat{\tau}$, where τ could be in millisecond or even microsecond. Suppose τ is measured in microsecond, hence the above example is a snapshot of a 10 microsecond interval, such that 5 trades arrive from Data Vendor A and 5 trades arrive from Data Vendor B. However, such irregular spaced data might cause problems when feeding into the forecasting model. Therefore, empty set ϕs are introduced into the gaps.

Suppose the data is sampled every microsecond from both data vendors, and let t denotes the sampling time. At $t = 0$, no trades stream in, therefore a ϕ is added to data sequences from both data vendor since nothing happened at $t = 0$. At the second sample, i.e. when $t = 1$, a trade $X_{1,\tau=1}$ streams in from Data Vendor A, but no trade streams in from Data Vendor B, so at $t = 1$, $X_{1,\tau=1}$ is added to the data sequence of Vendor A and a ϕ is added to the data sequence of Vendor B. Now the data sequences become,

t =	0	1	$\cdots$
Data Vendor A	ϕ	$X_{1,\tau=1}$	$\cdots$
Data Vendor B	ϕ	ϕ	$\cdots$

The same procedure continues until t reaches 10, or all the trades are sampled.

$t =$	0	1	2	3	4	5	6	7	8	9	10
Data Vendor A	ϕ	X_1	X_2	ϕ	ϕ	X_3	X_4	ϕ	ϕ	X_5	ϕ
Data Vendor B	ϕ	ϕ	X_1	X_2	ϕ	X_3	ϕ	ϕ	X_4	ϕ	X_5

Table 3.1: Data arrival time example

In previous chapter, the score matrix S is introduced which determine the matching score between $a, c, g, t, "", \sqcup$. However, each of the "letters" consists of trade data containing several fields, so that the alphabets contains 10^{20} to 10^{100} different letters. It is impossible to precompute the score matrix, however for any two trade data entries α and β, $Score(\alpha, \beta)$ can be evaluated using Zhao et al.'s approach,

$$Sim_F(\alpha, \beta) = \sum_{i=1}^{n} lcs_p(\alpha_{F_i}, \beta_{F_i}) \times W_i$$

where α_{F_i} is the i^{th} field of α and similarly for β. For instance, F_1 could be the bid volume and F_2 could be the bid price e.t.c.

Further, the threshold $\rho = 0.8$ is defined to be the threshold of the similarity score. If the score is greater than 0.8, then the pair of trade is categorised as the same trade, otherwise it is categorised as different trades, or outliers. Hence,

$$Score(\alpha, \beta) = \begin{cases} K & \text{if } Sim_F(\alpha, \beta) = \sum_{i=1}^{n} lcs_p(\alpha_{F_i}, \beta_{F_i}) \times W_i > 0.8 \\ 0 & \text{Otherwise} \end{cases}$$

Further, if a ϕ from A matches a ϕ from B,

$$Score(\phi, \phi) = k$$

Where $k << K$, this is because a match in trade data X_i contains much more relevant information than a match in ϕ. Hence, in the LCS-like algorithm, and higher score $K > k$ is assigned for a match in trade data, so that the level of significant of such match is higher than matching ϕs. As a result, the similarity matrix would be of the form,

	X_i	X_j	$\sqcup$	ϕ
X_i	K	0	0	0
X_j	0	K	0	0
$\sqcup$	0	0	0	0
ϕ	0	0	0	k

Table 3.2: Similarity matrix for high frequency financial data alignment

Where $Sim(X_i, X_j) < \rho = 0.8$, $Score(X_i, X_j) = 0$, and $Sim(X_i, X_i) \geq \rho = 0.8$, $Score(X_i, X_i) = K$.

3.1 Reduction of Score Matrix

If the Needleman-Wunsch Algorithm is used to align sequences with length m and n, the score matrix requires a memory of $O(mn)$ and computational time $O(mn)$ [4]. The efficiency of this algorithm becomes problematic when applying to high frequency data with length up to 10^6. Further, the modified algorithm is required to run in an online fashion, that when two sets of data stream in from data vendors, the algorithm can perform data cleaning and feed a set of cleaned data into a forecasting model, then through a trade and risk quantifier and finally into market execution. In the context of high frequency trading, such cycle are required to be done within seconds or even fraction of a second.

As mentioned at the beginning of this chapter, an empty set symbol ϕ is inserted into the data stream if no trade occurs at sample time t. During less volatile times, waiting time for trades might be in an order of a millisecond or a second. Suppose

a trade from $X_{1,\tau=1}$ from Data Vendor A arrives at $\tau = 1$, that τ is measured in microsecond; then the second trade $X_{2,\tau=1000000}$ arrives 1 second after the first trade, hence the waiting time is 1 second, which is equal to 10^6 microsecond, and 10^6 number of ϕs are added in between X_1 and X_2 since no trade occurs during the waiting time of 1 second. And suppose the first trade $X_{1,\tau=2}$ from Data Vendor B arrives at $\tau = 2$, and the second trade arrives 1 microsecond before it arrives at Data Vendor A, i.e. $X_{2,\tau=999999}$, then the data sequences will have the form,

t =	1	2	3	$\cdots$	999998	999999	1000000
Data Vendor A	X_1	ϕ	ϕ	$\cdots$	ϕ	ϕ	X_2
Data Vendor B	ϕ	X_1	ϕ	$\cdots$	ϕ	X_2	ϕ

Between $t = 3$ and $t = 999997$, both data sequences from Data Vendor A and Data Vendor B are filled with ϕs.

If one directly input this pair of data into the Needelman-Wunsch Algorithm, the score matrix M will require a memory and computational time of $O(10^{12})$. Therefore, this paper proposes a modified algorithm for computing a score matrix $\hat{M}$ and subsequently a modified backtracking algorithm.

The score matrix M certainly contains all the information regarding the optimal alignment of sequences A and B. However, not all entries in the matrix M are crucial; i.e. there are entries in M such that they can be deduced from or discarded when applying the backtracking algorithm. When computing the optimal alignment, the most important information is contained in entries with trade data; for instance, in the above example that only two trades occur within 1 second, and the first trade is at $t = 1$ and the second trade is at $t = 1000000$, the ϕs between the trades in the data sequence are seen to be useless in determining the optimal alignment. This is because, a match in ϕ from A and ϕ from B is assigned a score of k, and such match can be determined without further calculation; on the other hand, if one needs to determine whether the pair of trade data matches, the similarity score $Sim_F(A, B)$ must be calculated first before the score K or 0 is assigned.

The modified algorithm exploits such data structure, and removes the ϕ entries between trades. For simplicity, suppose the two data sequences given by each data

vendor are as follow,

t =	0	1	2	3	4	5	6	7	8	9	10
Data Vendor A	ϕ	X_1	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ	X_2
Data Vendor B	ϕ	ϕ	X_1	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ	X_2	ϕ

Table 3.3: Data sequence example

Then the score matrix M has the form,

			A										
	t =		0	1	2	3	4	5	6	7	8	9	10
			ϕ	X_1	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ	X_2
	0	ϕ	0	0	0	0	0	0	0	0	0	0	0
	1	ϕ	0	0	1	1	1	1	1	1	1	1	1
	2	X_1	0	(10)	10	10	10	10	10	10	10	10	(10)
	3	ϕ	0	10	11	11	11	11	11	11	11	11	11
B	4	ϕ	0	10	11	12	12	12	12	12	12	12	12
	5	ϕ	0	10	11	12	13	13	13	13	13	13	13
	6	ϕ	0	10	11	12	13	14	14	14	14	14	14
	7	ϕ	0	10	11	12	13	14	15	15	15	15	15
	8	ϕ	0	10	11	12	13	14	15	16	16	16	16
	9	X_2	0	(10)	11	12	13	14	15	16	16	16	(26)
	10	ϕ	0	10	11	12	13	14	15	16	17	17	26

Table 3.4: Pivot points for matrix M

Where $K = 10$ if trade data is matched, and $k = 1$ if ϕ is matched, and 0 otherwise for simplicity.

The crucial entries are called the "pivot points", which are the entries corresponding to a trade in both data series. As seen from the above matrix, there are 4 "pivot points" in total, this is because there are 2 trades from Data Vendor A, and 2 trades from Data Vendor B.

Because of the nature of the Needleman-Wunsch Algorithm, that entry $M_{i,j}$ is calculated using $M_{i-1,j-1}$, $M_{i-1,j}$ and $M_{i,j-1}$, the value of "pivot points" can be deduced from entries $M_{\bar{i},\bar{j}}$ given the sequence A and B and the value of other "pivot points",

where $\bar{i} < i$ and $\bar{j} < j$. However, for entries $M_{\grave{i},\grave{j}}$ such that $\grave{i} > i$ and $\grave{j} > j$, one can not deduce the value $M_{\grave{i},\grave{j}}$ just using $M_{i,j}$.

For example, the "pivot points" of the above matrix are $M_{2,1}$, $M_{2,10}$, $M_{9,1}$ and $M_{9,10}$. Given these 4 points, with values 10, 10, 10 and 20, other entries in the matrix $M_{i,j}$ can be deduced, but only for $2 \leq i \leq 9$ and $1 \leq j \leq 10$. In this case, the 0^{th} row, 1^{st} row, 2^{nd} row, 10^{th} row and the $0th$ column are impossible to be deduced by just using the 4 given "pivot points". Hence, more "pivot points" are required such that all possible entries of M could be deduced.

	t =		A										
			0	1	2	3	4	5	6	7	8	9	10
			ϕ	X_1	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ	X_2
	0	ϕ	(0)	(0)	0	0	0	0	0	0	0	0	(0)
	1	ϕ	0	0	1	1	1	1	1	1	1	1	1
	2	X_1	(0)	(10)	10	10	10	10	10	10	10	10	(10)
	3	ϕ	0	10	11	11	11	11	11	11	11	11	11
B	4	ϕ	0	10	11	12	12	12	12	12	12	12	12
	5	ϕ	0	10	11	12	13	13	13	13	13	13	13
	6	ϕ	0	10	11	12	13	14	14	14	14	14	14
	7	ϕ	0	10	11	12	13	14	15	15	15	15	15
	8	ϕ	0	10	11	12	13	14	15	16	16	16	16
	9	X_2	(0)	(10)	11	12	13	14	15	16	16	16	(26)
	10	ϕ	(0)	(10)	11	12	13	14	15	16	17	17	(26)

Table 3.5: Revised pivot points for matrix M

Now, there are 12 "pivot points" in total. Extra "pivot points" are added to the 0^{th} row, 0^{th} column and the 10^{th} row such that the entries in the 0^{th} row, 1^{st} row, 2^{nd} row, 10^{th} row and the $0th$ column can also be deduced using the "pivot points".

For any general score matrix M, the "pivot points" are identified by entries that correspond to a trade data in both A and B, and also by the first row/column and last row/column if the last entry of A or B is a ϕ.

Given all the necessary "pivot points", a reduced matrix $\hat{M}$ can be constructed by just using those "pivot points". In this case, since there are only 12 "pivot points",

the size of the reduced matrix $\hat{M}$ is 4 by 3. Compare to M, the memory usage of $\hat{M}$ is only of order $O(4 \times 3)$, where M required order $O(11 \times 11)$. The follow table shows the reduced matrix $\hat{M}$.

		α		
		$(\phi, 0)$	$(X_1, 1)$	$(X_2, 10)$
	$(\phi, 0)$	0	0	0
β	$(X_1, 2)$	0	10	10
	$(X_2, 9)$	0	10	26
	$(\phi, 10)$	0	10	26

Table 3.6: Reduced matrix $\hat{M}$

Where,

$$\alpha = (\phi, 0)(X_1, 1)(X_2, 10)$$

$$\beta = (\phi, 0)(X_1, 2)(X_2, 9)(\phi, 10)$$

And (X_i, τ_1) denotes the i^{th} trade at time τ_1, (ϕ, τ_2) denotes a ϕ, or no trade occurs at time τ_2.

The algorithm to compute the reduced score matrix $\hat{M}$ is described in the following section.

3.1.1 Modified Needleman-Wunsch Algorithm for Reduced Matrix

The following pseudo code shows the algorithm for computing $\hat{M}$[1],

1. **Initialisation**

   $n \leftarrow length(A)$
   $m \leftarrow length(B)$
   $\alpha \leftarrow (A$ without ϕ entries$)$
   $\beta \leftarrow (B$ without ϕ entries$)$
   if at time $t \geq 1$, the time of last entry of α is **not** $n - 1$ **then**
   Add ϕ and $t = n - 1$ to the end of α
   end if

[1]See Appendix for Matlab implementation

if at time $t \geq 1$, the time of last entry of β is **not** $m-1$ **then**
Add ϕ and $t = m-1$ to the end of β
end if
$\hat{n} \leftarrow length(\alpha)$
$\hat{m} \leftarrow length(\beta)$
$\hat{M} \leftarrow$ zero matrix of size $(\hat{m}, \hat{n})$

If the last column or row of reduced matrix $\hat{M}$ does not correspond to the last column or row of matrix M, information is lost and more "pivot points" are introduced into the matrix; therefore if the last entry of α or β is not the same as the last entry of A and B, i.e. last entry of A or B is ϕ, then this entry is added at the end of α and β so that the computed reduced matrix $\hat{M}$ can reproduce all the information from M.

Further, the similarity matrix S is included, and initialise the matrix $\hat{M}$
$st \leftarrow Score(X_\cdot, X_\cdot)$
$sp \leftarrow Score(\phi, \phi)$

st is the similarity score for the same trade X, such that $Score(X_\cdot, X_\cdot) = 10$, and $Score(\phi, \phi) = 1$ for sp. Further, $t_{\alpha(\hat{i})}$ is defined as the time of the i^{th} entry in α; in other words, if $\alpha(\hat{i})$ corresponds to a trade from Data Vendor A at time $t = \tau$, then $t_{\alpha(\hat{i})} = \tau$

2. **Computing entries**
for $\hat{i} \rightarrow \hat{m}$ **do**
for $\hat{j} \rightarrow \hat{n}$ **do**
if $\alpha(\hat{j}) = \beta(\hat{i})$ **and** $\alpha(\hat{j}) \neq \phi$ **and** $\beta(\hat{i}) \neq \phi$) **then**
$d \leftarrow 1$
else
$d \leftarrow 0$
end if
$delta \leftarrow min(t_{\alpha(\hat{i})} - t_{\alpha(\hat{i}-1)}, t_{\beta(\hat{j})} - t_{\beta(\hat{j}-1)})$
if $\alpha(\hat{j})$ equals ϕ **and** $\beta(\hat{i})$ equals ϕ **then**
$Match \leftarrow \hat{M}(\hat{i}-1, \hat{j}-1) + sp \times delta$
$Insertion \leftarrow \hat{M}(\hat{i}, \hat{j}-1) + sp \times delta$
$Delete \leftarrow \hat{M}(\hat{i}-1, \hat{j}) + sp \times delta$
$\hat{M}(\hat{i}, \hat{j}) \leftarrow max(Match, Delete, Insertion)$

else if $\alpha(\hat{j})$ equals ϕ **or** $\beta(\hat{i})$ equals ϕ **then**

$Match \leftarrow \hat{M}(\hat{i}-1, \hat{j}-1) + sp \times (delta - 1)$

$Insertion \leftarrow \hat{M}(\hat{i}, \hat{j}-1) + sp \times (delta - 1)$

$Delete \leftarrow \hat{M}(\hat{i}-1, \hat{j}) + sp \times (delta - 1)$

$\hat{M}(\hat{i}, \hat{j}) \leftarrow max(Match, Delete, Insertion)$

else

$Match \leftarrow \hat{M}(\hat{i}-1, \hat{j}-1) + d \times st + sp \times (delta - 1)$

$Insertion \leftarrow \hat{M}(\hat{i}, \hat{j}-1) + sp \times (delta - 1)$

$Delete \leftarrow \hat{M}(\hat{i}-1, \hat{j}) + sp \times (delta - 1)$

$\hat{M}(\hat{i}, \hat{j}) \leftarrow max(Match, Delete, Insertion)$

end if

end for

end for

The variable d is defined to be a *flag*, such that it is true if the pair of trades from α and β is categorised as the same, and otherwise the *flag* is false.

delta is known as the "jump" in the reduced score matrix $\hat{M}$. Consider the entry $M_{1,2}$ and $M_{9,10}$ in M (Table 3.5), which corresponds to entries $\hat{M}_{2,2}$ and $\hat{M}_{3,3}$, there is a "jump" in scores from 10 to 26, and the magnitude of the "jump" is determined by the structure of α and β. A "jump" of the score could be caused by a match or trade data, or a match of ϕ, and *delta* is defined as the number of jumps between the "pivot points".

Suppose $\alpha j = \beta i$ so that the *if* statement is true, then there is one "jump" of a score of 10 since $Score(\beta i, \alpha j) = 10$, and there are $delta - 1$ jumps of score 1. And suppose the *if* statement is false, then all *delta* number of jumps is of score 1, because all these jumps are caused by matches of ϕ, and 0 match of trade data. Further, suppose $t_{\alpha(\hat{i})} - t_{\alpha(\hat{i}-1)} > t_{\beta(\hat{j})} - t_{\beta(\hat{j}-1)}$, then the maximum number of matches, no matter a match in trade data or matches in ϕ, can only be $t_{\beta(\hat{j})} - t_{\beta(\hat{j}-1)}$. Therefore, the number of matches, or "jumps", is equal to the minimum of $t_{\alpha(\hat{i})} - t_{\alpha(\hat{i}-1)}$ and $t_{\beta(\hat{j})} - t_{\beta(\hat{j}-1)}$.

Further, suppose $\alpha(\hat{j}) = \phi$ and $\beta(\hat{i}) = \phi$, then there are *delta* number of jumps such that $Match$, $Insertion$ and $Delete$ equals to $delta \times sp$ plus $\hat{M}(\hat{i}-1, \hat{j}-1)$, $\hat{M}(\hat{i}, \hat{j}-1)$ and $\hat{M}(\hat{i}-1, \hat{j})$ respectively. However, if only either $\alpha(\hat{j}) = \phi$ or $\beta(\hat{i}) = \phi$, then this is a mismatch between $\alpha(\hat{j})$ and $\beta(\hat{i})$, so there is one less

jump than *delta*. Otherwise, there are $delta - 1$ jumps in ϕ and one jump in st since there is a match between $\alpha(\hat{j})$ and $\beta(\hat{i})$.

3.1.2 Modified Backtrack Algorithm for Reduced Matrix

Since the structure of the reduced score matrix $\hat{M}$ is different from M, a different backtracking algorithm is proposed. The following pseudo code outline the modified algorithm,

1. **Initialisation**

   The initialisation steps are almost the same as the one from computing $\hat{M}$, however the following pseudo code is added,

   $\hat{n} \leftarrow length(\alpha)$
   $\hat{m} \leftarrow length(\beta)$
   $\hat{i} \leftarrow \hat{m}$
   $\hat{j} \leftarrow \hat{n}$
   $i \leftarrow m - 1$
   $j \leftarrow n - 1$
   $\alpha\_alignment \leftarrow$ ""
   $\beta\_alignment \leftarrow$ ""
   $lcs \leftarrow$ ""

   (i, j) indicate the indices of the matrix M, while $(\hat{i}, \hat{j})$ indicate the indices of the reduced matrix $\hat{M}$. Although the actual score matrix M is not required for this algorithm, the indices of M are used as a proxy in order to compute the optimal alignment from the reduced matrix $\hat{M}$. $\alpha\_alignment$, $\beta\_alignment$ and lcs denote the output of this algorithm.

2. **Backtracking**
   The following algorithm starts at $\hat{i}$ and $\hat{j}$, and terminates when both $\hat{i}$ and $\hat{j}$ equals to 1

   if $\hat{i} \neq 1$ **and** ($\hat{j} = 1$ **or** ($\hat{M}(\hat{(i)}-1, \hat{j}-1) \neq \hat{M}(\hat{i}, \hat{j})$ **and** $\hat{M}(\hat{i}-1, \hat{j}) \neq \hat{M}(\hat{i}, \hat{j})$)) **then**
      $k \leftarrow t_{\beta(\hat{i})} - t_{\beta(\hat{i}-1)}$
      $\beta\_alignment \leftarrow \beta\_alignment + \beta(\hat{i})$

$\alpha\_alignment \leftarrow \alpha\_alignment + \text{" } \sqcup \text{"}$
$i \leftarrow i - k$
for $l = 1 \rightarrow k - 1$ **do**
$\beta\_alignment \leftarrow \beta\_alignment + \phi$
$\alpha\_alignment \leftarrow \alpha\_alignment + \text{" } \sqcup \text{"}$
end for
$\hat{i} \leftarrow \hat{i} - 1$
end if

The condition for the *if* statement is very similar to the one used in the original backtracking algorithm, which states that if the entry $M_{i,j}$ is calculated via *Delete*, then the algorithm proceeds to this section. Conditions $\hat{i} =!1$ and $\hat{j} =$ are used to ensure that the algorithm also performs in the desire way when it hits the boundary values. Further, $\hat{M}(\hat{i}-1, \hat{j}-1) \neq \hat{M}(\hat{i}, \hat{j})$ shows that the entry $M_{i,j}$ does not come from a *Match* at $\hat{i}$ and $\hat{j}$.

The variable k represents the number of ϕs between $\beta(\hat{i})$ and $\beta(\hat{i}-1)$. This is because the *if* statement is fulfilled, and implies that there are no match in either trade data or ϕs between $\hat{M}_{\hat{i},\hat{j}}$ and $\hat{M}_{\hat{i}-1,\hat{j}}$, therefore all the ϕs between $\beta(\hat{i})$ and $\beta(\hat{i}-1)$ corresponds to a gap $\sqcup$ in the $\alpha\_alignment$. Hence a *for loop* is used to create such alignments of $\alpha\_alignment$ and $\beta\_alignment$.

if $\hat{j} \neq 1$ **and** $(\hat{i} = 1$ **or** $(\hat{M}(\hat{(i)}-1, \hat{j}-1) \neq \hat{M}(\hat{i}, \hat{j})$ **and** $\hat{M}(\hat{i}, \hat{j}-1) \neq \hat{M}(\hat{i}, \hat{j})))$
then
$k \leftarrow t_{\alpha(\hat{j})} - t_{\alpha(\hat{j}-1)}$
$\alpha\_alignment \leftarrow \alpha\_alignment + \alpha(\hat{j})$
$\beta\_alignment \leftarrow \beta\_alignment + \text{" } \sqcup \text{"}$
$j \leftarrow j - k$
for $l = 1 \rightarrow k - 1$ **do**
$\alpha\_alignment \leftarrow \alpha\_alignment + \phi$
$\beta\_alignment \leftarrow \beta\_alignment + \text{" } \sqcup \text{"}$
end for
$\hat{j} \leftarrow \hat{j} - 1$
end if

This section of the code is symmetric to the one above, with index $\hat{i}$, $\hat{j}$, i and j switched.

If both *if* statements are not satisfied, then the algorithm proceeds to the next section.

if $\alpha(\hat{j}) \neq \phi$ **and** $\beta(\hat{i}) \neq \phi$ **and** $\alpha(\hat{j}) = \beta(\hat{i})$ **then**
  $\beta\_alignment \leftarrow \beta\_alignment + \beta(\hat{i})$
  $\alpha\_alignment \leftarrow \alpha\_alignment + \alpha(\hat{j})$
  $lcs \leftarrow lcs + \alpha(\hat{j})$
  $\hat{i} \leftarrow \hat{i} - 1$
  $\hat{j} \leftarrow \hat{j} - 1$
  $i \leftarrow i - 1$
  $j \leftarrow j - 1$
  $N \leftarrow \frac{\hat{M}(\hat{i},\hat{j}) - \hat{M}(\hat{i}-1,\hat{j}-1) - st}{sp}$
end if

The above *if* statement is true if $\alpha(\hat{j}) \neq \phi$ and $\beta(\hat{i}) \neq \phi$, and trade data $\alpha(\hat{j})$ matches trade data $\beta(\hat{i})$.

Since the pair of trade data is matched, it contributes to the LCS and hence lcs gets the entry $\beta(\hat{i})$. Further, after adding entries to $\alpha\_alignment$ and $\beta\_alignment$, the proxy indices (i, j) move back diagonally, and hence $i = i - 1$ and $j = j - 1$.

The variable N represents the number of "jumps" in ϕ. Using the structure of the reduced score matrix $\hat{M}$, the number of "jumps" can be calculated by the difference between $M_{i,j}$ and $M_{i-1,j-1}$ minus the jump of score st, because the jump st is already considered at the beginning of the *if* statement.

if $\alpha(\hat{j}) \neq \beta(\hat{i})$ **then**
  $k \leftarrow t_{\beta(\hat{i})} - t_{\alpha(\hat{j})}$
  if $k > 0$ **then**
    $\beta\_alignment \leftarrow \beta\_alignment + \beta(\hat{i})$
    $\alpha\_alignment \leftarrow \alpha\_alignment +$ " $\sqcup$ "
    $i \leftarrow i - 1$

for $l = 1 \rightarrow (k-1)$ **do**

$\beta\_alignment \leftarrow \beta\_alignment + \phi$

$\alpha\_alignment \leftarrow \alpha\_alignment + \phi$

$i \leftarrow i - 1$

$j \leftarrow j - 1$

end for

$\beta\_alignment \leftarrow \beta\_alignment +$ " $\sqcup$ "

$\alpha\_alignment \leftarrow \alpha\_alignment + \alpha(\hat{j})$

$i = i - 1$

$j - j - 1$

else if $k < 0$ **then**

$\beta\_alignment \leftarrow \beta\_alignment +$ " $\sqcup$ "

$\alpha\_alignment \leftarrow \alpha\_alignment + \alpha(\hat{j})$

$j \leftarrow j - 1$

for $l = 1 \rightarrow (k-1)$ **do**

$\beta\_alignment \leftarrow \beta\_alignment + \phi$

$\alpha\_alignment \leftarrow \alpha\_alignment + \phi$

$i \leftarrow i - 1$

$j \leftarrow j - 1$

end for

$\beta\_alignment \leftarrow \beta\_alignment + \beta(\hat{i})$

$\alpha\_alignment \leftarrow \alpha\_alignment +$ " $\sqcup$ "

$i = i - 1$

$j - j - 1$

end if

$N \leftarrow \frac{\hat{M}(\hat{i},\hat{j}) - \hat{M}(\hat{i}-1,\hat{j}-1)}{sp}$

end if

Clearly, the *if* statement at the top of this part of the algorithm indicates that $\alpha(\hat{j}) \neq \beta(\hat{i})$, and $\alpha(\hat{j})$ and $\beta(\hat{i})$ could be a trade data or ϕ.

The variable k, again, shows the number of ϕs that is present between $\alpha(\hat{j})$ and $\beta(\hat{j})$. Suppose $k > 0$, as shown in the second *if* statement, means that $t_{\beta(\hat{i})} > t_{\alpha(\hat{j})}$, i.e. the arrival time of $\beta(\hat{i})$ is later than $\alpha(\hat{j})$. Since $\alpha(\hat{j})$ and $\beta(\hat{i})$ is not matched, and $t_{\beta(\hat{i})} > t_{\alpha(\hat{j})}$, $\beta(\hat{i})$ is added to $\beta\_alignment$ while a gap $\sqcup$ is added to $\alpha\_alignment$, and $i = i - 1$. Further, k number of *phi*s are added to both alignments because there are k matches in ϕ between $\alpha(\hat{j})$ and $\beta(\hat{i})$, hence

$i = i - 1$ and $j = j - 1$. Finally, $\sqcup$ is added to $\beta\_alignment$ and $\alpha(\hat{j})$ is added to $\alpha\_alignment$, and $j = j - 1$.

The other case, where $k < 0$ is identical to the case $k > 0$, except $t_{\beta(\hat{i})} < t_{\alpha(\hat{j})}$, i.e. the arrival time of $\beta(\hat{i})$ is earlier than $\alpha(\hat{j})$, therefore $\alpha(\hat{j})$ is added to its alignment first rather than $\beta(\hat{i})$.

After considering these two cases, the variable N is defined as the difference between the score $M_{i,j}$ and $M_{i-1,j-1}$, which indicates the number of "jumps" in ϕ. In contrast to the previous section, where $N = \frac{\hat{M}(\hat{i},\hat{j}) - \hat{M}(\hat{i}-1,\hat{j}-1) - st}{sp}$, there is no jump in st because $\alpha(\hat{j})$ does not match $\beta(\hat{i})$, so that all "jumps" are caused by matching in ϕs.

The last case that this part of the algorithm needs to consider is when both $\alpha(\hat{j})$ and $\beta(\hat{i})$ equals a ϕ.

if $\alpha(\hat{j}) = \phi$ **and** $\beta(\hat{i}) = \phi$ **then**
  $N \leftarrow min(\hat{M}(\hat{i},\hat{j}) - \hat{M}(\hat{i}-1,\hat{j}), \hat{M}(\hat{i},\hat{j}) - \hat{M}(\hat{i},\hat{j}-1))$
end if

In this case, since both entries equal to a ϕ, all "jumps" are again caused by a ϕ, hence N is simply defined by the above formula. Further, because both entries are ϕ, no special treatment is required on the alignments, and the match in the ϕs will be accommodated in later section of the algorithm.

if $N \neq 0$ **then**
  for $l = 1 \rightarrow N$ **do**
    $\alpha\_alignment \leftarrow \alpha\_alignment + \phi$
    $\beta\_alignment \leftarrow \beta\_alignment + \phi$
    $i \leftarrow i - 1$
    $j \leftarrow j - 1$
  end for
end if
$\hat{i} \leftarrow \hat{i} - 1$

$\hat{j} \leftarrow \hat{j} - 1$

In previous parts of the algorithm, the variable N is calculated in different cases. Therefore, the number of ϕs in both alignments are determined by the structure of α and β

if $\alpha(\hat{j}) \neq \beta(\hat{i})$ **then**
  $ki \leftarrow i - t_{\beta(\hat{i})}$
  $kj \leftarrow j - t_{\alpha(\hat{j})}$
  if $kj > 0$ **then**
    for $l = 1 \rightarrow kj$ **do**
      $\alpha\_alignment \leftarrow \alpha\_alignment + \phi$
      $\beta\_alignment \leftarrow \beta\_alignment +$ " $\sqcup$ "
      $j \leftarrow j - 1$
    end for
  end if
  if $ki > 0$ **then**
    for $l = 1 \rightarrow ki$ **do**
      $\alpha\_alignment \leftarrow \alpha\_alignment +$ " $\sqcup$ "
      $\beta\_alignment \leftarrow \beta\_alignment + \phi$
      $i \leftarrow i - 1$
    end for
  end if
end if

The variables ki and kj are defined to be the difference between $t_{\alpha(\hat{j})} - t_{\alpha(\hat{j}-1)}$ and $t_{\beta(\hat{i})} - t_{\beta(\hat{i}-1)}$, and such expression can be simplied to $i - t_{\beta(\hat{i})}$ and $j - t_{\alpha(\hat{j})}$ within the algorithm. Suppose $(t_{\alpha(\hat{j})} - t_{\alpha(\hat{j}-1)}) > t_{\beta(\hat{i})} - t_{\beta(\hat{i}-1)}$, then $ki = 0$ and $kj > 0$; therefore kj number of ϕs are added to the $\alpha\_alignment$ while same number of gaps " $\sqcup$ " are added to the $\beta\_alignment$. On the other hand, if $(t_{\alpha(\hat{j})} - t_{\alpha(\hat{j}-1)}) < t_{\beta(\hat{i})} - t_{\beta(\hat{i}-1)}$, then $kj = 0$ and $ki > 0$, and ki number of ϕs and " $\sqcup$ "s are added to $\beta\_alignment$ and $\alpha\_alignment$.

The whole process is continued, with i, j, $ihat$ and $jhat$ decreasing until $ihat = 1$ and $jhat = 1$.

This completes the modified backtracking algorithm.

Note that both the modified backtracking algorithm and score matrix algorithm are more complex than the original Needleman-Wunsch Algorithm and its backtracking algorithm. Although memory usage for the score matrix is hugely improved, the improvement of time performance in computing both the score matrix and backtracking algorithm might not be too significant. The next section will explore the performance of both algorithms in relation to data length and the number of ϕs in between trade data.

3.1.3 Time Performance

Let

$$A = \phi X_1 \phi\phi X_2 \phi\phi\phi\phi X_3 \phi\phi\phi$$

$$B = \phi X_1 \phi\phi X_2 \phi\phi\phi X_3 \phi\phi\phi\phi$$

Computing the matrix M would require $O(13 \times 13)$ operations, however if both sequences are reduced to α and β, the computational time is only in $O(4 \times 4)$.

In the context of high frequency trading, the faster the algorithm, the more it is applicable, therefore speed plays a crucial role in high frequency finance.

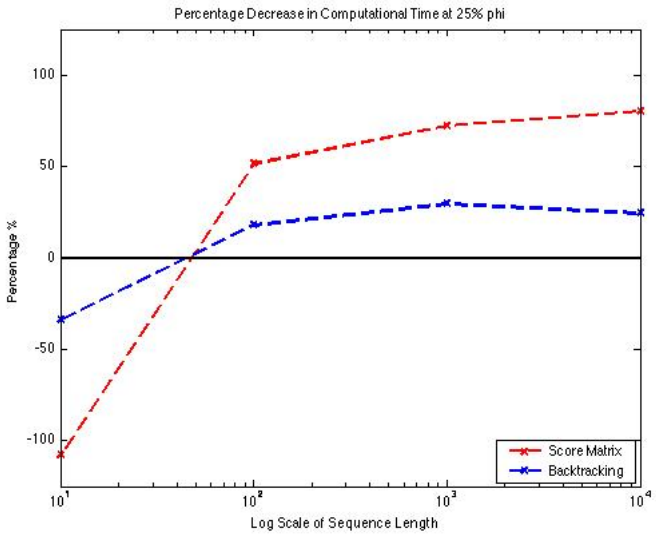

Figure 3.1: Time performance of algorithms with 37% ϕs

As seen from Figure 3.1, the percentage decrease in computational time of score matrix increases as the length of the data sequence increase. Although the algorithm for computing reduced score matrix $\hat{M}$ is more complex than the original matrix M, with around 25% of ϕ in the sequence, the performance improvement out-weights the extra computation required for length longer than 10^2. Further, the backtracking algorithm records similar results, where computation time decreases at around 25% for length greater than 10^2.

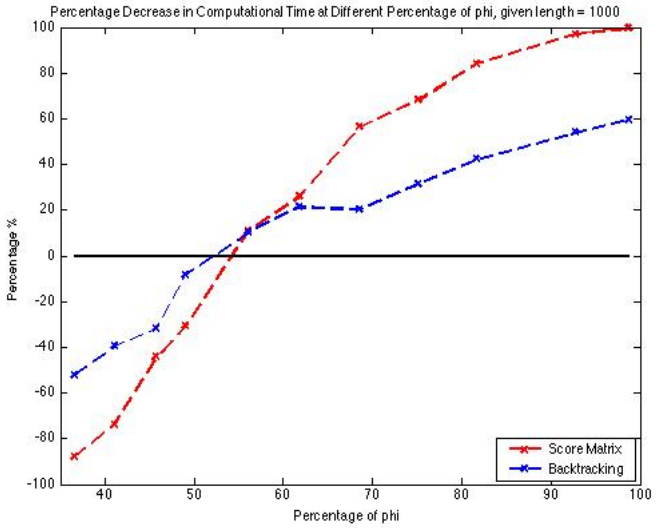

Figure 3.2: Time performance of algorithms using different percentage of ϕs

The figure shows that as the number of ϕ increases in the sequence, both the score matrix algorithm and backtracking algorithm significant reduce the required computational time. However, at around 55% of ϕs in the sequence, the performance is actually worse than the original algorithm, where the percentage decrease is negative, and suggests that one should use the ordinary score matrix and backtracking algorithm on sequences with more than 55% of ϕs.

These graphs help to determine the target data length and number of ϕs that the modified algorithm should apply. As seen from both figures, one should apply the modified algorithms to data sequences such that its length is larger than 10^2, and the

percentage of ϕs should be less than 55%.

In later chapters, it will be shown that real data from data vendors will fulfil both requirements and both modified algorithms for score matrix and backtracking are suitable for data cleaning.

3.2 Time Window

3.2.1 Reduction Matrix

Let's consider another example,

$t =$	0	1	2	3	4	5	6	7	8	9	10
Data Vendor A	ϕ	X_1	X_2	ϕ	ϕ	X_3	X_4	ϕ	ϕ	X_5	ϕ
Data Vendor B	ϕ	ϕ	X_1	X_2	ϕ	X_3	ϕ	ϕ	ϕ	X_4	X_5

Table 3.7: Data arrival time example 2

From the table, trade X_4 came in at $t = 6$ and $t = 9$ from data vendor A and data vendor B respectively. Such trade could be classified as a noise because its arrival time/information is inconsistent to a certain significant level.

Let $t_{A,i}$ be arrival time of trade X_i from data vendor A and $t_{B,i}$ from data vendor B, the trade X_i is *not synchronised* if $|t_{A,i} - t_{B,i}| > \omega$ for some ω. ω is called the *window* of the algorithm, and the value of ω varies for different kinds of data and level of confidence.

From Table 3.4 previously, $t_{A,4} = 6$ and $t_{B,4} = 9$. For example, setting $\omega = 2$, then $|t_{A,4} - t_{B,4}| > \omega$, and hence trade X_4 from data vendor A and data vendor B is not seen as the same data point. The new optimal alignment would be,

$\sqcup$	X_1	X_2	ϕ	ϕ	X_3	X_4	$\sqcup$	ϕ	ϕ	ϕ	X_5	ϕ
ϕ	X_1	X_2	$\sqcup$	ϕ	X_3	$\sqcup$	ϕ	ϕ	ϕ	X_4	X_5	$\sqcup$

Table 3.8: Optimal alignment of trade data example 2

And therefore the LCS is $\gamma = X_1X_2\phi X_3\phi\phi X_5$.

The following modified score matrix shows the N-W algorithm with window ω, and the computational time is hugely improved. For exapmle, setting $\omega = 2$, only entries corresponds to $|t_{A,i} - t_{B,j}| \leq \omega$ are calculated $\forall i, j$, and other entries are set to $-\infty$.

	$t =$						A						
			0	1	2	3	4	5	6	7	8	9	10
			ϕ	X_1	X_2	ϕ	ϕ	X_3	X_4	ϕ	ϕ	X_5	ϕ
	0	ϕ	0	0	0	$-\infty$	$\cdots$						
	1	ϕ	0	0	0	1	$-\infty$	$\cdots$					
	2	X_1	0	10	10	10	10	$-\infty$	$\cdots$				
	3	X_2	$-\infty$	10	20	20	20	20	$-\infty$	$\cdots$			
B	4	ϕ	$-\infty$	$-\infty$	20	21	21	21	21	$-\infty$	$\cdots$		
	5	X_3		$\cdots$	$-\infty$	21	31	31	31	31	$-\infty$	$\cdots$	
	6	ϕ			$\cdots$	$-\infty$	22	31	31	32	32	$-\infty$	$-\infty$
	7	ϕ				$\cdots$	$-\infty$	32	32	32	33	33	$-\infty$
	8	ϕ					$\cdots$	$-\infty$	32	33	33	33	34
	9	X_4						$\cdots$	$-\infty$	33	33	33	34
	0	X_5							$\cdots$	$-\infty$	33	33	43

Table 3.9: Time windowed score matrix

Seen from the above table, the entries that are required for computation are the diagonals; for the case $\omega = 2$, 5 diagonals are needed. Hence, for any $\omega > 0$, the required computational time is of $O(\omega \times n)$, which implies the complexity of the algorithm is linear in n. The larger ω is used, the higher is the computational time. The time windowed algorithm[2] is very similar to the one proposed by Hirschberg [5], except that further constraints are applied to boundary values.

Similar to previous section, one can derive a modified algorithm to take advantage of the data structure of high frequency finance data. One might think that the idea of "pivot points" is used again on the time window reduced matrix, however, the algorithm for computing a time window reduced matrix $\tilde{M}$ could be the same as computing a reduced matrix $\hat{M}$ such that it can conserve the time performance and efficiency shown in previous section.

[2]See appendix for Matlab code

Suppose the same trade X_1 arrives at $t = 4$ from Data Vendor A, and at $t = 10$ from Data Vendor B. And suppose $\omega = 2$, such that $|t_{A,1} - t_{B,1}| > \omega$, then the trade is treated as a mismatch; on the other hand, if $\omega = 7$, then $|t_{A,1} - t_{B,1}| = 10 - 4 = 6 \leq 7 = \omega$ and this pair of trade is treated as a match.

Using the same idea as to compute the reduced matrix $\hat{M}$, the time window reduced matrix $\tilde{M}$ is computed using the same algorithm, except if a pair of trade is out of the time window, it is treated as a mismatch.

For example, suppose these 2 data sequences arrive from the data vendors,

t =	0	1	2	3	4	5	6	7
Data Vendor A	ϕ	X_1	ϕ	ϕ	ϕ	ϕ	X_2	ϕ
Data Vendor B	ϕ	X_2	X_2	ϕ	ϕ	ϕ	ϕ	ϕ

Table 3.10: Data arrival example

Supose $\omega = 2$,clearly, the second trade X_2 is out of the time window; $t_{A,2} = 6$ while $t_{B,2} = 2$, such that $|t_{A,1} - t_{B,1}| > \omega$. The following tables show the difference between $\hat{M}$ and $\tilde{M}$,

		α			
		$(\phi, 0)$	$(X_1, 1)$	$(X_2, 6)$	$(\phi, 7)$
	$(\phi, 0)$	0	0	0	0
β	$(X_1, 1)$	0	10	10	10
	$(X_2, 2)$	0	10	20	20
	$(\phi, 7)$	0	10	20	21

Table 3.11: Matrix $\hat{M}$

Note that the difference in the score matrix is caused by X_2 being out of the time window ω. For matrix $\hat{M}$, X_2 is treated as a match hence there are two jumps in score of $st = 10$; while for matrix $\tilde{M}$, X_2 is treated as a mismatch hence there is only one jump of score $st = 10$.

		α			
		$(\phi, 0)$	$(X_1, 1)$	$(X_2, 6)$	$(\phi, 7)$
	$(\phi, 0)$	0	0	0	0
β	$(X_1, 1)$	0	10	10	10
	$(X_2, 2)$	0	10	10	10
	$(\phi, 7)$	0	10	14	15

Table 3.12: Matrix $\tilde{M}$

The algorithm for computing the time window reduced matrix $\tilde{M}$ is almost identical to the one for $\hat{M}$, except that for the pair of trades outside of the window ω, it is treated as a mismatch.

$n \leftarrow length(A)$
$m \leftarrow length(B)$
$\alpha \leftarrow (A$ without ϕ entries$)$
$\beta \leftarrow (B$ without ϕ entries$)$
if at time $t \geq 1$, the time of last entry of α is **not** $n-1$ **then**
    Add ϕ and $t = n-1$ to the end of α
end if
if at time $t \geq 1$, the time of last entry of β is **not** $m-1$ **then**
    Add ϕ and $t = m-1$ to the end of β
end if
$\tilde{n} \leftarrow length(\alpha)$
$\tilde{m} \leftarrow length(\beta)$
$\tilde{M} \leftarrow$ zero matrix of size $(\tilde{m}, \tilde{n})$
$st \leftarrow Score(X_\cdot, X_\cdot)$
$sp \leftarrow Score(\phi, \phi)$
for $\tilde{i} \rightarrow \tilde{m}$ **do**
    for $\tilde{j} \rightarrow \tilde{n}$ **do**
        if $\alpha(\tilde{j}) = \beta(\tilde{i})$ **and** $\alpha(\tilde{j}) \neq \phi$ **and** $\beta(\tilde{i}) \neq \phi)$ **and** $|t_{\alpha(\tilde{j})} - t_{\beta(\tilde{i})}| \leq \omega$ **then**
            $d \leftarrow 1$
        else
            $d \leftarrow 0$
        end if
        $delta \leftarrow min(t_{\alpha(\tilde{i})} - t_{\alpha(\tilde{i}-1)}, t_{\beta(\tilde{j})} - t_{\beta(\tilde{j}-1)})$
        if $\alpha(\tilde{j})$ equals ϕ **and** $\beta(\tilde{i})$ equals ϕ **then**

$Match \leftarrow \tilde{M}(\tilde{i}-1, \tilde{j}-1) + sp \times delta$
$Insertion \leftarrow \tilde{M}(\tilde{i}, \tilde{j}-1) + sp \times delta$
$Delete \leftarrow \tilde{M}(\tilde{i}-1, \tilde{j}) + sp \times delta$
$\tilde{M}(\tilde{i}, \tilde{j}) \leftarrow max(Match, Delete, Insertion)$
else if $\alpha(\tilde{j})$ equals ϕ **or** $\beta(\tilde{i})$ equals ϕ **then**
$Match \leftarrow \tilde{M}(\tilde{i}-1, \tilde{j}-1) + sp \times (delta - 1)$
$Insertion \leftarrow \tilde{M}(\tilde{i}, \tilde{j}-1) + sp \times (delta - 1)$
$Delete \leftarrow \tilde{M}(\tilde{i}-1, \tilde{j}) + sp \times (delta - 1)$
$\tilde{M}(\tilde{i}, \tilde{j}) \leftarrow max(Match, Delete, Insertion)$
else
$Match \leftarrow \tilde{M}(\tilde{i}-1, \tilde{j}-1) + d \times st + sp \times (delta - 1)$
$Insertion \leftarrow \tilde{M}(\tilde{i}, \tilde{j}-1) + sp \times (delta - 1)$
$Delete \leftarrow \tilde{M}(\tilde{i}-1, \tilde{j}) + sp \times (delta - 1)$
$\tilde{M}(\tilde{i}, \tilde{j}) \leftarrow max(Match, Delete, Insertion)$
end if
end for
end for

Clearly the only difference between the algorithm for computing $\hat{M}$ and $\tilde{M}$ is that the condition $|t_{\alpha(\tilde{j})} - t_{\beta(\tilde{i})}| \leq \omega$ is added to one of the *if* statement, that a pair of trade data only to be considered as a match if their similarity score $Sim(X_\cdot, X_\cdot$ is above ρ **and** this pair is within ω

By construction of the matrix, such that $\tilde{M}$ has the same structure as $\hat{M}$, one can use the same backtracking algorithm to compute the alignments, and again the condition $|t_{\alpha(\tilde{j})} - t_{\beta(\tilde{i})}| \leq \omega$ is added into the new backtracking algorithm.

1. **Initialisation**

$n \leftarrow length(A)$
$m \leftarrow length(B)$
$\alpha \leftarrow (A$ without ϕ entries$)$
$\beta \leftarrow (B$ without ϕ entries$)$
if at time $t \geq 1$, the time of last entry of α is **not** $n-1$ **then**
Add ϕ and $t = n-1$ to the end of α
end if
if at time $t \geq 1$, the time of last entry of β is **not** $m-1$ **then**

Add ϕ and $t = m - 1$ to the end of β

end if

$\tilde{n} \leftarrow length(\alpha)$

$\tilde{m} \leftarrow length(\beta)$

$\tilde{M} \leftarrow$ zero matrix of size $(\tilde{m}, \tilde{n})$

$st \leftarrow Score(X_., X_.)$

$sp \leftarrow Score(\phi, \phi)$

$\tilde{i} \leftarrow \tilde{m}$

$\tilde{j} \leftarrow \tilde{n}$

$i \leftarrow m - 1$

$j \leftarrow n - 1$

$\alpha\_alignment \leftarrow$ ""

$\beta\_alignment \leftarrow$ ""

$lcs \leftarrow$ ""

2. **Backtracking**

while $\tilde{i} > 1$ **or** $tildej > 1$ **do**

if $\tilde{i} \neq 1$ **and** ($\tilde{j} = 1$ **or** $(\tilde{M}(\tilde{(i)} - 1, \tilde{j} - 1) \neq \tilde{M}(\tilde{i}, \tilde{j})$ **and** $\tilde{M}(\tilde{i} - 1, \tilde{j}) \neq \tilde{M}(\tilde{i}, \tilde{j}))$) **then**

$k \leftarrow t_{\beta(\tilde{i})} - t_{\beta(\tilde{i}-1)}$

$\beta\_alignment \leftarrow \beta\_alignment + \beta(\tilde{i})$

$\alpha\_alignment \leftarrow \alpha\_alignment +$ " ⊔ "

$i \leftarrow i - k$

for $l = 1 \rightarrow k - 1$ **do**

$\beta\_alignment \leftarrow \beta\_alignment + \phi$

$\alpha\_alignment \leftarrow \alpha\_alignment +$ " ⊔ "

end for

$\tilde{i} \leftarrow \tilde{i} - 1$

else if $\tilde{j} \neq 1$ **and** ($\tilde{i} = 1$ **or** $(\tilde{M}(\tilde{(i)} - 1, \tilde{j} - 1) \neq \tilde{M}(\tilde{i}, \tilde{j})$ **and** $\tilde{M}(\tilde{i}, \tilde{j} - 1) \neq \tilde{M}(\tilde{i}, \tilde{j}))$) **then**

$k \leftarrow t_{\alpha(\tilde{j})} - t_{\alpha(\tilde{j}-1)}$

$\alpha\_alignment \leftarrow \alpha\_alignment + \alpha(\tilde{j})$

$\beta\_alignment \leftarrow \beta\_alignment +$ " ⊔ "

$j \leftarrow j - k$

for $l = 1 \rightarrow k - 1$ **do**

$\alpha\_alignment \leftarrow \alpha\_alignment + \phi$

$\beta\_alignment \leftarrow \beta\_alignment +$ " $\sqcup$ "

end for

$\tilde{j} \leftarrow \tilde{j} - 1$

else

if $\alpha(\tilde{j}) \neq \phi$ **and** $\beta(\tilde{i}) \neq \phi$ **and** $\alpha(\tilde{j}) = \beta(\tilde{i})$ **and** $|t_{\alpha(\tilde{j})} - t_{\beta(\tilde{i})}| \leq \omega$ **then**

$\beta\_alignment \leftarrow \beta\_alignment + \beta(\tilde{i})$

$\alpha\_alignment \leftarrow \alpha\_alignment + \alpha(\tilde{j})$

$lcs \leftarrow lcs + \alpha(\tilde{j})$

$\tilde{i} \leftarrow \tilde{i} - 1$

$\tilde{j} \leftarrow \tilde{j} - 1$

$i \leftarrow i - 1$

$j \leftarrow j - 1$

$N \leftarrow \frac{\tilde{M}(\tilde{i},\tilde{j}) - \tilde{M}(\tilde{i}-1,\tilde{j}-1) - st}{sp}$

else if $\alpha(\tilde{j}) \neq \beta(\tilde{i})$ —— $|t_{\alpha(\tilde{j})} - t_{\beta(\tilde{i})}| > \omega$ **then**

$k \leftarrow t_{\beta(\tilde{i})} - t_{\alpha(\tilde{j})}$

if $k > 0$ **then**

$\beta\_alignment \leftarrow \beta\_alignment + \beta(\tilde{i})$

$\alpha\_alignment \leftarrow \alpha\_alignment +$ " $\sqcup$ "

$i \leftarrow i - 1$

for $l = 1 \rightarrow (k-1)$ **do**

$\beta\_alignment \leftarrow \beta\_alignment + \phi$

$\alpha\_alignment \leftarrow \alpha\_alignment + \phi$

$i \leftarrow i - 1$

$j \leftarrow j - 1$

end for

$\beta\_alignment \leftarrow \beta\_alignment +$ " $\sqcup$ "

$\alpha\_alignment \leftarrow \alpha\_alignment + \alpha(\tilde{j})$

$i = i - 1$

$j - j - 1$

else if $k < 0$ **then**

$\beta\_alignment \leftarrow \beta\_alignment +$ " $\sqcup$ "

$\alpha\_alignment \leftarrow \alpha\_alignment + \alpha(\tilde{j})$

$j \leftarrow j - 1$

for $l = 1 \rightarrow (k-1)$ **do**

$\beta\_alignment \leftarrow \beta\_alignment + \phi$

$\alpha\_alignment \leftarrow \alpha\_alignment + \phi$
$i \leftarrow i - 1$
$j \leftarrow j - 1$
end for
$\beta\_alignment \leftarrow \beta\_alignment + \beta(\tilde{i})$
$\alpha\_alignment \leftarrow \alpha\_alignment +$ " $\sqcup$ "
$i = i - 1$
$j - j - 1$
end if
$N \leftarrow \frac{\tilde{M}(\tilde{i},\tilde{j}) - \tilde{M}(\tilde{i}-1,\tilde{j}-1)}{sp}$
else if $\alpha(\tilde{j}) = \phi$ **and** $\beta(\tilde{i}) = \phi$ **then**
$N \leftarrow min(\tilde{M}(\tilde{i},\tilde{j}) - \tilde{M}(\tilde{i}-1,\tilde{j}), \tilde{M}(\tilde{i},\tilde{j}) - \tilde{M}(\tilde{i},\tilde{j}-1)$
end if
if $N \neq 0$ **then**
for $l = 1 \rightarrow N$ **do**
$\alpha\_alignment \leftarrow \alpha\_alignment + \phi$
$\beta\_alignment \leftarrow \beta\_alignment + \phi$
$i \leftarrow i - 1$
$j \leftarrow j - 1$
end for
end if
$\tilde{i} \leftarrow \tilde{i} - 1$
$\tilde{j} \leftarrow \tilde{j} - 1$
if $\alpha(\tilde{j}) \neq \beta(\tilde{i})$ **or** $|t_{\alpha(\tilde{j})} - t_{\beta(\tilde{i})}| > \omega$ **then**
$ki \leftarrow i - t_{\beta(\tilde{i})}$
$kj \leftarrow j - t_{\alpha(\tilde{j})}$
if $kj > 0$ **then**
for $l = 1 \rightarrow kj$ **do**
$\alpha\_alignment \leftarrow \alpha\_alignment + \phi$
$\beta\_alignment \leftarrow \beta\_alignment +$ " $\sqcup$ "
$j \leftarrow j - 1$
end for
end if
if $ki > 0$ **then**
for $l = 1 \rightarrow ki$ **do**
$\alpha\_alignment \leftarrow \alpha\_alignment +$ " $\sqcup$ "

$\beta\_alignment \leftarrow \beta\_alignment + \phi$
$i \leftarrow i - 1$
end for
end if
end if
end if
end while

3.2.2 Parallel Computing

Other than using reduction matrix to compute optimal alignments with a time window ω, techniques in parallel computing could be employed by expoiting the struction of a time windowed score matrix.

				β				
		β_0	β_1	β_2	β_3	β_4	β_5	β_6	$\cdots$
	α_0	a_0	b_0						
	α_1	c_0	a_1	b_1					
α	α_2		c_1	a_2	b_2				
	α_3			c_2	a_3	b_3			
	α_4				c_3	a_4	b_4		
	α_5					c_4	a_5	b_5	
	α_6						c_5	$\ddots$	$\ddots$
	$\vdots$							$\ddots$	

Table 3.13: Score matrix for parallel computing

Suppose the time window is ω, then $2\omega + 1$ number of processors are required to compute the score matrix using such method. The above matrix shows how parallel computing could be used, where $\omega = 1$.

Since $\omega = 1$, 3 processors are required to compute this matrix in parallell. Let the processors be P_a, P_b and P_c, and all three of the processors need to communicate before computing the next entries of the matrix. This can also be implemented in Matlab using the package "Parallel Computing Toolbox".

The procedure of such algorithm is as follow,

1. **Initialisation**
   The entries a_0, b_0 and c_0 are set to 0, and processor P_a is responsible for elements a_i, P_b for b_i and P_c for c_i.

2. **Computing a_1**
   According to the original algorithm,

$$a_1 = max\left(a_0 + Score(\alpha_1, \beta_1), b_0 + Score(\sqcup, \beta_1), c_0 + Score(\sqcup, \alpha_1)\right)$$

   However, the processor P_a only scores entries a_i and has no information regarding b_i and c_i. Hence, before computing a_1, processors P_b and P_c need to send the value of b_0 and c_0 to processor P_a such that it has all the information required to calculute a_1

3. **Computing b_1 and c_1**

$$b_1 = max\left(a_1 + Score(\alpha_1, \beta_2), b_0 + Score(\sqcup, \beta_2)\right)$$
$$c_1 = max\left(a_1 + Score(\alpha_2, \beta_1), c_0 + Score(\sqcup, \alpha_2)\right)$$

   Both formula required the value a_1. Hence after a_1 is worked out by P_a, its value is immediately sent to both P_b and P_c, and these two processors can then calculate b_1 and c_1.

$$\begin{array}{ccccccc}
P_{b_0}\ (b_0) & \rightarrow & P_{b_1}\ (b_1) & \rightarrow & P_{b_2}\ (b_2) & \rightarrow & \cdots \\
 & \searrow & \uparrow & \searrow & \uparrow & \searrow & \cdots \\
P_{a_0}\ (a_0) & \rightarrow & P_{a_1}\ (a_1) & \rightarrow & P_{a_2}\ (a_2) & \rightarrow & \cdots \\
 & \nearrow & \downarrow & \nearrow & \downarrow & \nearrow & \cdots \\
P_{c_0}\ (c_0) & \rightarrow & P_{c_2}\ (c_1) & \rightarrow & P_{c_2}\ (c_2) & \rightarrow & \cdots
\end{array}$$

Table 3.14: Information flow between processors

The above diagram shows how entries of the score matrix are communicated between each processors, and this method can extend to a higher number of processors. However, the use of parallel computing is limited by the number of processors that is available to user. When dealing with a large set of high frequency data, where a unit of time t represents 1 millisecond or even 1 microsecond, and one might use a time window of $\omega = 100$. In such cases, 201 processors are required to complete this

algorithm, and a cluster of more than 201 CPUs is expensive to build. Therefore, GPUs are a better and more suitable resource to implement the method in this case.

Chapter 4

Testing with High Frequency Data

For the purpose of testing all the proposed algorithms, historical noisy high frequency financial data are used. The data format is as follow

Time Stamp	Bid Volume	Bid Price	Ask Price	Ask Volume
1311919207025085	4	7091	7092	1

Table 4.1: High frequency data format

The time stamp is in Unix Time format, where 1 unit of Unix Time in this data set represents a micro second ($10^{-6}s$), and it is defined as the number of micro second that have elapsed since 00:00:00 Coordinated Universal Time (UTC), 1 January 1970. Therefore, the time stamp 1311919207025085 is precisely at 29/07/2011 $06:00:07.025085$ GMT. For simplicity, the first trade is set at $t = 0$, and hence if a second trade occurs 400 micro second after the first trade, then $t = 400$ for the second trade.

Further, weights W_i are required to assign to all the fields for the high frequency financial data in order to compute the similarity score for each trade. Since historical data is used for testing, the time stamp represents the arrival time, and only fields bid price, ask price, trade and volume are considered.

The following table shows the corresponding weights

Fields F_i	Bid Volume	Bid Price	Ask Price	Volume
Weights W_i	0.2	0.3	0.3	0.2

Table 4.2: Weights for historical high frequency data

6 different sets of data are available for testing from two different futures market, Eurex[1] and CME[2]; DAX futures (FDAX.EX) and EuroStoxx futures (FEXS.EX) from Eurex, Henry Hub Natural Gas Futures (NG CME), E-mini S&P 500 Futures (ES CME), Light Sweet Crude Oil (WTI) Futures (CL CME) and EUR/USD Futures (6E CME) from Chicago Mercantile Exchange. Data from Eurex accounts for roughly 14 hours, while data from CME accounts for roughly 23 hours. The following table shows the average waiting time for a trade and average number of ϕs between a trade.

Symbol of Securities	Average Waiting Time (s)	Average Number of ϕs
FDAX	0.4523	4.5×10^5
FESX	2.2554	2.3×10^6
CL.F	1.036	1.0×10^6
6E.F	0.9138	9.1×10^5
ES.F	2.7814	2.8×10^6
NG.F	6.2068	6.2×10^6

Table 4.3: Historical high frequency data statistics

The statistics show that a reduced time windowed algorithm is best suited for the data. For a large average number of ϕs between each data, up to $O(10^6)$, computational time is hugely improved, which is very crucial in the context of high frequency trading.

However, other than concerning speed of computation, memory usage is also a huge issue. With large number of ϕs in between each trade, storing large data sets with total length of $O(60 \times 60 \times 10^6)$ for only one hour of data of one asset would become

[1]Eurex is one of the world's leading derivatives market located in Eschborn, Germany.

[2]CME, Chicago Mercantile Exchange, is a financial and commodities derivatives exchange in Chicago, USA.

problematic. Therefore, in order to compute the LCS and the optimal alignments efficiently and introduce this algorithm "online" such that as data streaming in, the algorithm will work out the alignments in real time, one might want to sample data every second or every millisecond. For instance, as data streaming in, the alignment is computed every millisecond, and the optimal alignment of 1 second of data would be produced by concatenating 1000 optimal alignments of each millisecond.

Using such method, the alignments obtained by concatenating smaller alignments might be locally optimal but not globally optimal. For instance, if the rolling window is set at 1000 microseconds, then the LCS and alignments are computed for every 1000 microseconds. Although one is certain that those alignments are optimal, the concatenated of these alignments might not be the same as directly computing alignments for every 1 second; such error in theory could be reduced by increasing the rolling window, but higher computational cost might be required.

In order to mimic two sets of raw data, some errors will be introduced to the original 6 data sets; some trades will be removed with probability of 0.25%, while errors of timestamps will follow a normal distribution, and such errors occur with probability 0.75%, i.e..

$U \leftarrow Uniform(0,1)$
if $U < 0.1$ **then**
  $\hat{U} \leftarrow Uniform(0,1)$
  if $\hat{U} < 0.75$ **then**
    $N \leftarrow Normal(0,1)$
    $TimeStamp_i \leftarrow TimeStamp_i + \lfloor e \times N \rfloor$
  else
    i^{th} entry is deleted
  end if
end if

The above algorithm shows that the i^{th} entry of the data set is perturbed with probability 0.1, and the variable e is defined as the error on arrival times, i.e. the time difference in arrival of the i^{th} trade data follows a Normal Distribution.

Using Zhao et al.'s idea of a rolling window, the following function is constructed

for data cleaning,

function ROLLINGWINDOW($x, xe, t, count, counte, rw$)
  $x\_ \leftarrow$ zero array of size $(rw, 5)$
  $xe\_ \leftarrow$ zero array of size $(rw, 5)$
  for $i = 1 \rightarrow rw$ **do**
    if $t_{x\_(count)} \leq t$ **then**
      $x\_(i) \leftarrow x(count)$
      $count \leftarrow count + 1$
    else
      $x\_(i) \leftarrow \phi$
    end if
    if $t_{xe\_(counte)} \leq t$ **then**
      $xe\_(i) \leftarrow xe(counte)$
      $counte \leftarrow counte + 1$
    else
      $xe\_(i) \leftarrow \phi$
    end if
    $t \leftarrow t + 1$
  end for
  return $(x\_, xe\_, t, count, counte)$
end function

The function **RollingWindow** is to provide a suitable data sequence for the time window modified algorithms, where rw is the size time rolling window, x is the original data, xe is the perturbed data, t is the time of the start of the rolling window, $count$ is the index of x and $counte$ is the index of xe.

Suppose rw, the timed rolling window, is 1000, which is 1000 microsecond, then the function scans the orignal data sequence from time t up to $t + rw$; if there is a trade arrives in the window at time $t + \delta$, where $\delta < rw$,$x\_(\delta)$ is filled with the trade data and similarly for $xe\_$, otherwise $x\_$ and $xe\_$ are filled with ϕ.

Further, another function is constructed for the actual data cleaning,

function CLEAN(x, xe, ω, s, rw)
  $s \leftarrow s \times 10^6$

$n \leftarrow \lfloor \frac{s}{rw} \rfloor$
$remainder \leftarrow smodrw$
$t \leftarrow min(t_{x(1)}, t_{xe(1)})$
$count \leftarrow 1$
$counte \leftarrow 1$
$lcs \leftarrow$ ""
$alignments \leftarrow$ ""
for $i = 1 \rightarrow n$ **do**
$(x\_, xe\_, t, count, counte) \leftarrow RollingWindow(x, xe, t, count, counte, rw)$
$\tilde{M} \leftarrow reducedNWwindow(x\_, xe\_, \omega)$
$(lcstemp, alignmentstemp) \leftarrow backtrack(\tilde{M}, x\_, xe\_, \omega)$
$alignments \leftarrow alignments + alignmentstemp$
$lcs \leftarrow lcs + lcstemp$
end for
if $remainder \neq 0$ **then**
$(x\_, xe\_, t, count, counte) \leftarrow RollingWindow(x, xe, t, count, counte, remainder)$
$\tilde{M} \leftarrow reducedNWwindow(x\_, xe\_, \omega)$
$(lcstemp, alignmentstemp) \leftarrow backtrack(\tilde{M}, x\_, xe\_, \omega)$
$alignments \leftarrow alignments + alignmentstemp$
$lcs \leftarrow lcs + lcstemp$
end if
return $(lcs, alignments)$
end function

The function **Clean** takes the arguments (x, xe, ω, s, rw), where x is the original data sequence, xe is the perturbed data sequence, ω is the time window in microseconds, s is the total time of data that required cleaning in seconds, and rw is the timed rolling window in microseconds.

First of all, the variable s is converted back to microseconds, n is defined to be the number of rolling windows required to clean s microseconds of data, and t is defined to be the starting time of the data sequences. Further, suppose $s \not\equiv 0 \bmod rw$, and $n = \lfloor \frac{s}{rw} \rfloor$, then the remainding entries are cleaned outside of the *for* loop.

Inside the *for* loop, for each i, data sequences with ϕs inserted by the function **RollingWindow**, and the function **backtrack** produces a lcs and alignments by taking the score matrix $\tilde{M}$, $x\_$, $xe\_$ and ω as arguments.

Finally, the *lcs* and *alignments* of s seconds of data are produced by concatenating n *lcstemp* and *alignmentstemp*.

As seen from the following figure, the total computational time for FDAX and NGF is different at every value of time rolling window rw, this is because trade data from FDAX arrives the most frequent, with 55 trades in the first 5 seconds, while trade data from NGF arrives the least frequent, with 0 trades.

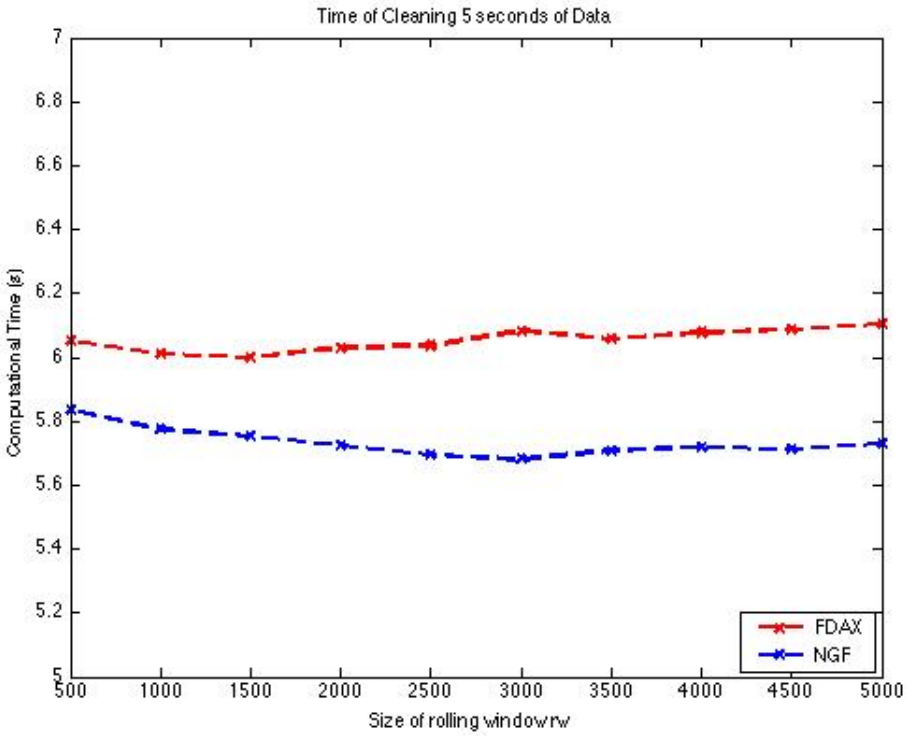

Figure 4.1: Total data cleaning time of 5 seconds of FDAX

Since this figure suggests that trade data arrival frequency affects the overall computational time, the following table shows the total computational time across all 6 securities for the first 60 seconds.

Securities	FDAX	ESF	CLF	FESX	6EF	NGF
Total Computational Time (s)	69.76	68.68	68.36	68.6	68.24	67.9
Number of Trades	280	86	45	43	30	1

Table 4.4: Total computational time for 60s of data

As the number of trade arrives decreases, the computational time also decreases. However, the difference is not significant, with only 2% more in computational time between FDAX and NGF but with around 53 times more trades.

In order to practically implement this cleaning algorithm, the computational time to clean data for τ seconds must be smaller than τ, such that there is extra time to feed the cleaned data to a forecast model and hence trade execution. Currently, the cleaning algorithm is tested under a sample time of 1 microsecond; in order to dramatically improve the computational time, the sampling time can be changed to 10 microsecond instead, then the timestamp will have to round up to order of 10^1.

Suppose now the sampling frequency is at $t = 10$ microsecond, for 60s of data, it only requires around 6.8 seconds to compute the LCS and alignment, with 53.2s remaining to forecast and execute trade.

Due to limited computational power, only a subset of the data is used to test the accuracy of the algorithms.

	True # of matches	$rw = 100$	$rw = 500$	$rw = 1000$
10s of data	78	72 (8.3% error)	77 (1.3% error)	78 (0% error)
30s of data	164	157 (4.3% error)	163 (0.6% error)	164 (0% error)
60s of data	261	251 (3.8% error)	260 (0.4% error)	261 (0% error)
120s of data	442	424 (4.1% error)	441 (0.2% error)	442 (0% error)

Table 4.5: Errors of algorithms using different rolling window

The above test is done on FDAX data, since it contains the most trade within any give time period.

Clearly, the amount of error decreases as the time rolling window increase. Suppose a trade X_1 arrives at $t = 100$ from Data Vendor A, and the same trade arrives at $t = 101$ from Data Vendor B, so that the data sequences look like the following,

t =	0	$\cdots$	100	101	$\cdots$	200	$\cdots$	300
Data Vendor A	ϕ	$\cdots$	X_1	ϕ	$\cdots$	ϕ	$\cdots$	ϕ
Data Vendor B	ϕ	$\cdots$	ϕ	X_1	$\cdots$	ϕ	$\cdots$	ϕ

Now, let $\omega = 2$ and $rw = 100$.

t =	0	$\cdots$	100	101	$\cdots$	200	$\cdots$	300
Data Vendor A	ϕ	$\cdots$	X_1	ϕ	$\cdots$	ϕ	$\cdots$	ϕ
Data Vendor B	ϕ	$\cdots$	ϕ	X_1	$\cdots$	ϕ	$\cdots$	ϕ

The vertical lines show how the data sequence of length 300 is separated into 3 different subsequence since $rw = 100$. Suppose there is no time rolling window, and theoretically, X_1 is matched; however, a rolling window of $rw = 100$ separated the sequence such that X_1 from Data Vendor A belongs to the first rolling window while X_1 belongs to the second rolling window from Data Vendor B. As a result, the LCS would be empty, and there is no match in the optimal alignments.

This problem can be solved by increasing the size of the rolling window; as demonstrated in the empirical test, the larger the rolling window, the more accurate the algorithms.

Chapter 5

Conclusion and Future Work

5.1 Conclusion

Previous work on LCS-like algorithms mainly focuses on bioinformatics, and research papers about LCS-like algorithms on finance are rarely seen. This paper built on the work of Needleman-Wunsch Algorithm and the concept of optimal alignments, and produced an algorithm that is suitable for high frequency financial data. This paper proceeds as follows,

1. Rather than using DNA and protein sequences a, g, c, t, the "alphabets" of high frequency financial data are introduced. Using the idea of similarity score of Zhao et al.'s paper, the similarity matrix is constructed accordingly for high frequency financial data.

2. By exploiting the sparse structure of data sequences, such that a large number of ϕs are inserted between trade data, much more efficient algorithms are proposed. The new algorithm for computing a reduced score matrix $\hat{M}$ hugely increase computational time; by referring to section 3.1.3 Time performance, a data sequence with around 99% of ϕs improve the computational time for score matrix by almost 100%. Subsequently, by taking advantages of the "pivot points" of the original matrix, a modified backtracking algorithm is also proposed.

3. Further, in order to practically apply the algorithm to real data, the concept of time window ω is used so that any pair of trades that is not "synchronised" is treated as mismatch. Due to server lags of exchanges, the same trade from a particular data vendor might arrive much later or earlier than another data

vendor. One should use a data cleaning algorithm to remove such outliers; hence this paper also proposed a time window modified Needleman-Wunsch Algorithm to accommodate these errors. A method using parallel computing is briefly introduced, however this method will require a large cluster of CPUs, or GPU computing. Due to limited computational resources, this method was not tested with performance in this paper

4. Moreover, based on the idea of a rolling window by Zhao et al., a timed rolling window is used to compute the LCS and alignments of real data. This method further improve the computational time and memory usage in storing data sequences with length more than 10^9. Computational time and accuracy are tested, and promising results are obtained with low or even zero errors.

5.2 Future Work

Much future work can be done based on this paper in order to practically implement in finance, particularly in high frequency trading firms and hedgefunds that rely on tick data.

5.2.1 Time Performance

As seen from empirical testing in last chapter, the algorithms performed accurately; however, in order to be implemented in an online fashion, one needs to further improve the efficiency of the algorithm. Since all of the algorithms are run in Matlab, there is a huge potential in computational time reduction by using low level programming languages, such as Java or C depends on industries needs. Further, an extension to the section Parallel Computing could also be worked on; high frequency trading firm or hedge funds typically have a large clusters of CPU that could be used for this method. The calculations within the algorithms are not complex, for instance, the entry $M_{i,j}$ equals the maximum of $Match$, $Insertion$ and $Delete$, and such simple calculations can also be implemented in GPUs using other programming languages.

5.2.2 Rolling Window

As explained also in previous chapter, the original data sequences will be separated into subsequences by the rolling window, and results in errors of matching which could be solved by increasing the size of rw; however, as the length of the data sequences tends to infinity, and such that the rolling window is finite, there always exists a

pair of trades which is separated by the rolling window. One could further modify the algorithms; for instance, suppose $rw = 100$ and $\omega = 2$, and the length of data sequences equals 300. In this paper the LCS and alignments are computed from 3 different subsequencs, where the first sequences equal entries from 1 to 100, second sequences equal entries from 101 to 200, and similarly third sequences equal 201 to 300. The rolling window could be modified such that the first sequences equal entries from 1 to 100, but the second sequences equal entries from 98 to 200, the third equals entries from 198 to 300. This method can eliminate the errors mentioned previously, however duplicates might appear in the LCS and alignments. In future, one can work on how to efficiently remove duplicates, and find the optimal rolling window to compensate errors using computational time in removing duplicates.

5.2.3 Similarity Matrix

The similarity matrix proposed in this paper is arbitrary;

	X_i	X_j	$\sqcup$	ϕ
X_i	K	0	0	0
X_j	0	K	0	0
$\sqcup$	0	0	0	0
ϕ	0	0	0	k

Table 5.1: Similarity matrix for high frequency financial data alignment

Where K is set to be 10, and k is set to be 1. Other form of similarity matrix might perform better than this one; for instance, K could be set as the similarity score, $Sim(X_i, X_j) \times 10$, however, algorithms for reduced score matrix $\hat{M}$ and $\tilde{M}$ would be far more complicated. Moreoever, one can use techniques in machine learning, such that given two sets of raw data and one set of clean data, the method of machine learning can compute an optimal similarity matrix based on historical data.

.1 Needleman-Wunsch Algorithm

```
1
2  function [M] = NW(x,y)
3
4  tic
5
6  stt = 10;
7  spp = 1;
8  sgap = 0;
9
10
11 m = length(x);
12 n = length(y);
13
14 x = [0 x];
15 y = [0 y];
16
17 M(m+1,n+1) = 0;
18
19 for i = 2:(m+1)
20     for j = 2:(n+1)
21         if x(i) == y(j)
22             d = stt;
23         elseif (isnan(x(i)) && isnan(y(j)))
24             d = spp;
25         else d = sgap;
26         end
27         match = M(i-1,j-1) + d;
28         delete = M(i-1,j) + sgap;
29         insertion = M(i,j-1) + sgap;
30         M(i,j) = max(delete, max(match,insertion));
31     end
32 end
33
34 toc
35
36 end
```

.2 Backtrack Algorithm

```
function[lcs,x_, y_] = backtrack(M,x,y)

tic

stt = 10;
spp = 1;
sgap = 0;

x = [0 x];
y = [0 y];

[m,n] = size(M);

i = m;
j = n;

x_ = '0';
y_ = '0';
lcs = '0';

while i > 1 || j > 1

    if x(i) == y(j)
        d = stt;
    elseif (isnan(x(i)) && isnan(y(j)))
        d = spp;
    else d = sgap;
    end
    if i > 1 && j > 1 && d~=0 && M(i-1,j-1) == M(i,j) - d
        x_ = [num2str(x(i)) x_];
        y_ = [num2str(y(j)) y_];
        if ~isnan(x(i))
            lcs = [num2str(x(i)) lcs];
        end

        j = j - 1;
        i = i - 1;
    elseif j > 1 && M(i,j) == M(i,j-1) + sgap
        x_ = ['-' x_];
        y_ = [num2str(y(j)) y_];
        j = j - 1;
    elseif i > 1 && M(i,j) == M(i-1,j) + sgap
        x_ = [num2str(x(i)) x_];
        y_ = ['-' y_];
        i = i - 1;
    end
end

x_ = x_(1:(end-1));
```

```
y_ = y_(1:(end-1));
lcs = lcs(1:(end-1));

toc

end
```

.3 Similarity Score

```
function[score] = LCSscore(x,y)

x = num2str(x);
y = num2str(y);

x = [0 x];
y = [0 y];

m = length(x);
n = length(y);

M = zeros(m+1, n+1);

for i = 1:(m+1)
    M(i,1) = 0;
end

for j = 1:(n+1)
    M(1,j) = 0;
end

for i = 2:(m+1)
    for j =2:(n+1)
        if (x(i-1) == y(j-1))
            M(i,j) = M(i-1,j-1) + 1;
        else
            M(i,j) = max(M(i-1,j), M(i,j-1));
        end
    end
end

score = (M(m+1,n+1) - 1)/(max(m,n) - 1);
end
```

.4 Modified NW Algorthim Aindowed

```
function[M] = reducedNWwindowdata(x,y,w)

%tic

```

```
x = [0 0 0 0 0 ;x];
y = [0 0 0 0 0 ;y];

m = length(x);
n = length(y);

x = [(0:(m-1))' x];
y = [(0:(n-1))' y];

x = x(~isnan(x(:,2)),:);
y = y(~isnan(y(:,2)),:);

if x(end,1) ~= m-1
    x = [x; [m-1 nan nan nan nan nan ]];
end

if y(end,1) ~= n-1
    y = [y; [n-1 nan nan nan nan nan ]];
end

[mhat,temp] = size(x);
[nhat,temp] = size(y);

M(mhat,nhat) = 0;

stt = 10;
spp = 1;


for i = 2:mhat
    for j = 2:nhat

        if ~isnan(x(i,2)) && ~isnan(y(j,2))

             score = 0.2*LCSscore(x(i,2),y(j,2)) + ...
                 0.3*LCSscore(x(i,3),y(j,3)) ...
                + 0.3*LCSscore(x(i,3),y(j,3)) + ...
                0.2*LCSscore(x(i,5),y(j,5));
        else score = 0;
        end

    if score >= 0.8 && abs(x(i,1) - y(j,1)) <= w
        d = 1;
    else
        d = 0;
    end

    M(i,j) = max(max(M(i-1,j-1) + d*stt + spp*(min(x(i,1) - x(i-1,1),...
        y(j,1) - y(j-1,1)) - 1), M(i-1,j)),M(i,j-1));

    if isnan(x(i,2)) && isnan(y(j,2))
        a = spp*(min(x(i,1) - x(i-1,1),y(j,1) - y(j-1,1)));
        M(i,j) = max(max(M(i-1,j-1) + a,...
            M(i-1,j) + a),M(i,j-1) + a);
```

```
59
60      elseif isnan(x(i,2)) || isnan(y(j,2))
61          a = spp*(min(x(i,1) - x(i-1,1),y(j,1) - y(j-1,1))-1);
62          M(i,j) = max(max(M(i-1,j-1) + a,...
63              M(i-1,j) + a),M(i,j-1) + a);
64
65
66      end
67      end
68
69
70  end
71
72  %toc
73
74  end
```

.5 Modified Backtracking Algorithm

```
1   function[lcs,align] = reducedbacktrackwindowdata(M,x,y,w)
2
3   %tic
4
5   x = [0 0 0 0 0 ;x];
6   y = [0 0 0 0 0 ;y];
7
8   m = length(x);
9   n = length(y);
10
11  x = [(0:(m-1))' x];
12  y = [(0:(n-1))' y];
13
14  x = x(~isnan(x(:,2)),:);
15  y = y(~isnan(y(:,2)),:);
16
17  if x(end,1) ~= m-1
18      x = [x; [m-1 nan nan nan nan nan ]];
19  end
20
21  if y(end,1) ~= n-1
22      y = [y; [n-1 nan nan nan nan nan ]];
23  end
24
25  [mhat,temp] = size(x);
26  [nhat,temp] = size(y);
27
28  stt = 10;
29  spp = 1;
30
31
32  ihat = mhat;
```

```
33 jhat = nhat;
34
35 i = m - 1;
36 j = n - 1;
37
38 x_ = cell(m+n,5);
39 y_ = cell(m+n,5);
40 lcs = cell(max(mhat,nhat),5);
41 count = length(x_);
42 lcscount = length(lcs);
43
44 while ihat > 1 || jhat > 1
45
46     if ihat ~= 1 && (jhat == 1 || (M(ihat-1,jhat-1) ~= M(ihat,jhat)...
47             && M(ihat-1,jhat) == M(ihat,jhat)))
48         k = (x(ihat,1) - x(ihat-1,1));
49         x_(count,:) = num2cell(x(ihat,2:end));
50         y_(count,:) = {'_'};
51         count = count - 1;
52         i = i - k;
53         x_((count-(k-1)+1):count,:) = {nan};
54         y_((count-(k-1)+1):count,:) = {'_'};
55         count = count - (k-1);
56         ihat = ihat - 1;
57     elseif jhat ~= 1 && (ihat == 1 || (M(ihat-1,jhat-1) ~= M(ihat,jhat) ...
58             && M(ihat,jhat-1) == M(ihat,jhat)))
59         k = (y(jhat,1) - y(jhat-1,1));
60         y_(count,:) = num2cell(y(jhat,2:end));
61         x_(count,:) = {'_'};
62         count = count - 1;
63         j = j - k;
64         y_((count-(k-1)+1):count,:) = {nan};
65         x_((count-(k-1)+1):count,:) = {'_'};
66         count = count - (k-1);
67         jhat = jhat - 1;
68     else
69
70         if ~isnan(x(ihat,2)) && ~isnan(y(jhat,2))
71
72              score = 0.2*LCSscore(x(ihat,2),y(jhat,2)) + ...
73                  0.3*LCSscore(x(ihat,3),y(jhat,3)) ...
74                + 0.3*LCSscore(x(ihat,3),y(jhat,3)) + ...
75                  0.2*LCSscore(x(ihat,5),y(jhat,5));
76         else score = 0;
77         end
78
79
80         if (~isnan(x(ihat,2)) && ~isnan(y(jhat,2))) ...
81                 && score >= 0.8 && abs(x(ihat,1) - y(jhat,1)) <= w
82             x_(count,:) = num2cell(x(ihat,2:end));
83             y_(count,:) = num2cell(y(jhat,2:end));
84             lcs(lcscount,:) = num2cell(x(ihat,2:end));
85             count = count - 1;
86             lcscount = lcscount - 1;
```

```
87              i = i - 1;
88              j = j - 1;
89              numofphi = (M(ihat,jhat) - M(ihat-1,jhat-1) - stt)/spp;
90          elseif ~(isnan(x(ihat,2)) && isnan(y(jhat,2))) ...
91                  && (score < 0.8 || abs(x(ihat,1) - y(jhat,1)) > w)
92
93              k = x(ihat,1) - y(jhat,1);
94              if k > 0
95                  x_(count,:) = num2cell(x(ihat,2:end));
96                  y_(count,:) = {'_'};
97                  count = count - 1;
98                  i = i - 1;
99                  x_((count-(k-1)+1):count,:) = {nan};
100                 y_((count-(k-1)+1):count,:) = {'_'};
101                 count = count - (k-1);
102                 i = i - (k-1);
103                 j = j - (k-1);
104                 x_(count,:) = {'_'};
105                 y_(count,:) = num2cell(y(jhat,2:end));
106                 count = count - 1;
107                 i = i - 1;
108                 j = j - 1;
109             else
110                 k = -k;
111                 y_(count,:) = num2cell(y(jhat,2:end));
112                 x_(count,:) = {'_'};
113                 count = count - 1;
114                 j = j - 1;
115                 x_((count-(k-1)+1):count,:) = {nan};
116                 y_((count-(k-1)+1):count,:) = {'_'};
117                 count = count - (k-1);
118                 i = i - (k-1);
119                 j = j - (k-1);
120                 y_(count,:) = {'_'};
121                 x_(count,:) = num2cell(x(ihat,2:end));
122                 count = count - 1;
123                 i = i - 1;
124                 j = j - 1;
125             end
126             numofphi = (M(ihat,jhat) - M(ihat-1,jhat-1))/spp;
127         else
128             numofphi =  min(M(ihat,jhat) - M(ihat-1,jhat),...
129                 M(ihat,jhat) - M(ihat,jhat-1));
130         end
131
132         ihat = ihat - 1;
133         jhat = jhat - 1;
134
135         if numofphi ~= 0
136
137             y_((count-numofphi+1):count,:) = {nan};
138             x_((count-numofphi+1):count,:) = {nan};
139             count = count - numofphi;
140             i = i - numofphi;
```

```
            j = j - numofphi;
        end

        if ~(~(isnan(x(ihat,2)) && isnan(y(jhat,2))) && ...
                (score < 0.9 || abs(x(ihat,1) - y(jhat,1)) > w))

            ki = max(i - x(ihat,1),0);
            kj = max(j - y(jhat,1),0);
            if kj ~= 0
                x_((count-kj+1):count,:) = {'-'};
                y_((count-kj+1):count,:) = {nan};
                j = j - kj;
                count = count - kj;
            end
            if ki ~= 0
                y_((count-ki+1):count,:) = {'-'};
                x_((count-ki+1):count,:) = {nan};
                i = i - ki;
                count = count - ki;
            end
        end
    end
end


x_ = x_((count+1):end,:);
y_ = y_((count+1):end,:);
lcs = lcs((lcscount+1):end,:);

align = [x_ y_];

%toc

end
```

.6 Rolling Window

```
function[x_,y_,t,count,counte] = rollingwindow(x,y,t,count,counte,rw)


x_ = zeros(rw,5);
y_ = zeros(rw,5);

for i = 1:rw

    if x(count,1) <= t
        x_(i,:) = x(count,:);
        count = count + 1;
    else x_(i,:) = nan;
    end

```

```
    if y(counte,1) <= t
        y_(i,:) = y(counte,:);
        counte = counte + 1;
    else y_(i,:) = nan;
    end

    t = t + 1;

end
end
```

.7 Data Cleaning

```
function[lcs] = clean(x,y,s,rw)

s = s*10^6;

lcs = cell(1,5);

n = floor(s/rw);

r = rem(s,rw);

t = min(x(1,1),y(1,1));
count = 1;
counte = 1;

w = 100;

tic

for i = 1:n


    [x_,y_,t,count,counte] = rollingwindow(x,y,t,count,counte,rw);

    if any(~isnan(x_(:,1))) && any(~isnan(y_(:,1)))

        [lcs1,align] = reducedbacktrackwindowdata(...
            reducedNWwindowdata(x_,y_,w),x_,y_,w);
        if ~isempty(lcs1)
            lcs = [lcs;lcs1];
        end
    end

end

if r ~= 0

    [x_,y_,t,count,counte] = rollingwindow(x,y,t,count,counte,r);

```

```
39          if (~any(isnan(x_(:,1))) && (~any(isnan(y_(:,1)))))
40
41              [lcs1,align] = reducedbacktrackwindowdata(...
42                  reducedNWwindowdata(x_,y_,w),x_,y_,w);
43              if ~isempty(lcs1)
44                  lcs = [lcs;lcs1];
45              end
46          end
47  end
48
49
50  toc
51
52  lcs = lcs(2:end,:);
53  end
```

Bibliography

[1] Nathaniel Popper *High-Speed Trading No Longer Hurtling Forward, NewYork Times* Oct. 2012

[2] Michel M. Dacorogna et al. *An Introduction to High-Frequency Finance* 2001

[3] L. Zhao et al. *A New Efficient Data Cleansing Method* 2002

[4] Saul B. Needleman and Christian D. Wunsch. *A General Method Applicable to the Search for Similarities in the Amino Acid Sequence of Two Proteins* 1970

[5] D.S. Hirschberg *A Linear Space Algorithm for Computing Maximal Common Subsequence* 1975