FAKULTÄT FÜR !NFORMATIK

Event-Based Similarity Search and its Applications in Business Analytics

MASTERARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Software Engineering & Internet Computing

eingereicht von

Martin Suntinger
Matrikelnummer 0405478

an der

Fakultät für Informatik der Technischen Universität Wien

Betreuung:
Betreuer/Betreuerin: Univ.-Prof. Dipl.-Ing. Dr.techn. Günther Raidl

Wien, 23.03.2009

(Unterschrift Verfasser/in) (Unterschrift Betreuer/in)

Abstract

Event-based systems enable real-time monitoring of business incidents and automated decision making to react on threats or seize time-critical business opportunities. Applications thereof are manifold, ranging from logistics, fraud detection and recommender systems to automated trading. Business incidents reflect in sequences of events. Understanding these sequences is crucial for designing accurate decision rules. At the same time, analysis tools for event data are still in their infancy.

The on-hand thesis presents a comprehensive and generic model for similarity search in event data. It illuminates several application domains to derive requirements for fuzzy retrieval of event sequences. Similarity assessment starts at the level of data fields encapsulated in single events. In addition, occurrence times of events, their order, missing events and redundant events are considered. In a graphical editor, the analyst models search-constraints and refines the pattern sequence. The model aims at utmost flexibility and configurability which is achieved by pattern modeling, configurable similarity techniques with different semantics and adjustable weights for similarity features.

The algorithm computes the similarity between two event sequences based on assigning events in the target sequence to events in the pattern sequence with respect to given search constraints. The deviations in the best possible assignment make up the final similarity score. This assignment is discovered by applying an efficient Branch-&-Bound algorithm. In addition, a novel way for time-series similarity is introduced and integrated. It slices a time-series at decisive turning points of the curve and compares the slopes between these turning points.

We surveyed applicability in real-world scenarios in four case studies. Results are promising for structured business processes of limited length. When choosing appropriate weights and configuration parameters to focus the search on aspects of interest, it is able to reveal if a reference case is a reoccurring pattern in the data.

Table of contents

1 Introduction

1.1 Technological background

Event-based systems and particularly the concept of Complex Event Processing (CEP) [29] have been developed and used to control business processes with loosely coupled systems. CEP enables monitoring, steering and optimizing business processes with minimal latency. It facilitates automated, near real-time closed-loop decision making at an operational level to discover exceptional situations or business opportunities. Typical application areas are financial market analysis, trading, security, fraud detection, customer relationship management, logistics like tracking shipments and compliance checks.

In an event-based system, any notable state change in the business environment is captured in the form of an event. Events are data capsules holding data about the context of the state change in so called event attributes. Chains of semantically or temporally correlated events reflect complete business processes, sequences of customer interactions or any other sequence of related incidents.

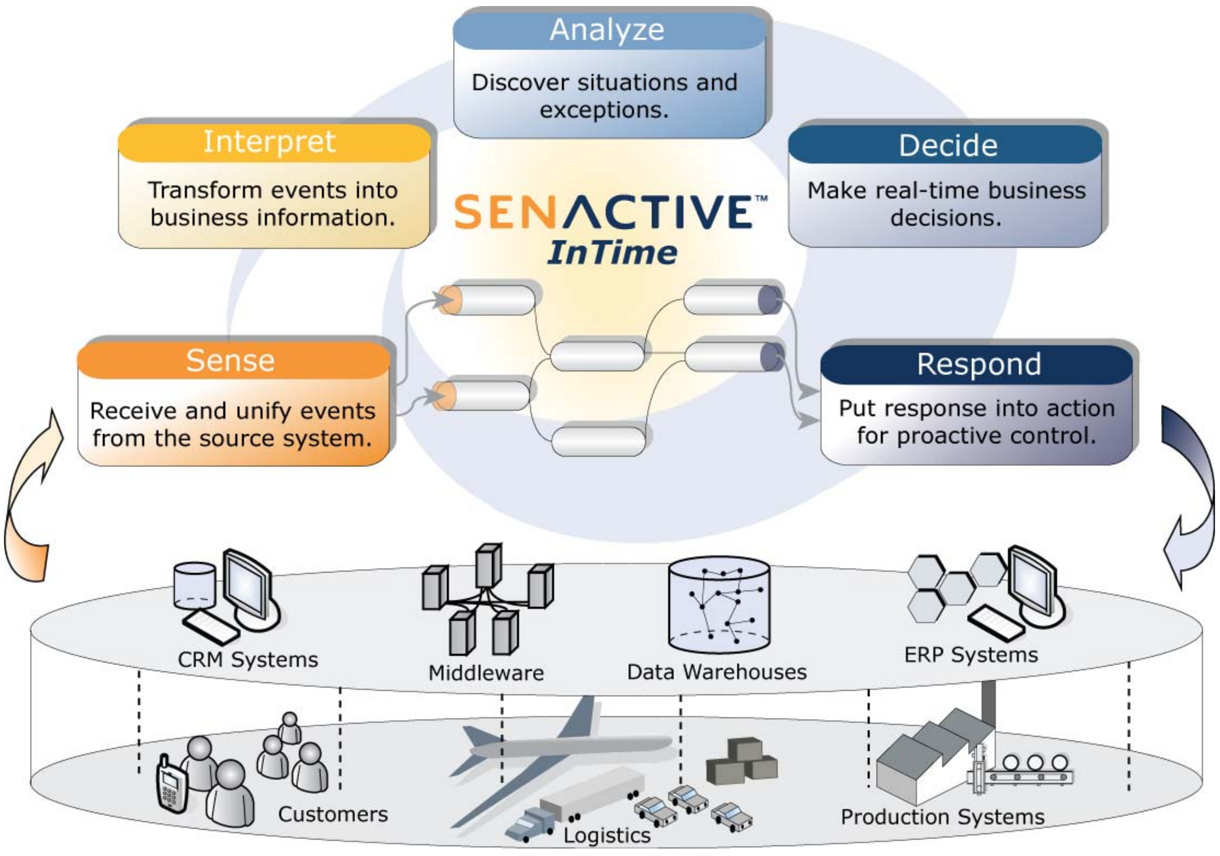

Analyze
Discover situations and exceptions.
Interpret
Transform events into business information.
Decide
Make real-time business decisions.
SENACTIVE™ InTime
Sense
Receive and unify events from the source system.
Respond
Put response into action for proactive control.
CRM Systems
Middleware
Data Warehouses
ERP Systems
Customers
Logistics
Production Systems

Figure 1: Sense and respond model[1]

Figure 1 illustrates the closed-loop decision processes employed by CEP software. One common conceptual (business) model is the so-called sense and respond model. Hereby, each cycle consists of 5 steps. In the "sense" step adapters capture input data from the IT landscape of an enterprise (which is a reflection of the physical business world). Interpretation refers to understanding, transforming, preparing and enriching the

[1] Figure by courtesy of SENACTIVE Inc.

data. This step is followed by an analysis step which tries to illuminate the given situation and context. Finally, a decision can be made and carried out by responding to the business environment. Typically a system of configurable rules is used for the decision process.

In addition to the real-time processing, during the past years one requirement has clearly emerged: The success of event-driven business solutions depends on an ongoing learning process. It is an iterative cycle including the analysis and interpretation of past processing results and the conversion of them into the event-processing logic. Analysis tools are required which are tailored to the characteristics of event data to answer questions like: Where did irregularities occur in my business? Did processes change over time? Which patterns can be recognized in my business? To answer these questions, the analyst has to be equipped with a whole range of supporting tools such as extensive retrieval facilities to extract required data sets. Expressive visualizations are necessary to navigate through event data and recognize recurring patterns and irregularities that influence the business performance.

For the analysis of historical event data, but also for the operational system, one question is of particular interest: Having an event sequence on hand, which other sequences are similar to this sequence? For data analysis, answering this question helps for searching the historic data for incidents and event patterns similar to a known reference pattern. In the operational system, the discovery of similarities can be integrated into the decision processes for automated system decisions to react in near real-time to certain event patterns. In addition, it can be used for forecasting of events or process measures based on similar historic incidents.

The on-hand mechanisms for searching similar event sequences have been designed and developed for being integrated into the SENACTIVE product suite. SENACTIVE Inc.[2] offers its customers a generic complex event processing engine with various graphical modeling facilities for designing the event processing flow. In addition, analysis software (the SENACTIVE *EventAnalyzer*^{TM}) provides facilities for analyzing historic event data. Despite this fact, the proposed mechanisms and algorithms can be applied in any other event-based system environment as well, as the data representation we rely on conforms to common CEP structures.

One major characteristic immanent to CEP is its claim of being generic. This means in particular the possibility to apply it in different application domains. In fact, some of nowadays applications for CEP solutions have not even been considered at all when CEP first emerged. This could be experienced for the real-time event processing but also for the analysis solutions. With the diversity of applications comes also a great diversity in the data sets. This reaches from the types of events occurring over the length and structure of correlating event sequences to the data types and number of event attributes contained in each event. Hence, an approach towards event-similarity intended to be integrated into such a generic environment must not only fulfill the requirements for one specific domain and fall short in others. Instead it must be generic, configurable and adaptable to multiple data sets.

1.2 Objectives

The aims pursued by this work are manifold. The first objective is to analyze and concrete the requirements for a similarity framework to be applied to event sequences. Many approaches and techniques towards similarity have already been published (see also chapter 2 - Related work), but none of these applies directly to the given data sets. Several current application areas are taken as a basis to find different use cases for similarity searching and derive a set of requirements to be covered by the similarity model.

[2] www.senactive.com

The second major objective was to define a coherent similarity assessment model, which is able to take into consideration the different data characteristics and also provides sufficient flexibility to be adjusted as required, for instance by configurable weighting factors and search pattern constraints.

The third and most comprehensive objective includes the development of algorithms to efficiently execute the similarity model. Hereby, the focus is set to enhanced techniques for considering different semantics of attributes (such as continuous value series spanning multiple events) and on modeling a search sequence in order to restrain the search process and optimize the matching.

Finally, the work aims at providing the resulting similarity search mechanisms in a user-friendly way to business analysts. Hereby, a compromise should be found between maximum control over the search process and minimum complexity of the user interface.

A decent performance evaluation with respect to different use cases rounds up the thesis.

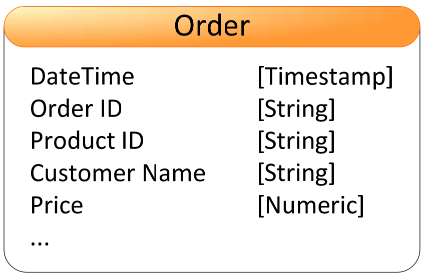
1.3 Data structure and data repository

This section describes the data representation the presented similarity search model is able to cope with, and provides insights into how these data are stored in the SENACTIVE *InTime*™ system.

Continuous capturing and processing of events produces vast amounts of data. An efficient mass storage is required to store all events and prepare the data for later retrieval and access. This mass storage is called *EventBase*, a specific database repository for events in the SENACTIVE *InTime*™ system. During the processing, events which should be kept persistent are pushed into this repository. Also, information about event correlations is captured and stored. In addition, the events can be indexed for later retrieval with full-text search as described by Rozsnyai et al. [42].

1.3.1 Single events

Events represent business activities. In order to maintain information about the reflected activity, events capture attributes about the context when the event occurred. Event attributes are items such as the agents, resources, and data associated with an event, the tangible result of an action (e.g., the placement of an order by a customer), or any other information that gives character to the specific occurrence of that type of event. For example, Figure 2 shows some context attributes of a typical order event.



Order	
DateTime	[Timestamp]
Order ID	[String]
Product ID	[String]
Customer Name	[String]
Price	[Numeric]
...	

Figure 2: Event type definition of simple order event

This template of attributes defines the structure of a certain class of events and is called e*vent type*. It indicates the underlying type of state change in a business process that is reflected by the event. The concept of event types is strongly related to the concept of a *class* in object-oriented programming (OOP). Event attributes might by of various data types. The SENACTIVE *InTime*^{TM} system supports all basic .NET runtime types such as *Int32* or *String*, but also multi-value types (lists, dictionaries) and arbitrary custom implemented objects. In addition, events can be nested as attributes in other events, whereby an arbitrary hierarchy is theoretically possible. The used event model is called SARI event model. It was originally proposed by Schiefer and Seufert [43] and described in greater detail by Rozsnyai et al. [41].

Figure 3 illustrates the event model in UML notation. Event types can inherit from other event types and may contain various attributes of different types.

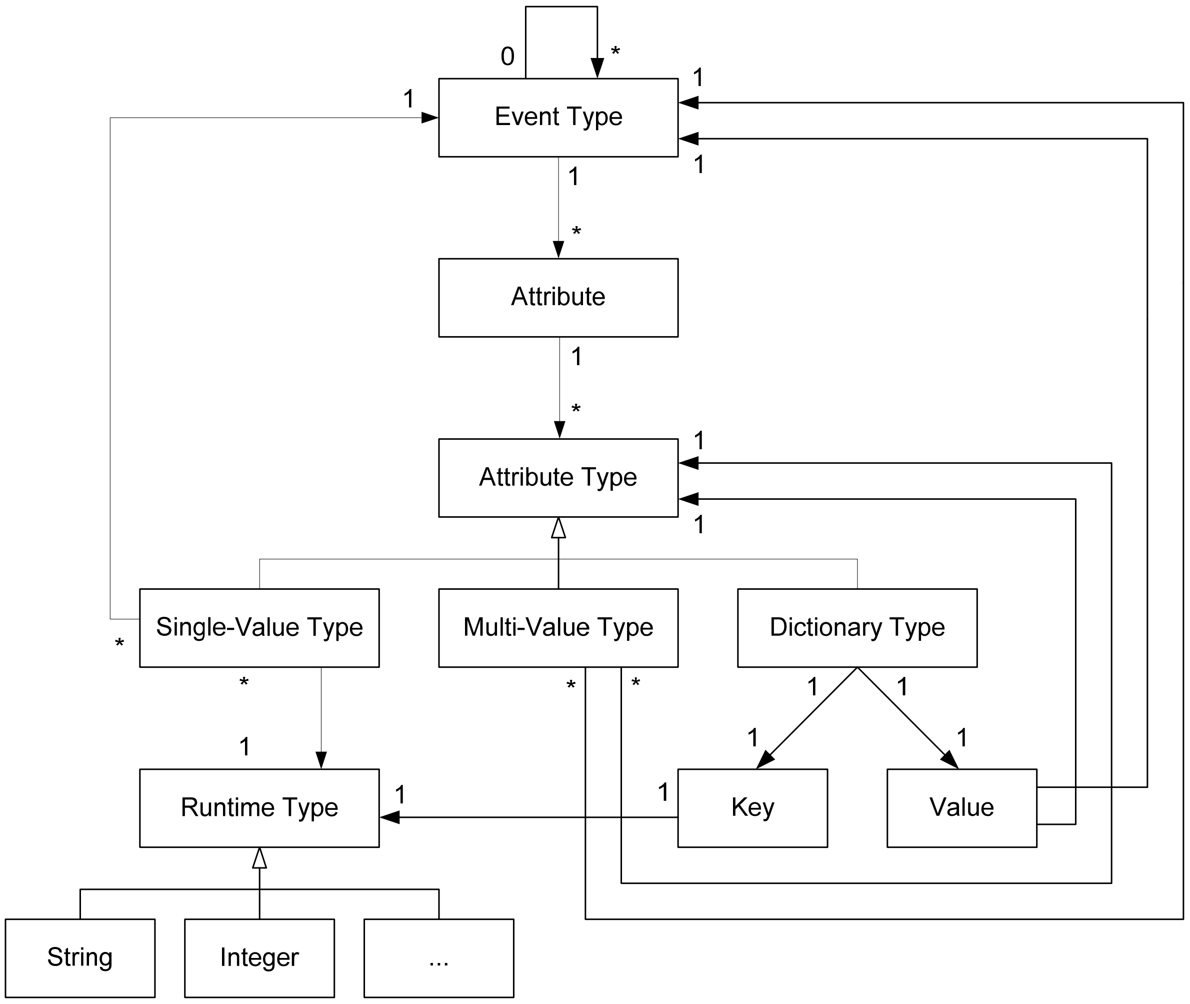


Figure 3: The SARI event model

1.3.2 Event correlations

In many cases single events do have a certain context and are semantically related to other events. For instance, a "task started" event is probably semantically related to a "task completed" event with the same task identifier. Correlations are sequences of semantically related events and form the basis for most of the following algorithms.

An event correlation is defined as a set of related events. A correlation set is a template definition for how correlations are identified. The correlation set defines tuples of attributes whose values must match in order for events to correlate.

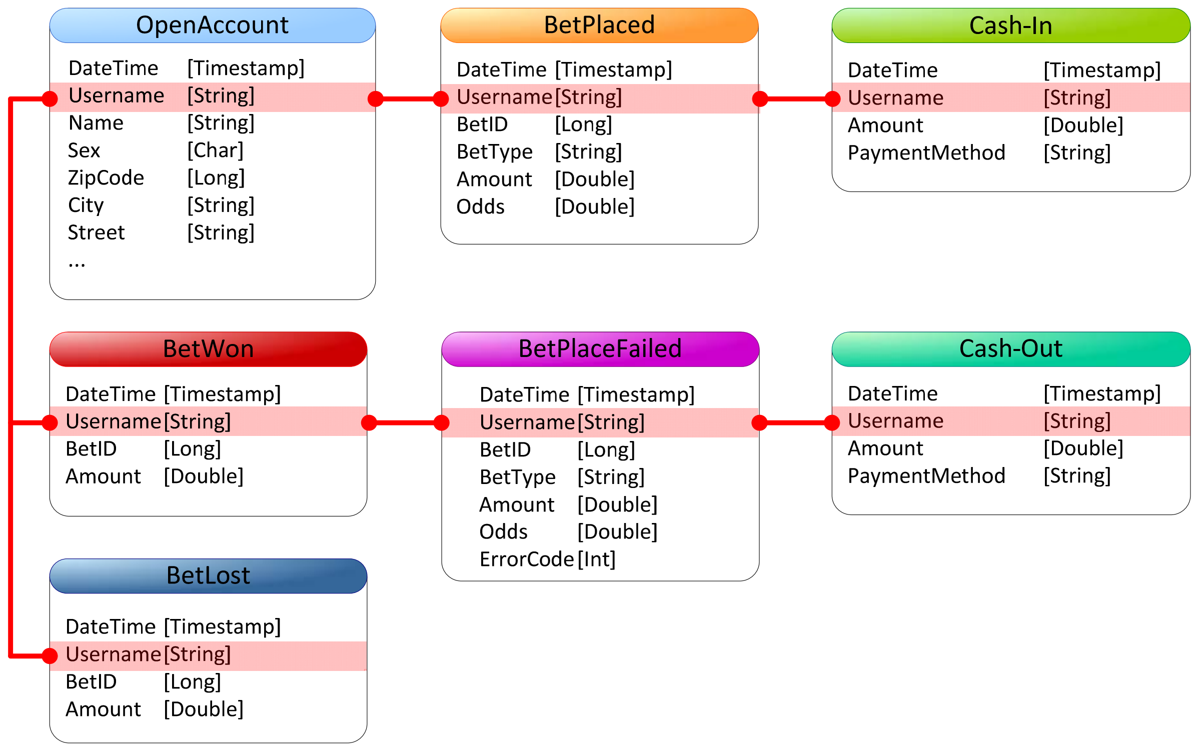


Figure 4: Correlation set definition

Figure 4 provides an example of a correlation set. Several events of different event types are correlated to a coherent sequence if the value of the attribute “username” matches. Such a correlation is not limited to a single event attribute, but can be defined based on multiple attributes. The red items are a group of matching tuples, each matching each other event type. Also, the order of the events occurring is not decisive. In case of a cash-in event occurring first and a cash-out event occurring second, these events will also be correlated. A sequence of correlated events may contain an arbitrary number of events of each event type. Thus, an event sequence based on the above correlation set may contain for instance 10 “bet placed” and and 2 “cash-out” events.

1.3.3 Database structure

In the *EventBase*, a specific table for each event type is automatically created when modelling the event type definition. This specific events table contains a separate column for each event attribute, whereby basic .NET runtime types such as *String* can be mapped directly to database types (i.e. *varchar*). Complex types such as lists or nested types are serialized to XML to ease handling. A generic event table contains an xml representation, id and timestamp of each event.

Correlations are also stored in the database. Per unique value group of correlation attributes a database entry exists, and a relational table links them to the actual events in the generic events table.

The *EventBase* also contains all required metadata used during the similarity search process such as event type definitions and correlation sets.

1.4 The SENACTIVE EventAnalyzer^{TM}

The SENACTIVE *EventAnalyzer*^{TM} is a business intelligence tool built on top of the *EventBase*. It allows the user to query the event data and generate interactive graphical views of events. Its major components are a search and query module, the patented event-tunnel visualization looking into the historic events like a cylinder, event charts, several configuration parameters for the visualizations such as colors mapping, size mapping, shape mapping and positioning of data points and utilities such as a snapshot functionality to capture analysis results and create ready-to-use view templates or a details view to browse all attribute values of an event. Figure 5 shows a screenshot of the *EventAnalyzer*^{TM} with some of the named modules.

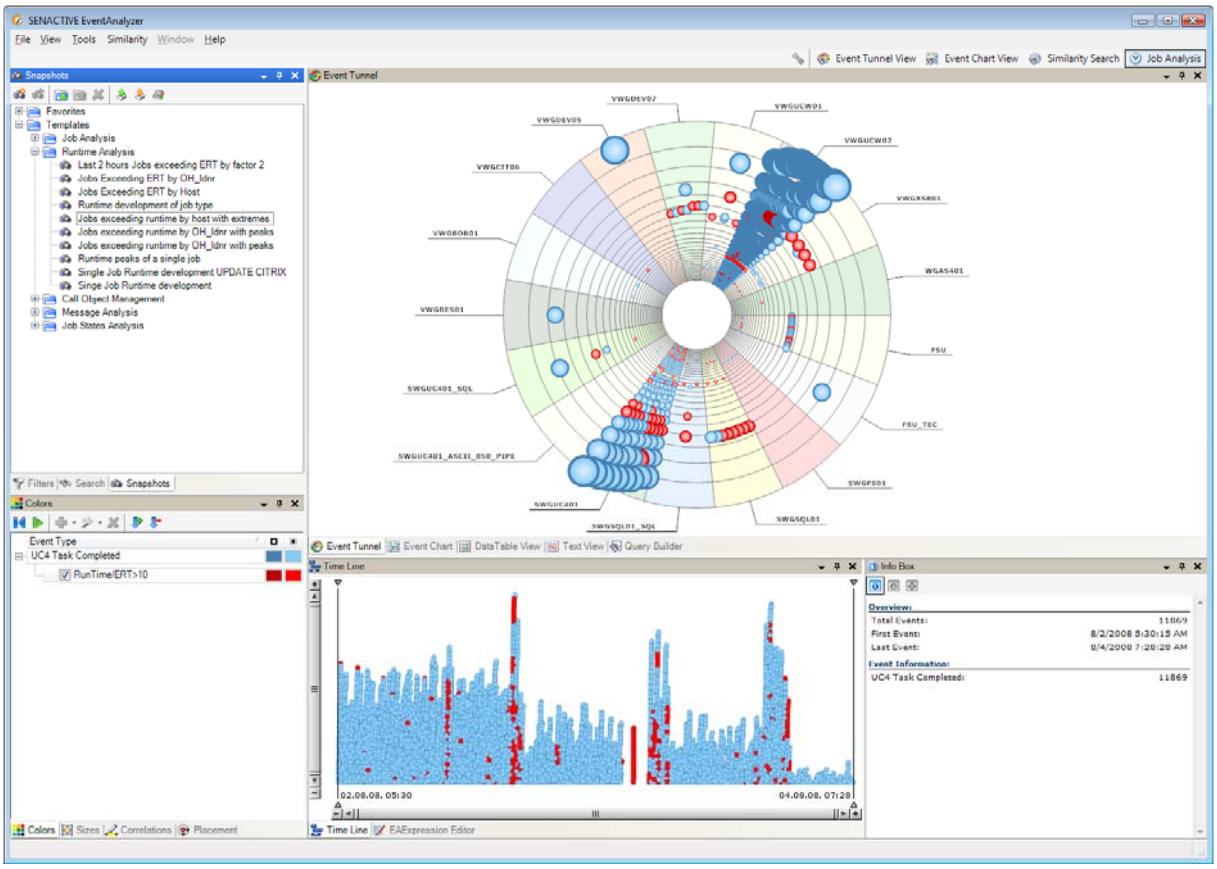


Figure 5: The SENACTIVE *EventAnalyzer*^{TM}

For further information on the visualizations provided by the *EventAnalyzer*^{TM}, the interested reader is referred to Suntinger et al. [48].

The *EventAnalyzer*^{TM} is intended to be a generic framework for event visualization and mining. It is constantly extended by new visualizations and data mining features. The elaborated similarity search mechanisms are also integrated directly into this framework. The objective is to trigger a similarity search directly from any of the visualizations to search for event sequences similar to those identified in the graphical views.

1.5 General remarks

The on-hand thesis builds upon a similarity model and framework that has been designed and implemented in collaboration with Hannes Obweger. The result of this collaboration was the basic model for assessing similarity between event sequences, considering various possible extensions. In his thesis [37], this basic model for determining the similarity between single events and sequences of correlated events is depicted in great detail and illuminated from a theoretical as well as an algorithmic point of view. Building upon this model, this work focuses on enhancements and extensions in order to cover requirements arising in different application domains. Among these extensions are enhanced event attribute similarity techniques and search pattern modeling and constraining. Hence, considerations on the base similarity model are reduced to necessary essentials in order to understand the presented model enhancements. For further, in-depth considerations the interested reader is referred to Obweger's thesis [37]. Also, the evaluation has been done in collaboration so that presented results in the evaluation section are overlapping as regarding the base similarity features.

2 Related work

This section discusses related work. It is divided into several categories, each treating a specific aspect of the on-hand thesis. The objective of the section is to give an overview of what has already been done in the context of and related to this work and has been taken as a basis for the event-based similarity model.

2.1 Similarity applications

In recent years a multitude of approaches and models have been published related to the broad topic of similarity searching. These models have been applied in various application domains. For instance, Agrawal et.al. [1] focus on discrete time-series databases and mention the following applications: company growth patterns, product selling patterns, stock price movement patterns and comparison of a musical score with copyrighted material. Pratt [38] applies time-series pattern searching to temperature measures and electroencephalogram data. Other datasets which have been used for testing are photon arrival rates (astronomy), space shuttle orientations during flights [25] and measures from production machines, like size deviations. Data set sizes presented in these works vary from a few thousand up to a couple of millions of data points. Another application for time-series similarity discussed for instance by Vlachos et al. [54] are location trails, so-called trajectories, which have fuelled the interest in similarity searching algorithms in recent times.
Aside of time-series similarity, Moen [34] proposes a model for attribute, event type and event sequence similarity. Application areas investigated in this work are news articles with keywords as attributes, and student courses enrolment data, whereby the courses are classified by several categories and properties. In addition, event sequence similarity was tested with a dataset of telecommunication company alarms and a WWW page requests log. Similar data were also investigated by Weiss and Hirsh, who try to predict telecommunication equipment failures from alarm messages [56].
Other applications requiring similarity search are image databases [30], biology/genetics (e.g. comparison of proteins and protein sequences [59]) and user behaviour patterns for interfaces [28].

In this article, several similar application areas are discussed, whereby some extend already explored application examples. For instance, the topic of news articles and the stock price movement patterns can be combined for detecting complex trading scenarios considering price movement and industry news at the same time. For other applications such as image retrieval or protein sequence similarity, the presented approach is not directly applicable.

2.2 Similarity models

For the different application areas discussed in section 2.1, also different similarity models for assessing the similarity between the items to be compared have been developed. Lin [27] describes 3 intuitive rules for assessing similarity: (1) Similarity is related to commonalities. The more commonalities two items share, the more similar they are. (2) Similarity is related to differences. The more differences two items have, the less similar they are. (3) The maximum similarity between two items is when they are identical, no matter how much commonalities they share.
On top of these basic assumptions, similarity models have been proposed which can be roughly categorized into [19]:

- Geometric models
- Feature-based models

- Alignment-based models
- Transformational models

Geometric models such as the nonmetic multidimensional scaling model (MDS) proposed by Shepard [44] try to express similarity by representing items as points in a usually low dimensional metric space and assessing the distance between the items in this space. Subsequently, similarity is inversely related the items' distance in the metric space. Resulting from the underlying geometric model, several mathematical basics apply for the similarity assessment. An example is the triangle inequality. Let $d: X \times X \rightarrow \mathbb{R}_{\geq 0}$ be a distance function in the metric space expressing the dissimilarity between two items, the triangle inequality defined as

$$d(a,b) + d(b,c) \geq d(a,c)$$

Formula 1: Triangle inequality

applies, whereby a, b and c are compared items. In the context of similarity, especially this triangle inequality may lead to "intuitively incorrect" results.

Due to this and further shortcomings of geometric models, Tversky [53] proposed an alternative, feature-based approach. The idea of Tversky's similarity model is that similarity is measures by common and distinctive features. Let $sim(a,b)$ denote an interval similarity expressing the similarity between two items a and b. Furthermore, let f be a scale defined on the relevant feature scale. Tversky proposed to compute the similarity between two items a and b as

$$sim(a,b) = f(A \cap B) - f(A - B) - f(B - A)$$

Formula 2: Tversky similarity model

with $A \cap B$ representing the features which a and b have in common. $A - B$ are features which a has, but b has not. Equivalently, $B - A$ are features b has but a has not. Later, Gati and Tversky [18] proposed to multiply these values with different weighting factors. Factor θ weights common features, α is the weight for unique features of a and β is the weight for unique features of b. The resulting formula is called the contrast model:

$$sim(a,b) = \theta * f(A \cap B) - \alpha * f(A - B) - \beta * f(B - A)$$

Formula 3: Tversky and Gati contrast similarity model

For instance, common features are weighted stronger as compared to distinct features. Based on common and distinct features, also other computation models have been proposed. Examples are the Sjoberg similarity model [46]

$$sim(a,b) = \frac{f(A \cap B)}{f(A \cup B)}$$

Formula 4: Sjoberg similarity model

which computes similarity from the ratio of common features to the total number of features, or the Eisler and Enkman similarity model [14] and the Bush and Mosteller similarity model [7].

$$sim(a,b) = \frac{f(A \cap B)}{f(A) + f(B)}$$

Formula 5: Eisler and Enkman similarity model

$$sim(a,b) = \frac{f(A \cap B)}{f(A)}$$

Formula 6: Bush and Mosteller similarity model

These three models can all be seen as a variation of the general equation

$$sim(a,b) = \frac{f(A \cap B)}{f(A \cup B) + \alpha f(A - B) + \beta f(B - A)}$$

Formula 7: General ratio function for feature-based similarity

which differs from Tversky's contrast model by applying a ratio function as opposed to a linear combination of common and distinctive features [22].

Most of these models have been tested exclusively for the similarity of images, and the formulas emerged as the best similarity measures for the given purpose and the selected features. Thus, a feature-based similarity approach strongly depends on the feature selection, and is currently applied mainly in the area of retrieval in image databases.

Alignment-based similarity models have been developed to overcome some of the shortcomings in feature-based models, especially in the domain of image comparisons. The main idea behind alignment-based models is the following: When comparing an image of a woman wearing a red hat and a car having a red hood both share the common feature "red". In an alignment-based model, such a common feature may not increase the similarity score, because the hat does not correspond to the car's hood. Markman and Gentner [33] argue that similarity is more accurate and intuitive, if matching features are weighted stronger if they belong to parts that are placed in correspondence, whereby they refer specifically to images.

The last one of the four essential similarity models is the transformational model. The idea behind this model is to assess similarity by the costs required to transform one item into the other. Hereby, different transformation operations may have different costs. For instance, Moen [34] applies such a model to event sequences. Transformation operations are moving an event, insertion and deletion. The idea is to first find the sequence of transformations which is most efficient in terms of transformation costs and to assess the similarity based on the sum of all transformation costs for the ideal sequence of transformations.

The different approaches for defining and computing the similarity between two items form the basis for the similarity model applied in this thesis. A geometric model has known shortcomings such as the triangle inequality problem, but brings the advantage of being "exact" in terms of comparing the original items instead of meta-information about the items. This makes it applicable only to a limited subset of data types.
The feature-based model brings the advantage of being able to deal with huge masses of data. In addition, many experiments have proven that it often leads to intuitive results. Yet, the model strongly depends on the right features being selected. Current research efforts mainly focus on feature selection for images. For event sequences, no equivalent publications are available.

The alignment-based model solves several characteristic shortcomings of feature matching in image processing, and the idea of making the feature weighting dependent on whether the feature context is similar may be adapted to the on-hand requirements.
Transformation models have been shown to be applicable also in the domain of event sequence similarity. One open issue of the approach is the handling of sub-item matching.

2.3 Event sequence and attribute similarity

The general similarity models discussed in section 2.2 are taken from various application domains. Many are strongly related to the image retrieval domain and have their origin in cognitive psychology. In this section, related publications are discussed which deal specifically with event sequences or cover the similarity assessment of attributes.

Moen [34] proposed a model for attribute, event sequence and event type similarity, whereby the event sequence similarity model has originally been published by Mannila and Moen in [31]. Thereby, the attribute model is a simple pairwise similarity computation which considers the complete set of values as a reference. The event sequence similarity model uses the edit distance between two event sequences. First, the minimal number of transformations to transform the first sequence into the second one is found (transformations are insertion, deletion and moving in time), and subsequently the similarity is assessed by the costs of these operations. The edit distance is computed using a dynamic programming algorithm. The event type similarity model treats the question of how the type of an occurring event can be considered for the similarity. For instance, two different types of alert events may be considered as being similar, even if it is not the same event type, because they are semantically related.
While the edit distance approach towards event sequence similarity is intuitive, it has several shortcomings: subsequence matching is not supported by this approach. Therefore, only sequences expected to have equal length can be compared. In addition, the edit distance computation takes time $O(nm)$ for sequences of lengths n and m. Also, finding a suitable cost model for the edit operations is problematic.

Mannila and Seppänen [32] try to alleviate some of these shortcomings and propose an approach which makes use of random projections assigning each event type a random k-dimensional vector. For the searching process, the vector of the pattern sequence is compared to the data set and some items where the distance between the vectors in the k-dimensional space is the smallest are retrieved. In a next step, the edit distance approach is used to compute a precise similarity score. Due to the fact that most of the search can be performed in k-dimensional Euclidian space and the vectors can be hold in index structures such as an R-tree [21], the method performs well for large data sets.

The issue of attribute similarity is discussed by Lin [27] in an information-theoretic view on similarity. The publication discusses similarity of ordinal values based on the distribution of values in the data set, feature vectors and string similarity. Das et al. [13] point out that similarity metrics cannot only be user defined, but also defined on the basis of the data. Their similarity notion considers relations to other attributes and two items are considered to be similar, if they share similar relations. Such relations can for instance be determined with known data mining approaches such as clustering and association mining [8].

2.4 Time series similarity

In terms of event-based similarity search, time-series similarity can be seen as a specific type of attribute similarity for numeric event attributes. The major difference is that it is not an attribute similarity technique comparing attributes on an event-by-event level, but at the level of the complete sequence of attribute values. Translating a sequence of events to a time-series means seeing each event as a data point in time and taking an event's numeric attribute as the corresponding amplitude of the time series at the concerned point in time.

Many approaches have been published towards efficient similarity algorithms for time series. These are intended to be applicable for various computations including indexing, subsequence similarity, clustering, rule discovery and many more.
Many of the similarity models published so far for the comparison of time-series are based on the idea of dimension reduction, which is to transform the original signal into a transformed space and to select some subset of the transformed coefficients as features.

The first one to apply dimension reduction for time-series similarity was Agrawal et.al. [1] [2] who used Discrete Fourier Transformation (DFT) for the dimension reduction. Other approaches based on DFT can be found in [12], [8], [15] and [40]. The DFT is used to map the time series to the frequency domain. The first few Fourier coefficients, which represent the time series accurately are then indexed using an R*tree, which can then be used for fast retrieval. The major shortcoming of the DFT is its unsuitability when signals have discontinuities. It is well-suited for sinus-like signals.
Discrete Wavelet Transformation (DWT) is an alternative approach to DFT-based dimension reduction. The Haar wavelet is most commonly used for this purpose [47] but other wavelets are applicable as well and provide reasonable or better results, as discussed by Popivanov and Miller [9]. The main problem of wavelets is that they are not smooth. Therefore, for approximating smooth time series many coefficients are required, which in turn reduces the performance. A further discussion on dimensionality reduction with DFT and DWT can be found in [24] and [57].
A third dimension reduction approach is Singular Value Decomposition (SWD) proposed by Korn et al. [26]. It uses the KL transform for dimension reduction, but is inapplicable in practice, because it needs to recompute basis vectors with every database update.
Piecewise Aggregate Approximation (PAA) [58] is a fast dimension reduction technique. It performs the reduction by subdividing a time series into subsequences of equal length. Taking the mean of each subsequence, a feature sequence is formed. Obviously, the major problem of the approach is that it only provides a rough estimation of similarity.

Toshniwal and Joshi [50][51] propose a distinct similarity model for time series based on slope variations. In a preprocessing step, time series are brought to the same time range and the coefficients are proportionally scaled. After the preprocessing, for small subsequences of equal length, the slopes are compared, and for the similarity assessment, the cumulative variation in slopes is computed. The technique can handle vertical shifts, global scaling and shrinking as well as variable length queries. One shortcoming of the approach is the missing support of subsequence matching.

Negi and Bansal [36] generalized Agrawal's basic model in order to allow subsequence matching and variable length queries. In the model, the data a first preprocessed. The second step is a so-called Atomic Matching trying to find source subsequences matching target subsequences. A KD-tree is used for indexing the items. In a third step, the subsequence matching, it is tried to stitch all subsequences to form a long sequence matching the target sequence.

Vlachos et al. [54] argue that for efficient retrieval, additional mechanisms that integrate above discussed distance computations may be required. The proposed solution is an index structure capable of supporting multiple distance measures.

2.5 Similarity pattern modeling and search interfaces

Very early considerations on interfaces and how to provide fuzzy searching to users can be found in the work of Motro [35] who proposed vague queries for relational databases. The idea was to extend the relational model with data metrics as definitions of distances between values of the same domain. Though innovate, entering textual, vague queries is still difficult for the user.

In the area of genetics, a set of tools with simple user interfaces exist focusing on searching biological sequence databases. Examples are SimSearcher [52] or DELPHI [17]. Yet, these interfaces do not allow directly entering or modifying a search pattern, but are limited to configuration options or general search constraints and an output of search results.

The most wide-spread application which is in worldwide use probably is BLAST (Basic Local Alignment Search Tool) [3]. BLAST is an umbrella term for searching tools to compare DNA and amino sequences to existing and documented sequences.

One noteworthy project is called Smart Sequence Similarity Search (S4) System, proposed by Chen et.al. [11]. S4 is an expert system with a web-based user interface which helps biochemical researches not experienced with similarity search algorithms to choose for the right search method and parameters. The underlying expert knowledge is a decision tree, which can be edited by expert users in a separate interface. This advising tool helps users getting started with difficult sequence similarity searches. The agent-based user interface is especially valuable in case of many different algorithms to choose from and many parameters to be adjusted. Introducing a recommendation system or wizard for event sequence similarity searching would be possible as well and could help in speeding up the learning phase with the software.

Berchtold and Kriegel [4] proposed S3, a system for similarity search in CAT database systems. S3 supports the query types “query-by-example”, “query-by-sketch” and “thematic-query”. A sketch-based user interface is also presented by Pu et.al. [39] for the retrieval of 3D CAT models. Hereby, the user can draw simple 2D freehand sketches and search for similar figures in the model database. It is possible to sketch the front view, the top view and the side view separately.

Wattenberg provides a sketch-based interface specifically for querying stock prices [55]. QuerySketch[3] is a prototype program where the user can draw a stock chart over a given, fixed time period and the system immediately searched for similar stock movements. The interface is very simplistic but still intuitive and simple to use.

In summary, user interfaces for similarity searches are still in their infancy. Query language models have the downside of being complex and hard-to-learn. The advantage is that they offer precise control over the searching process. Sketch-based models appear to be most promising for object and media searches. Even time-series retrieval is easily possible by query sketching. Still, what remains apart from modeling a search pattern is the necessity to set adequate configuration parameters for the various search algorithms. This task is addressed by agent-based expert systems, guiding inexperienced users though the configuration and selection process.

[3] At the time of writing this paper, an online demo is freely available at http://www.bewitched.com/projects/querysketch/sketch.html

3 Application examples and arising requirements

Event-based similarity search is a broad topic. Event-based systems as such may be applied in various application domains, so can be event-based similarity search. Accordingly, the requirements are manifold. In this section, several application domains are discussed in order to derive the matching requirements for event-based similarity search. Based on these requirements we subsequently defined the similarity assessment model.

3.1 Finance - market analysis and trading scenario discovery

3.1.1 Overview

For market analysis, a major application of similarity search is the discovery of stock chart patterns and correlations between several traded values (e.g. correlation of gold price with a certain gold explorer stock, or correlation of a currency with an exporting company's stock). When applying event-based similarity search, besides time-based price series additional information can be taken into consideration for the discovery of complete scenarios. For instance, news events can be considered to search for a chart pattern where at a certain point a decisive news event was published, influencing the price.

Figure 6 depicts several event types which may occur in an event-based stock market analysis application. For the options and futures market, instead of the stock ticks other data may be available, but basically the data will be the same. For the foreign exchange, ticks will be available for pairs of currencies.

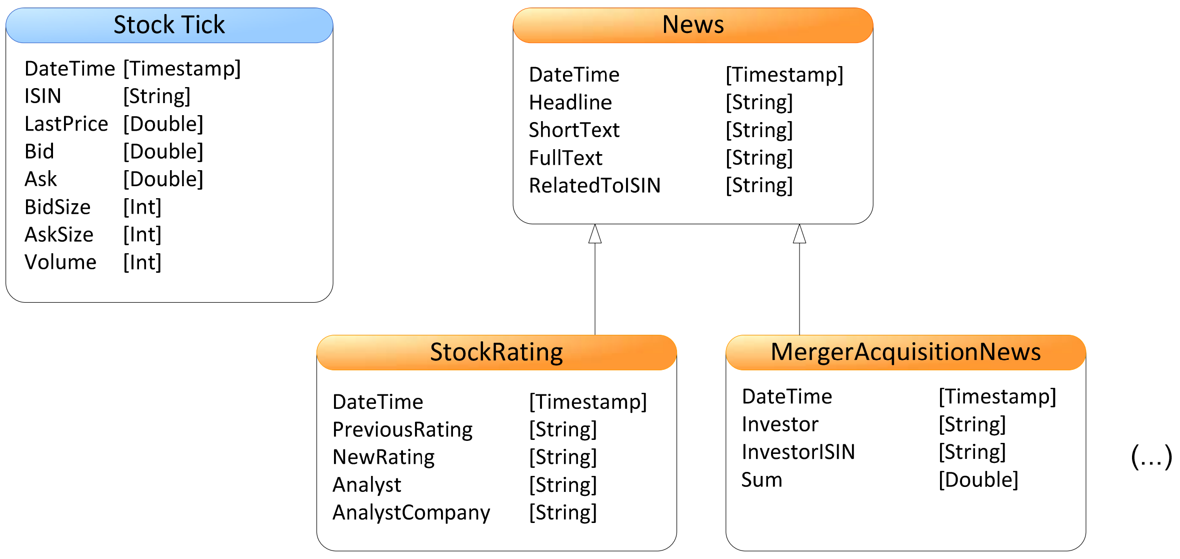


Figure 6: Event types for event-based stock trading

3.1.2 Similarity search example – trading scenarios

Many traders have a set of trading scenarios in mind, which they try to detect. On occurrence they buy or sell accordingly.

As an example, Figure 7 depicts such a trading scenario. In this case, a stock whose price first moved sideward and formed a support level rose strongly, but after several news events it plunged down again to the support level. A trader could for instance want to buy exactly at the support level after the plunge, to profit from a little rebound at this level, which is likely to occur.

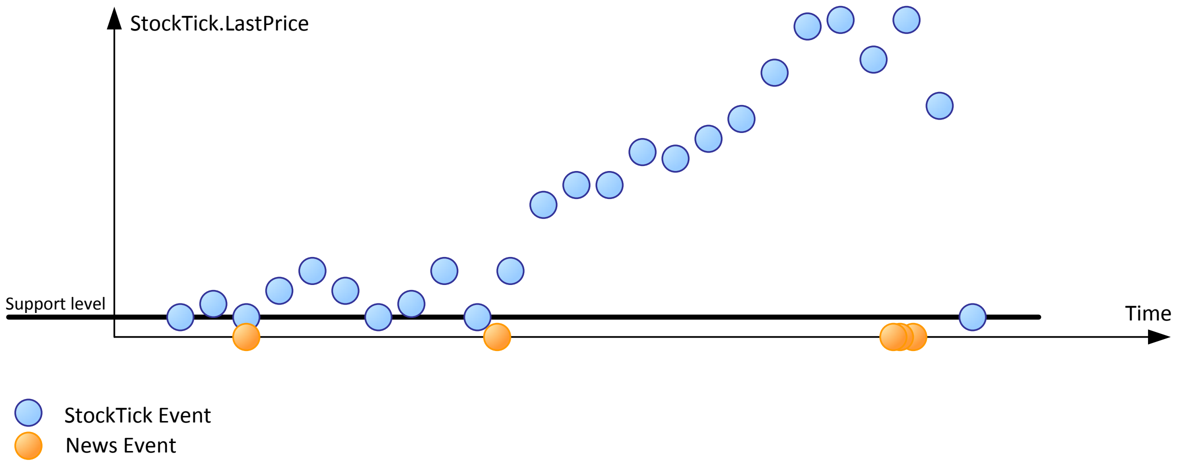


Figure 7: Trading pattern of stock ticks and news events

Such a pattern is easy to detect manually, when looking at the chart. On the other hand these scenarios are quite rare. Therefore, it would be valuable to detect it among thousands of stocks, which is not possible manually. Hence, a similarity search which is capable of a fuzzy detection of such a pattern is required.

3.1.3 Requirements for similarity searching

In order to apply similarity search in this area of financial market analysis and automated trading, at least the following requirements have to be covered:

- It must be possible to not only compare numeric event attributes in an event-by-event matter with absolute difference similarity, but also to compare the complete sequence of values in the pattern sequence to the sequence of attribute values in the target sequence (time-series similarity).
- Time-series similarity for attributes must be independent of absolute values, and ideally also support different relative scaling of the complete pattern.
- It should be possible to "weaken" the search sequence. For instance, in the example, the number of news events is not relevant, so the occurrence of one news event is as equal as the occurrence of 5 news events.
- Similarity search must deal with different length of event sequences.
- It should be possible to omit certain parameters for the similarity search. For instance, the event attributes of the news events are not relevant, but rather their occurrence only. Also, for the tick events only the attribute "price" is relevant.

3.2 Online betting fraud detection – user behavior profiles

3.2.1 Overview

In online betting and gambling, one important issue is fraud detection and prevention. Hereby, one approach is to selectively filter user actions by rules in the sense of "If a user does XY, then block this user". Yet, the definition of "if the user does XY" is not as easy as it looks at first sight. The possibilities of strict rules on incoming events are limited. In this way, only exactly defined data values and thresholds can be tested. An alternative approach is to use behavioral patterns, and formulate the rule as “If a user behaves similar to pattern XY (a known fraud pattern), then block this user”. For the latter approach it is required to compare the user behavior profile of a user with those of other users. The problem of the similarity approach is that it might be too fuzzy, because the behavior of users, incorporated in sequences of events can vary, but might still be similar. In order to alleviate this problem, maybe a hybrid approach of a fuzzy similarity search couples with a set of rules could be applicable. Yet, a further discussion of this issue is beyond the scope of this work.

Figure 8 shows a set of typical event types for an online betting environment. In the following, a similarity search example is defined based on these event types.

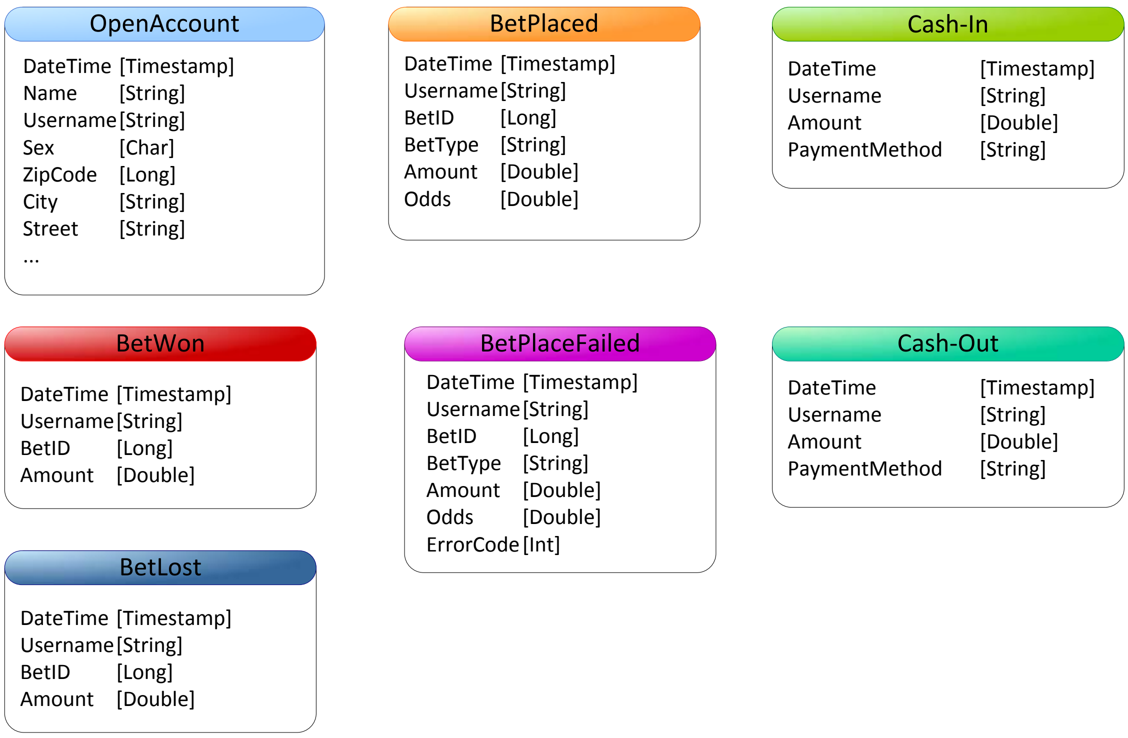


Figure 8: Event types in an event-based online betting application

3.2.2 Similarity search example

Applications in online betting and gambling are mostly one of the following: fraud detection, or the discovery of cross/up selling opportunities with custom recommendations. For fraud detection, the recognition of behavioral patterns is a valuable approach. Fraud as such, and also “suspicious behavior” is hard to define. Yet, it is possible to take a behavioral profile from a known fraudster and compare it to others.

An example of a characteristic behavior profile is depicted in a simplified matter in Figure 9. Here, a so-called sleeper account is illustrated. This user hasn’t placed bets for quite a long time, only one small bet directly after opening the account, but then cashes-in a high amount, places a bet for nearly the same amount, wins it and cashed out immediately. This sequence repeats a second time.

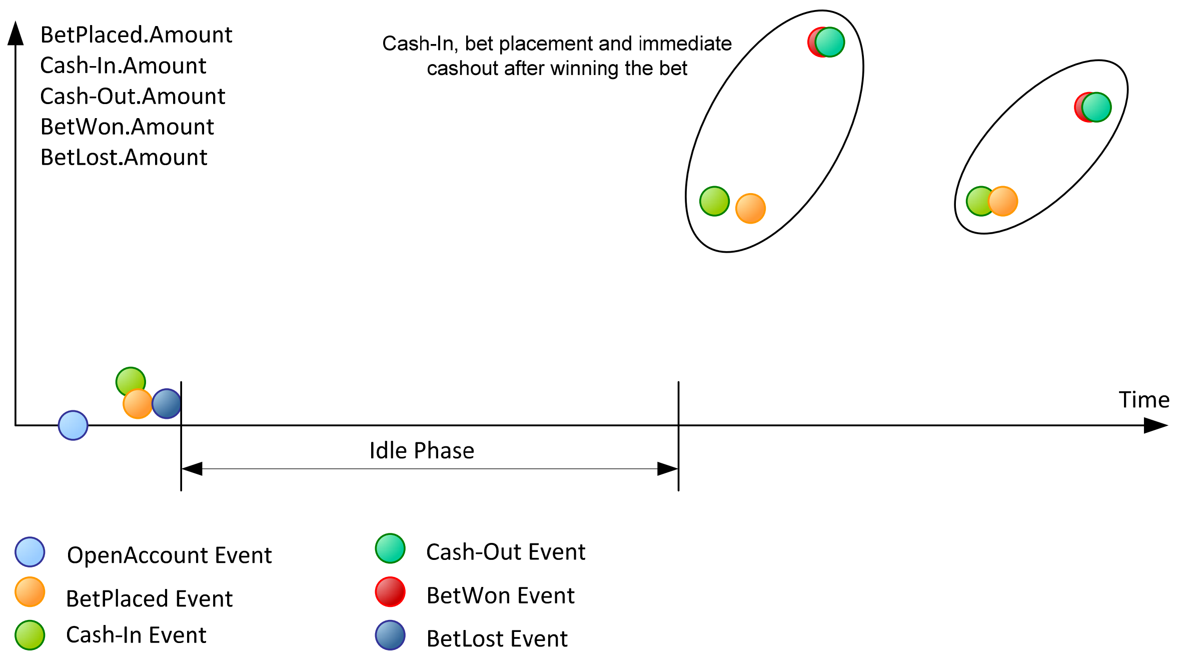


Figure 9: Example for a similarity search pattern in online gambling

While this sequence of events is not fraud per definition, it may be an indication, because it is an unusual betting behavior as compared to typical customers. For instance, that fact that the high-stake bet is placed after a long idle time may indicate that the user is very sure of this bet. Maybe she has insider information.

3.2.3 Requirements for similarity searching

From the above example, for the area of fraud pattern searching, the following requirements for similarity searching can be derived:

- The occurrence times of events should be considered.
- In the example, the length of the idle time is not decisive as long as it is above a certain threshold. It should be possible to model that for instance the idle time can be between 1 months and 5 years without changing the similarity scoring.
- It should be possible to model that a recurring sub sequence of events such as the sequence of cash-in, bet placement, bet won and immediate cash-out may occur multiple times without decreasing the similarity score.

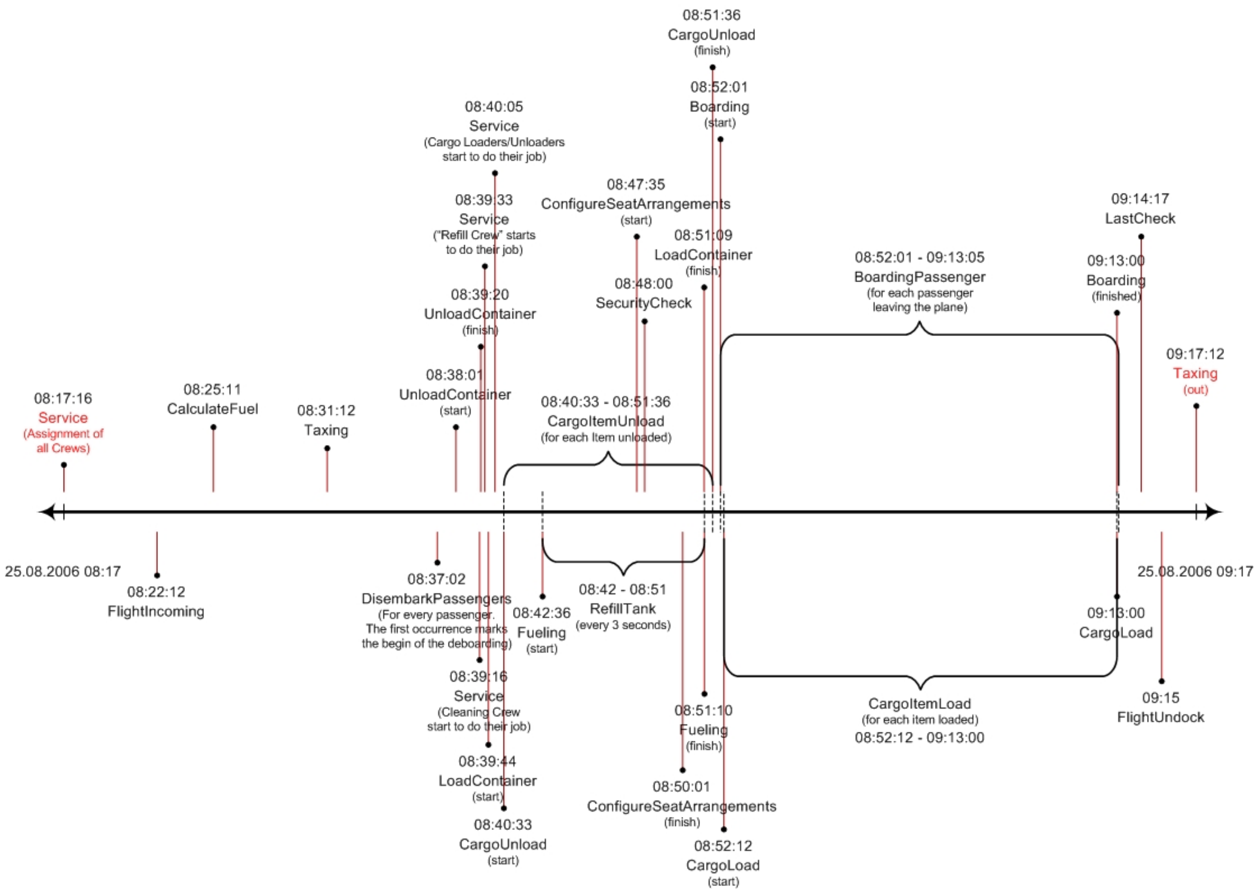
3.3 Airport turnaround – detecting process deviations

3.3.1 Overview

On airports, the sequence of actions which are to be performed from when an airport lands to its takeoff is typically a standardized process, including deboarding, reflueling, cleaning, and many more steps until boarding and takeoff. The detection of deviations from the typical process can be done either by checking every single action in the process with a specific rule, or, more intuitively, by comparing a process instance with a default process and assessing the similarity between these processes. In this way, the deviation assessment is not bound to hard value thresholds, but it is a fuzzy comparison of the complete sequence pattern. For historic data analysis, it may be of interest, to retrieve those processes, where the most decisive deviations occurred, for instance to answer questions like “Which airline caused the most deviations?” or “At which time of the day do most of the deviations occur?”

3.3.2 Similarity search example

In Figure 10, the events in a typical turnaround scenario are depicted in temporal order. As an application example it could be required to take this sequence as the normal process execution, and perform a similarity search to discover sequences with strong deviations from the typical process execution. Hereby, mainly the occurrence of events of a certain type is relevant.



08:51:36 CargoUnload (finish)
08:52:01 Boarding (start)
08:40:05 Service (Cargo Loaders/Unloaders start to do their job)
08:47:35 ConfigureSeatArrangements (start)
08:39:33 Service ("Refill Crew" starts to do their job)
08:51:09 LoadContainer (finish)
09:14:17 LastCheck
08:52:01 - 09:13:05 BoardingPassenger (for each passenger leaving the plane)
09:13:00 Boarding (finished)
08:39:20 UnloadContainer (finish)
08:48:00 SecurityCheck
08:38:01 UnloadContainer (start)
09:17:12 Taxing (out)
08:17:16 Service (Assignment of all Crews)
08:25:11 CalculateFuel
08:31:12 Taxing
08:40:33 - 08:51:36 CargoItemUnload (for each Item unloaded)
25.08.2006 08:17
08:22:12 FlightIncoming
08:37:02 DisembarkPassengers (For every passenger. The first occurrence marks the begin of the deboarding)
08:42:36 Fueling (start)
08:42 - 08:51 RefillTank (every 3 seconds)
25.08.2006 09:17
09:13:00 CargoLoad
08:39:16 Service (Cleaning Crew start to do their job)
08:51:10 Fueling (finish)
CargoItemLoad (for each item loaded) 08:52:12 - 09:13:00
09:15 FlightUndock
08:39:44 LoadContainer (start)
08:50:01 ConfigureSeatArrangements (finish)
08:40:33 CargoUnload (start)
08:52:12 CargoLoad (start)

Figure 10: Airport turnaround scenario

3.3.3 Requirements for similarity searching

From the above example, the following requirements can be derived:

- The weighting of certain characteristics, such as the occurrences time of events, should be adjustable.
- Event attributes such as the flight ID are not relevant and it should be possible to completely omit them for the similarity searching.
- The discovery logic should be invertible, so that the similarity search can also be used to retrieve the most deviating sequences.

3.4 Other application areas

3.4.1 Supply-chain/shipment processes

Shipment processes and supply-chains are standardized within large companies. Such processes reach from the initial customer order over order processing to the manufacturing, shipment and finally the delivery at the customer. For optimization, the analysis of historic processes is of interest. Hereby, a first step is to have the processes visualized to see how a normal process evolves. The second step is search for processes that are not similar to the default case, and why there where deviations. This step leads to the error cases where optimization potential is available.

3.4.2 ITSM – Trouble-ticket tracing

IT Service Management (ITSM), including the support of business processes by IT has grown to an important business factor in recent years. One major component of ITSM is the efficient management of so-called trouble tickets. Trouble tickets are issues reported by users. Subsequently, a member of the service team picks up the ticket and resolves the problem. Similar to bug tracking systems in software development, such issues may reoccur and similar issues may be reported by different users. In order to enable a steady improvement of service quality, it is essential to evaluate these trouble tickets, and find those which occur very often, or have some kind of noticeable history.
If an interesting history for a certain ticket is discovered, it may be of interest to discover other tickets with a similar history. One concrete requirement of a large IT service provider is to find similar assignment patterns of events. This company faces the problem of tickets being assigned from support group to support group (and back) until finally the responsible group receives handles it. The problem is that it’s not totally clear in which cases this occurs and for which groups. Only certain reference cases have been discovered. Based on them a similarity search could help to evaluate if there are many similar cases and it can be considered as a recurring assignment pattern. With this knowledge the assignment process can be optimized.

3.4.3 Clickstream – Usage patterns

In e-commerce, custom and intelligently placed product recommendations on a website, the webshop layout and the presentation of the offers is a key factor to success. Thus, in order to design a webshop as efficiently as possible, customer usage patterns have to be explored and understood in detail. For this purpose, many techniques exist, reaching from visualizations such as heatmaps to trace statistics. In recent times, the analysis of trajectories, i.e. navigation paths in programs and websites has grown to an interesting application for similarity mining. With the support of similarity analysis, behaviour patterns could be clustered in different groups, and also repeating usage patterns (eventually event some which are unsatisfactory for the customers) could be discovered.

4 Similarity assessment model

According to the requirements emerging from the above examples, a model for similarity assessment can be derived. The model determines how similarity between two sequences of events is defined and what influences the similarity computation. The model also considers the requirements for search sequence and constraint modeling.

4.1 Summary of approach

In section 2.2 – "Similarity models" a set of similarity model classes have been introduced. Namely these have been the geometric models, feature-based models, alignment models and transformational models. Feature-based models strongly depend on the feature selection process. For image and object searches this is a well-researched problem, but for event sequences it is yet an open issue. In addition, the basic idea of feature-based models which is, to put it crudely, to extract certain features and see how many features two items have in common and how many differentiating features they have does not apply in the given context. In our case the event sequences' features are well known, i.e., the strongly typed event attributes, the sequence of types, the occurrence times of events etc. It is decisive which *specific values* these known features have. We therefore decided not to use a feature-based model. Alignment models are closely related to the image retrieval and bioinformatics domain as well and cannot directly be employed in our given context.

Transformational models are proven to be usable for event sequence comparison. Yet, in the on-hand case with many more similarity features to consider and the requirement to perform subsequence matching and take sequence constraints into consideration such a model is difficult to apply and an efficient algorithmic evaluation is complicated. Finally, the idea of geometric models remains. The core idea is to have a set of single data characteristics which are to be compared. Each characteristic can be seen as one dimension in an n-dimensional feature space. Subsequently, similarity is assessed based on the distance between two items in the geometric space. Thereby, different metrics can be used, for instance Euclidian distance or the city-block metric. The problem with this approach is that the features must be numeric, or have to be mapped to something numeric. In case of complex events, with string event attributes, multi-value types or nested events to be included in the similarity computation this is not intuitively possible.

We therefore designed an adjusted similarity model. Simply put, it foresees a range of individual similarity features, each computed separately. The overall similarity score is then a computed aggregate value from these individual functions. The computation model corresponds to the simple weighted average model proposed by Gowser [20]. Let $sim(a, b)$ with denote the similarity between two event sequences a and b. We compute its value as

$$sim(a, b) = \frac{\sum_{i=1}^{n} sim_i(a_i, b_i) * w_i}{\sum_{i=1}^{n} w_i}$$

Formula 8: Similarity aggregation

with a_i to a_n and b_1 to b_n being the features to be considered and sim_i being the respective similarity function for the i^{th} feature. In addition, w_i is a weight or weighting function for the concerned feature, which returns a normalized value between 0 and 1.

Figure 11 illustrates all aspects of event sequences which are currently considered. Each of these aspects is described throughout this chapter.

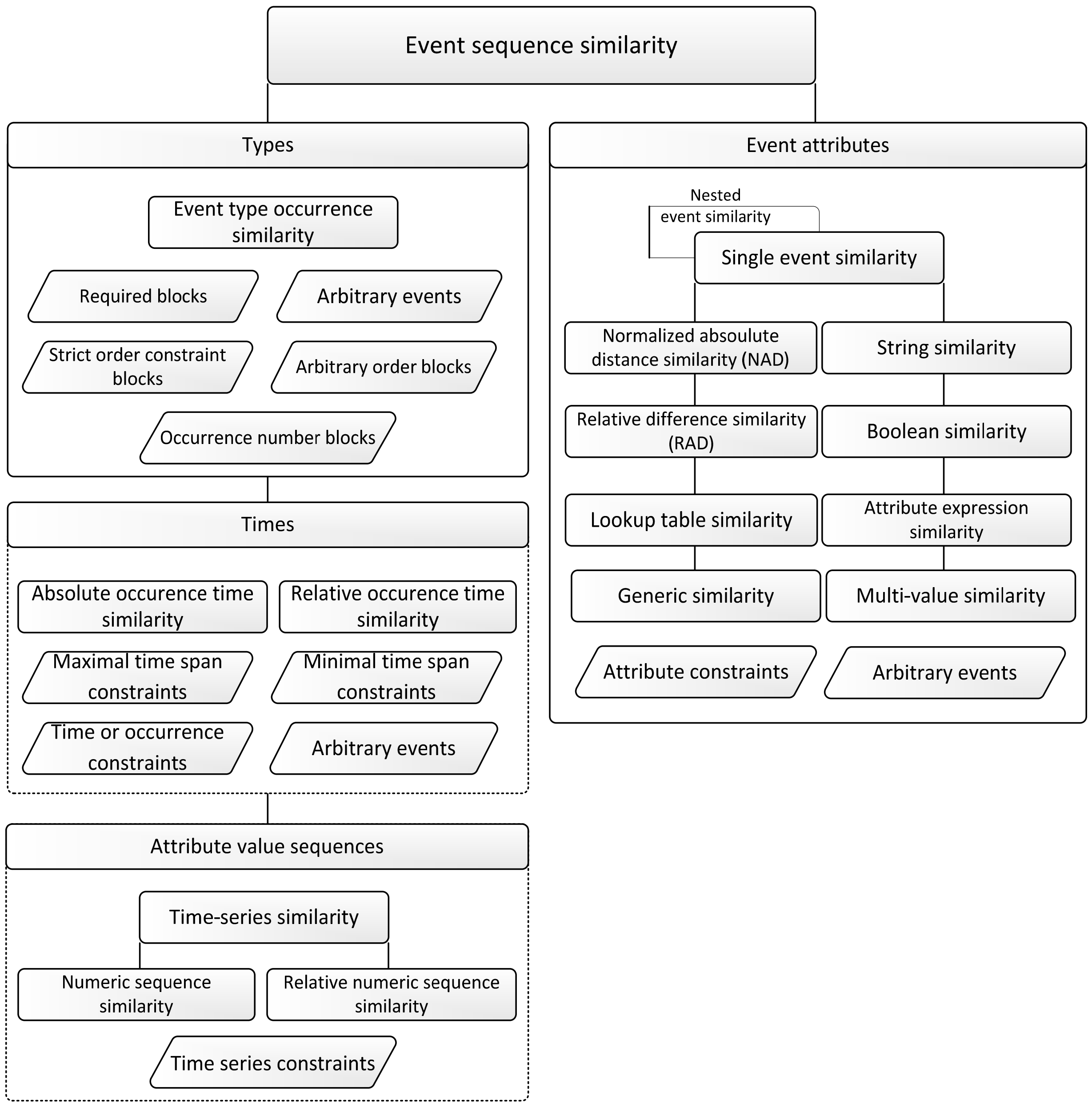

Event sequence similarity
Types
Event type occurrence similarity
Required blocks
Arbitrary events
Strict order constraint blocks
Arbitrary order blocks
Occurrence number blocks
Times
Absolute occurence time similarity
Relative occurence time similarity
Maximal time span constraints
Minimal time span constraints
Time or occurrence constraints
Arbitrary events
Attribute value sequences
Time-series similarity
Numeric sequence similarity
Relative numeric sequence similarity
Time series constraints
Event attributes
Nested event similarity
Single event similarity
Normalized absoulute distance similarity (NAD)
String similarity
Relative difference similarity (RAD)
Boolean similarity
Lookup table similarity
Attribute expression similarity
Generic similarity
Multi-value similarity
Attribute constraints
Arbitrary events

Figure 11: Overview similarity model

4.1.1 A multi-level similarity approach

In practice, in order to balance the similarity computation and, speaking colloquially, to not compare apples with pears we need to introduce multiple levels of similarity. We define a *multi-level similarity computation model* as a model in which not all individual features are aggregated directly according to Formula 8. Instead, first the “lowest level” similarity features, i.e. the single event attribute similarities are aggregated to one event-to-event similarity. This single event similarity is then aggregated with similarity features on event sequence level to the overall event sequence similarity. Without applying this multi-level approach, event sequence level features would be overruled by a potentially large set of event attribute similarities through the weighted average process.

4.1.2 Similarity versus distance

Up to this point, we have expressed similarity in terms of a similarity function $sim: A \times A \rightarrow [0,1]$, i.e., a normalized function returning values between 0 (unequeal) and 1 (equal). The approach is intuitive and brings the advantage that all results of individually assessed similarity features are directly comparable and combinable. Yet, in practice it imposes one major downside caused by the fact that the maximum *dissimilarity* between two items is 0. Considering an example of three similarity aspects or properties $p1$, $p2$ and $p3$ illustrates the problem. Taking two items a and b to be compared, let us assume that the similarity functions sim_{p1} through sim_{p3} each access these properties of the items a and b and assess similarity based on their values. Let us further assume that these properties are optional and may be totally absent. In case of total absence of such a property, the overall similarity should be drastically reduced much more than in case the property values are present but dissimilar. For the example, we now assume that a and b are equal with respect to sim_{p1} and sim_{p2} whereby the $p3$ is absent in b. Expressed in terms of our known similarity function $sim: A \times A \rightarrow [0,1]$ this would mean $sim_{p1}(a,b) = 1$, $sim_{p2}(a,b) = 1$ and $sim_{p3}(a,b) = 0$. Taking equal weights of 1 of for w_{p1} to w_{p3} the aggregated similarity score is $\frac{1+1+0}{3} = 0{,}66$. Yet, the result contradicts with the desire to drastically reduce the overall similarity score in case of the absence of an item's property. Also, using different weights does not solve the problem, as in case of presence of all properties, the equal weighting is desired and appropriate.

Such situations, which we will show to occur regularly in our context especially at the level of event sequence similarity, can be tackled in various ways. One could argue that linear similarity aggregation is not the best choice in general for combining the independent similarity aspects. Yet, the claim that the model is simple and intuitive and found an alternative approach to avoid above said shortcoming. The first possible solution is to apply a logarithmic adjustment function *before* the similarity aggregation, mapping similarity inversely to a distance value between 0 and $+\infty$. The alternative is to calculate based on a distance or cost model already up-front, and perform the mapping from distance to similarity inversely. In this model, we can easily overcome the problem of the previous example, by assigning the appropriate cost function $cost_{p3}$ any arbitrarily high value, for instance 100000. Accordingly, we would set $cost_{p1} = 0$ and $cost_{p2} = 0$ as we stated that a and b are equal with respect to $p1$ and $p2$. It is obvious that now a weighted average would still result in large total costs and subsequently in a very low similarity.

Due to these considerations, we apply a cost model for event sequence level similarities. For single event similarities we stick with known similarity function, as it is more intuitive and there is no need to perform a conversion of costs to similarity. Obviously now, in order to integrate the cost model for event sequence similarity with the similarity model for event-level similarities, a conversion of costs to similarity or vice-versa is required. Hereby, we follow the model of Shepard [45] who defined an exponential relation between a distance, or cost function and a similarity measure. Given a set of entities A and a similarity measure $sim: A \times A \rightarrow [0,1]$, a corresponding distance function $d: A \times A \rightarrow \mathbb{R}_0^+$ is defined as:

$$d(a,b) = -\ln sim(a,b)$$

Formula 9: Converting similarity to distance

Equivalently, given a distance measure $d: A \times A \rightarrow \mathbb{R}_0^+$ a corresponding similarity measure $sim: A \times A \rightarrow [0,1]$ is defined as follows:

$$sim(a,b) = e^{-d(a,b)}$$

Formula 10: Converting distance to similarity

4.2 Single event similarity

The assessment of similarity for considering the event attributes depends on the similarity technique applied for the similarity comparison. The following table lists different attribute similarity techniques applied in the course of this work. Entries marked with an asterisk are techniques which cannot be applied in an event-by-event comparison process. These techniques must be treated separately.

Attribute data type	Similarity technique
Numeric, Timestamp, Timespan	Normalized absolute difference similarity
	Relative difference similarity
	Normalized sequence similarity *
	Normalized relative sequence similarity *
	Lookup table similarity
String	String distance metric similarity
	Semantic similarity
	Lookup table similarity
Boolean	Boolean similarity
	Lookup table similarity
Multi-value types	Multi-value similarity
Nested events	Single event similarity
All	Attribute expression similarity
Unknown / Custom objects	Generic similarity

Table 1: Similarity techniques for event attributes

4.2.1 Normalized absolute difference similarity

Normalized absolute difference (NAD) similarity computes the relative distance of two values with respect to the overall value range of all considered items. Given an event type T_1 with an event attribute x, and X being the set of all attribute x' values extracted from the events of type T_1 in the searched data set, we define the NAD similarity measure as

$$sim_{NAD}(x_i, x_j) = \frac{|x_i - x_j|}{max(X) - min(X)}$$

Formula 11: Normalized absolute difference similarity

whereby i and j are fictive indices of two events to be compared in the total set of events and thus x_i is the value of the event attribute x for the event at index i, and x_j is the respective value for the event at index j.

This implies that the minimum and maximum occurring attribute value in the complete data set must be known up-front. The technique is applicable for continuous numeric attributes. In case of numeric attributes representing categories it might be misleading. A common example of numeric attributes which are not comparable by normalized absolute difference is an error code attribute. Here, similar values may have a completely different meaning.

4.2.2 Relative difference similarity

For cases where a relative similarity measure, independent from the complete data range, is more appropriate, we provide the *relative absolute difference (RAD) similarity*, which is calculated as

$$sim_{RAD}(x_i, x_j) = \frac{|x_i - x_j|}{max(x_i, x_j)}$$

Formula 12: Relative difference similarity

whereby again x_i and x_j denote the event attribute values of the attribute x for the events indexes i and j.

4.2.3 String distance metric similarity

Measuring similarity between strings is a research topic with a long history. Many approaches and measures have been published, which can be roughly divided into syntactic, phonetic and semantic approaches. Yet, a detailed discussion of these methods is out of scope for this thesis.

We have integrated a set of string similarity techniques, e. g. Levenstein distance, L2 distance, Jacard Similarity and many more available in the open-source similarity library SimMetrics, developed by Sam Chapman at Sheffield University [10].

4.2.4 Lookup table similarity

Despite of type-specific similarity measures, one of the simplest techniques for similarity is the use of lookup tables, where the user explicitly assigns similarity values to arbitrary value-pairs. From such a mapping, a similarity measure can then be derived: When comparing two attribute-values, the table is simply looked up and the corresponding similarity value is returned. The advantage is that highly purpose-specific, semantic similarities can be defined. For instance, Table 2 lists some similarities which could be set of a string attribute "sport type" in a "sports bet placed"-event.

Term 1	Term 2	Similarity
Rugby	American Football	1.0
Free throws	Penalty	0.7
Penalty	Direct free kick	0.2

Table 2: An exemplary similarity lookup table from the sports domain

Please note that from such a table we do *not* derive associate relations. For instance,

$$sim_{LT}(A, B) = 0.4, sim_{LT}(B, C) = 0.6 \nLeftrightarrow sim_{LT}(A, C) = 0.5$$

Formula 13: No association rules in lookup table similarity

with A, B and C being items to compare and sim_{LT} denoting a function looking up the items' similarities in the lookup table.

4.2.5 Boolean similarity

Given a Boolean event attribute x we define Boolean similarity as

$$sim(x_i, x_j) = \begin{cases} 1, & x_i = x_j \\ 0, & else \end{cases}$$

Formula 14: Boolean similarity

whereby x_i is the attribute value of attribute x of an event at the fictional index i, and x_j is the respective attribute value of a second event at index j.

4.2.6 Multi-value similarity

A multi-value type is defined as any ordered or unordered set of values of a given runtime type. The definition of a similarity measure for multi-value types must therefore consider the single values. Yet, the question remaining is which items to compare to each other. A simple example is the following: An order event could contain a list of products ordered. Let's say, the first event contains the list [A,B,C] with A, B, and C being product names, and the second event contains the list [D,E,F]. Now, even if we know that products A and E are similar (for instance with a lookup table), how to we know which items to compare? An n-to-n comparison could lead to significant performance problems. Therefore we propose, referring to Sjoberg's feature-based similarity-model [46] to compute multi-value similarity based on common items in the two value sets in relation to total items as

$$sim(x_i, x_j) = \frac{count(X_i \cap X_j)}{count(X_i \cup X_j)}$$

Formula 15: Multi-value similarity

whereby x_i is the attribute value of attribute x of an event at the fictional index i (in this case x_i is a container object for a typed value set), and x_j is the respective attribute value of a second event at index j. X_i and X_j are the sets of values contained in x_i and x_j and $Count(A)$ denotes a function returning the number of items in a set A.

4.2.7 Nested event similarity

Besides runtime types and multi-value types, attributes can also be of another event type (see section 1.3.1). Even multi-value types may again contain a set of other events. For instance, an alert notification event may contain an incoming error event which triggered the alert. Obviously such events may be important to consider in certain business cases.

We define similarity for nested events recursively as

$$sim_{ET}(x_i, x_j) = \frac{\sum_{k=1}^{n} \left(sim\left(x_{i_k}, x_{j_k} \right) * w_k \right)}{\sum_{k=1}^{n} w_x}$$

$$sim(a,b) = \begin{cases} sim_{ET}(a,b), & typeof(a), typeof(b) \in Event\ Types \\ sim_{*}\ (a,b), & else \end{cases}$$

Formula 16: Recursive similarity definition for nested events

with x_i and x_j being event attributes of an attribute x with the attribute type "event type", x_{i_1} to x_{i_n} the attributes of the nested event, n the number of attributes in the nested event object and w_1 to w_n the user-configured weights for the attributes of the nested event types. In addition, $sim_{*}(a,b)$ refers to all other attribute similarity techniques except of nested event similarity described and selected for the concerned attributes.

4.2.8 Attribute expression similarity

Above described similarity measures are defined to always compare single event attributes. In certain cases though, it might be of interest to consider a compound value of multiple attributes. For instance, when having an attribute "start time" and a second attribute "end time", the compound value "duration", derived by computing end time minus start time could be of interest.
We therefore introduce the attribute expression similarity. It allows evaluating an arbitrary *EventAccess (EA) expression* which returns a typed value computed based on the event. Depending on the return type of the EA expression, one of the above named similarities can subsequently be applied.

4.2.9 Generic similarity

Our similarity model foresees the integration of custom similarity measure implementations. The framework for generic similarity basically allows their integration into the matching process. The purpose is mainly to keep an open, extendible and customizable character of our event processing platform. Details on the integration of custom similarity measures are provided in section 5.4.

4.2.10 Event level constraints

4.2.10.1 Attribute constraints

Attribute constraints are set for single event attributes in the pattern sequence. In Figure 12 an example of a numeric attribute constraint is given. Attribute constraints are supported for numeric values and string attributes, limiting the set of allowed attribute values to a given range or a list of allowed values.

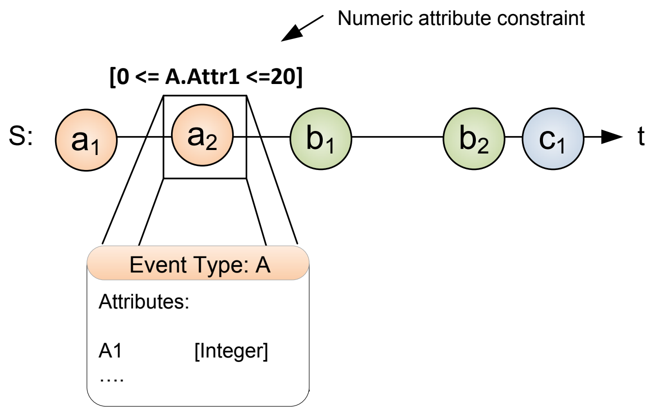


Figure 12: Numeric attribute constraint

In the computation model, attribute constraints are evaluated in companion with attribute similarities. If the constraint evaluation fails, the matching process stops and the specific match is omitted.

4.3 Event sequence similarity

4.3.1 Overview and definitions

Throughout the following sections, various similarity features are discussed concerning sequences of events. We define an event sequence S of length n formally as an ordered sequence of n events e_1 to e_n where $\mathrm{v}_t(e_i) < \mathrm{v}_t(e_{i+1})\ \forall i = 1, \dots, n$, with $v_t(e)$ representing the occurrence time of an event e.

As already mentioned above, in order to compute the similarity of two event sequences, we define a cost-based computation model. This cost model describes the costs for a possible solution, i.e. one combination of *mappings* from events of the searched sequence to events in the reference sequence. Formally, we define a *solution* as a function $s: S_p \rightarrow S_t \cup \{\varepsilon\}$. Hereby, S_p is the reference or pattern sequence and S_t denotes the target, or searched sequence. We further define ε as a null-node, or missing event, an event that is virtually inserted into a solution if for a mapping no respective event is available in S_t. Formally, we define a mapping as a pair of events $(e, s(e))$, $e \in S_p$, $s(e) \in S_t \cup \{\varepsilon\}$. if $s(e) = \varepsilon$, it is considered a *null-mapping*, else we refer to it as a *normal mapping*.

In the following, all cost factors for the similarity are conceptually summarized. Details on the cost computation are provided in companion with the algorithmic implementation in section 5.

4.3.2 Event type occurrence

The first factor to consider for the assessment of similarity between a reference event sequence and the compared event sequence is the occurrence of event types.

In terms of event type occurrence we define:

(1) *Full event sequence equality* of two event sequences in terms of event type occurrence is given, if a solution $s: S_p \rightarrow S_t$ (without ε!) exists so that $pos(e, S_t) = pos(s(e), S_p)\ \forall s \in S_t$ and $n_{S_p} = n_{S_t}$, i.e., for each event in the pattern sequence S_p a corresponding mapping can be found in S_t at the same event position and the two sequences are of equal length. In Figure 13, events of different event types, denoted by characters a to c are illustrated on a time axis according to their occurrence time t. In terms of event type occurrence, these sequences are equal.

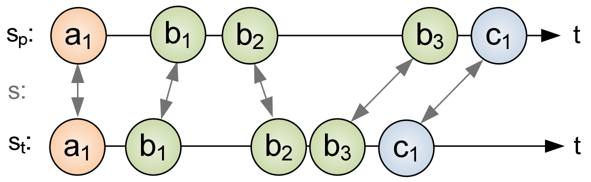


Figure 13: Full event sequence equality in terms of event type occurrence

(2) *Subsequence equality* of two event sequences in terms of event type occurrence is given, if a solution $s: S_p \rightarrow S_t$ (without ε!) exists so that $pos(e, S_t) = pos(s(e), S_p)\ \forall s \in S_t$. The only difference to full

event sequence equality is that other events, which are not mapped may precede or follow the events in the mapping as illustrated in Figure 14.

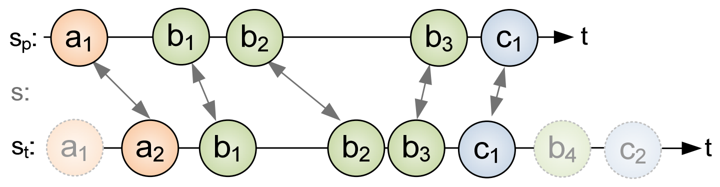


Figure 14: Subsequence equality in terms of event type occurrence

(3) One or several mapping pairs of events may be in incorrect order. An incorrect order increases costs (and thereby decreases the similarity score) proportionally to the number of events between the desired position (as it is in the pattern sequence) and the actual position (as it is in the target sequence). Please note that, unlike Figure 15 might suggest the pure type sequence deviation solely considers the type order, but not the occurrence time of the events. Up to this point, the model is comparable to edit-distance approaches. However, later extensions will underline that a pure edit distance model is not applicable for all considered similarity features.

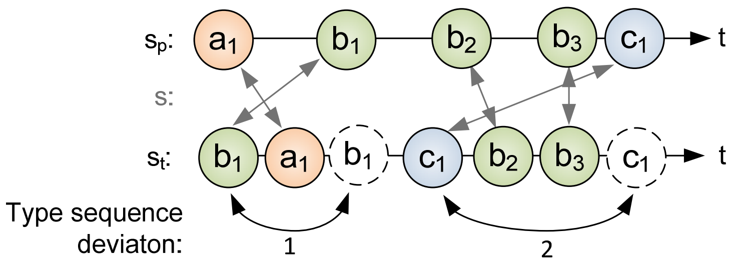


Figure 15: Event type sequence deviations

(4) One or several events might be present in S_t which are not considered for a solution as no corresponding event is present in the pattern sequence S_p. We refer to these events as *redundant events*. The costs of a solution are increased proportionally to the number redundant events.

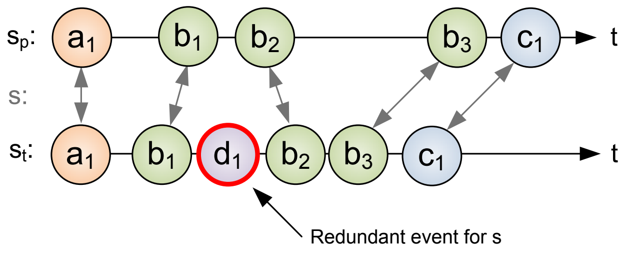


Figure 16: Redundant events

(5) If for an event in the pattern sequence no suitable corresponding event in the searched sequence can be found to build a mapping, a virtual event must be inserted. We have defined this as a *null-mapping*. These mappings cause additional costs and decrease the similarity score by a user-configurable factor.

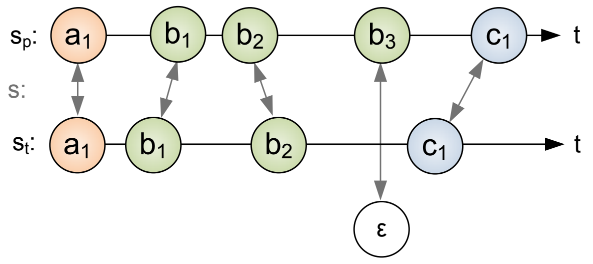


Figure 17: Null-mapping

4.3.3 Occurrence times of events

In the previous section, order deviations between an event in the target sequence and the corresponding event in the pattern sequence for a given solution s have been discussed independent from the actual occurrence times of the events despite the fact that order also in that case referred to the temporal order.
In the following we refer to a function $t(e_i, e_j)$ as the *time span between the occurrence times of two events* e_i *and* e_j. Based on the so-calculated time spans we define 2 modes for computing deviation costs.

4.3.3.1 Absolute time spans

In the absolute time span mode, the absolute difference between $t(e_i, e_j)$ and $t(s(e_i), s(e_j))$ is used to compute similarity costs. This Figure 18 depicts these time spans. In practice, the absolute time difference will be adjusted by a however defined function or constant in order to align resulting costs with other cost factors.

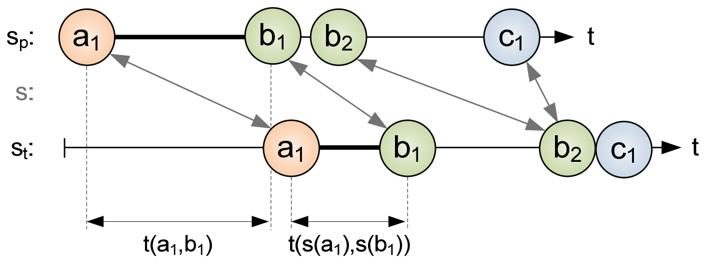


Figure 18: Absolute deviations in events' occurrence times

4.3.3.2 Relative time spans

The absolute time span mode implicitly results in the "expectation" that the length of the target sequence corresponds in absolute values to the length of the pattern sequence. Yet, in many cases the absolute values are not relevant. In contrast, the time gaps *within* the sequence are decisive. We introduce the relative time span mode to cover this case. It sets the time span between two events in the pattern sequence $t(e_i, e_j)$ in relation to the total time span the sequence is covering, denoted as (S_p) . Equivalently, the time span between two subsequent mappings $t(s(e_i), s(e_j))$ in a solution s is set in relation to the total time span between the first and last event in the mapping $l(s)$.

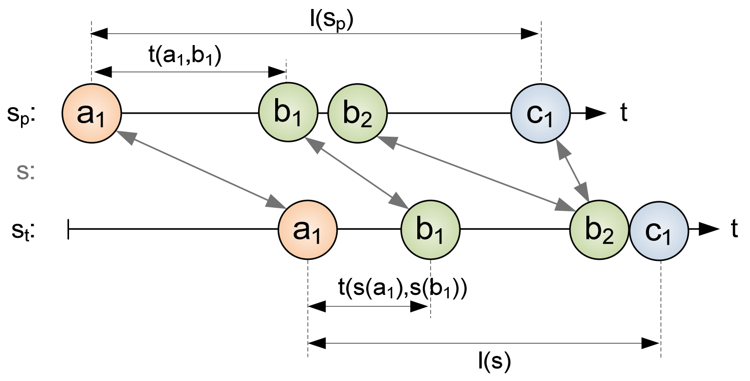


Figure 19: Absolute deviations in events' occurrence times

Costs are computed based on the absolute difference between these time span ratios. This means in particular, that a sequence of events can be relatively stretched or jarred without decreasing the similarity score.

4.3.4 Numeric sequence similarity

Numeric sequence similarity and relative numeric sequence similarity are special cases of attribute similarities which cannot be evaluated on an event-by-event level. Here, the complete sequence of attribute values must be extracted first and compared separately. The resulting similarity is then one additional factor like for instance the result of the type similarity comparison. Further details on the applied time-series similarity model for numeric sequence similarity can be found in section 0.

4.3.5 Event sequence level constraints blocks

Sequence level constraints concern the occurrence of a single event or set of event within the event sequence or in relation to each other (e.g. the order). We distinguish restrictive and broadening blocks. Restrictive blocks are limiting the set of possible solutions by certain constraints, e.g. constraints on occurrence times of events or order constraints. Broadening blocks "weaken" the similarity assessment by allowing more possible solutions. For instance, a block allowing a subset of events to occur in arbitrary order without decreasing the similarity score is counted as a broadening block.

4.3.5.1 Restrictive blocks

4.3.5.1.1 Required block

A "required"-block indicates that for all solutions $s: S_p \rightarrow S_t \cup \{\varepsilon\}$, the comprised pattern-sequence events must have a counterpart in the target-sequence, i.e., for each event $e \in S_p$ that is part of a "required"-block, $s(e) \neq \varepsilon$ must hold.

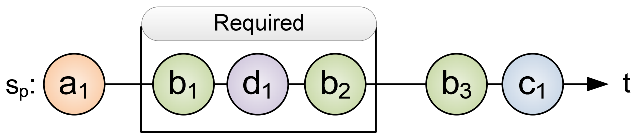


Figure 20: Required block

4.3.5.1.2 Time of occurrence constraints

A “time of occurrence“-constraint block indicates that for all solutions $s: S_p \rightarrow S_t \cup \{\varepsilon\}$, the comprised pattern-sequence events must be mapped to target-sequence events who’s times of occurrence are inside a certain, user-specified time interval, as indicated in Figure 21.

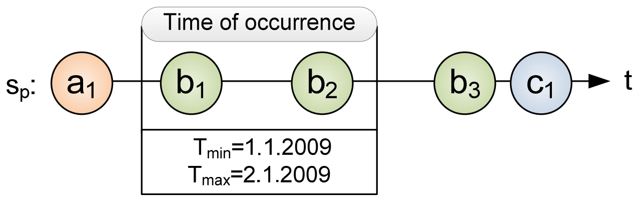


Figure 21: Time of occurrence constraint block

This means in particular, that the block checks whether the events in the target sequence occur at respective points in time or not, but it does *not* increase or decrease the similarity score.

4.3.5.1.3 Maximal time span constraints

A “maximal time span“-constraint block indicates that for all solutions $s: S_p \rightarrow S_t \cup \{\varepsilon\}$, the comprised pattern-sequence events are mapped to target-sequence events so that the time span between the earliest and the latest target-sequence event is smaller than a user-defined time span m. Before giving a more formal description, let us define the concept of the *maximal time span in a set of events*:

Definition: Given a set of events $E \subseteq \mathbb{E}$, with e_i addressing the i^{th} event in E and $|E|$ addressing the number of events in E, we refer to the result of a function $t: \mathbb{E}^* \rightarrow \mathbb{R}_+$ with $t(E) = \max_{1 \le i \le |E|} v_t(e_i) - \min_{1 \le j \le |E|} v_t(e_j)$ as the *maximal time span* in E.

Thus, given a “maximal time span”-block $M \subseteq S_p$ and a maximal time span m, $t(M_{S_t}) < m$ with $M_{S_t} = (s(e) | e \in M)$ must hold for all solutions $s: S_p \rightarrow S_t \cup \{\varepsilon\}$.

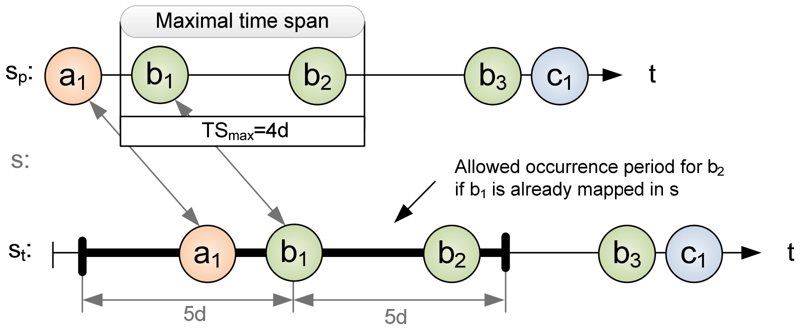


Figure 22: Maximal time span constraint

A violation of the maximal time span constraint leads to omitting the possible match.

4.3.5.1.4 Minimal time span constraints

A “minimal time span“-block can be considered the opposite of a “maximal time span” block: It indicates that for all solutions $s: S_p \rightarrow S_t \cup \{\varepsilon\}$, the comprised pattern-sequence events are mapped to target-sequence events with a time span greater than a user-defined, minimal time span. More formally, given a “minimal time span”-block $M \subseteq S_p$ and a minimal time span m, $t(M_{S_t}) > m$ with $M_{S_t} = (s(e) | e \in M)$ must hold for all solutions $s: S_p \rightarrow S_t \cup \{\varepsilon\}$.

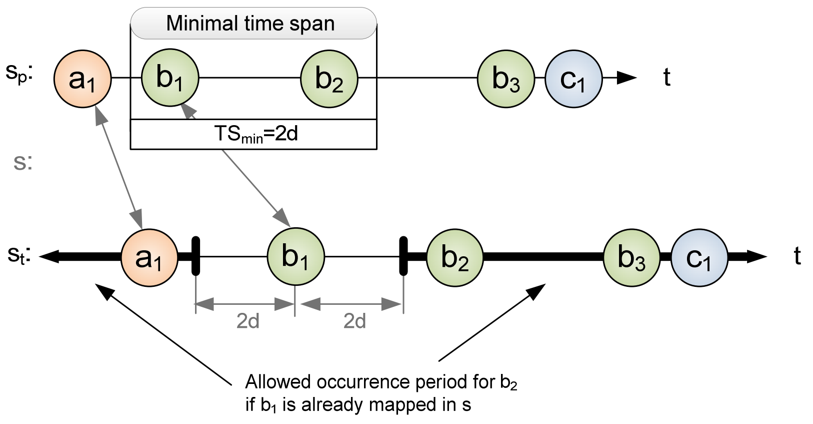


Figure 23: Minimal time span constraint

4.3.5.1.5 Strict order constraint block

A "strict order"-constraint block indicates that for all solutions $s: S_p \rightarrow S_t \cup \{\varepsilon\}$, the comprised pattern-sequence events must be in the correct order in s, i.e., for each pair of (successive) events e and f of a "strict order"-block $O \subseteq S_t,\ e, f \in O,\ pos(e, S_t) > pos(f, S_t) \rightarrow pos(e, s) > pos(f, s)$ (or, equivalently, $v_t(e) > v_t(f) \rightarrow t(e, s) > t(f, s)$), most hold.

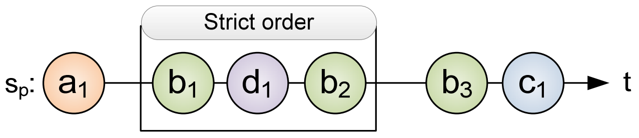


Figure 24: Strict order constraint block

4.3.5.2 Widening blocks

4.3.5.2.1 Arbitrary order block

An "arbitrary order"-block $A \subseteq S_p$ indicates that when calculating the overall costs of a target-sequence S_t, not only all "normal" solutions $S_p \rightarrow S_t$ shall be taken into account, but also all solutions for the so-called *temporal permutations* of S_p with respect to A.

In the following, the concept of *temporal permutations* is clarified by a simple example: Consider a sequence S_p with a "arbitrary order"-block A as shown in Figure 25.

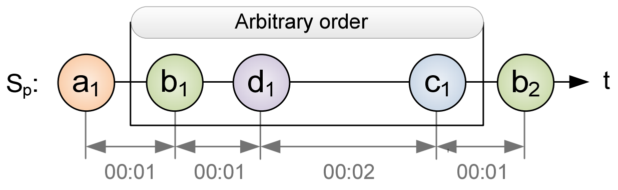


Figure 25: Arbitrary order block

The *temporal permutations* of S_p can now be considered event sequences that are, in most respects, equal to S_p but contain different permutations of A; yet retaining the original set of time stamps. Figure 26 shows all permutations of S_p (including S_p itself) with respect to A:

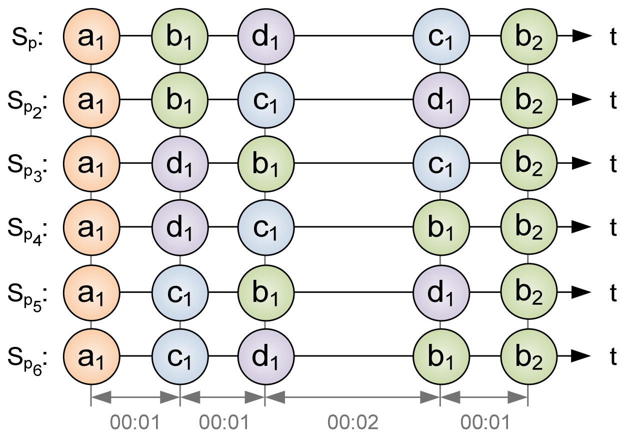


Figure 26: Temporal permutations in an arbitrary order block

Thus, a *temporal permutation* of an event sequence S_p with respect to a sub-sequence $A \subseteq S_p$ is an event sequence $S_p{}'$ where the times of occurrence (and, consequently, the positions in $S_p{}'$) are permutated for all events in A. All other event attributes remain equal across the events in S_p and $S_p{}'$.

4.3.5.2.2 Occurrence number blocks

An "occurrence number"-block $\subseteq S_p$, defining a minimal occurrence of min and a maximal occurrence of max, indicates that when calculating the overall costs of a target-sequence S_t, not only all "normal" solutions $S_p \to S_t$ shall be taken into account, but also all solutions for the so-called *foldings* of S_p with respect to O.
Again, let us clarify the concept of *foldings* in a simple example. Note that at this point, we do not take the exact times of occurrence into account; we will deal with this issue in next section.
Example: Consider a sequence S with an "occurrence number"-block O as shown below in Figure 27.

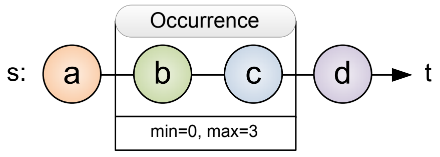


Figure 27: Example for an occurrence number block

For $n > 1$, the *n-folding* S_n of S can now be considered an adapted version of S with the events in O appearing n times, one "block" following the other.[4] For $n = 0$, S_n does not contain the events in O at all. For $n = 1$, $S_n = S$. Below, we list all foldings S_i of S with $min \leq i \leq max$.

[4] Given a sequence S and an "occurrence number"-block O, we refer to the i^{th} appearance of a block O in a folding S_n, $n \leq i$, of S as the i^{th} iteration of O in S_n.

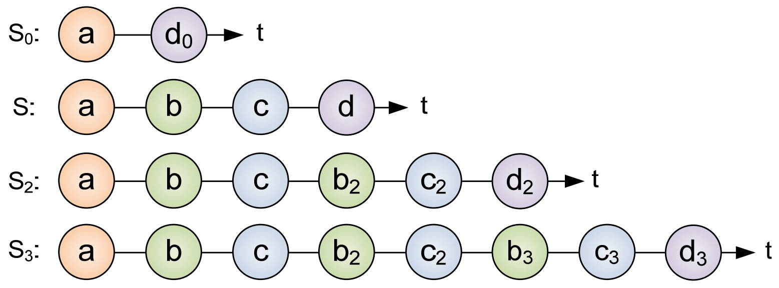


Figure 28: Foldings for a simple occurrence number block

Here, d_0, d_2 and d_3 can be considered as "shifted" clones of d, i.e., d_0, d_2 and d_3 equal to d regarding all event-attributes but the time of occurrence. Consequently, b_2 and b_3, and c_2 and c_3, can be considered shifted clones of b and c, respectively.

Temporal structure

It is easy to see that the *order* of events in a folding S_n is defined. This is not the case, however, for the exact temporal structure. Consider, for instance, S_3 as shown above. Here, the following time spans between events derive naturally from the base sequence S:

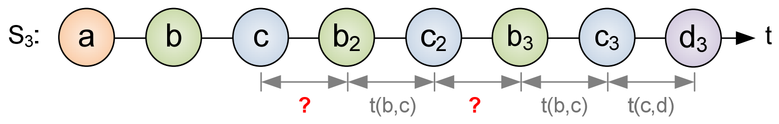


Figure 29: Temporal structure problem for folding in case of occurrence number blocks

The time spans between c and b_2 and between c_2 and b_3, i.e., the "borders" between successive iterations, are still to be defined, though. Also, for a zero-folding, the time-span between the event that precedes the (not existing) block and the event that succeeds the (not existing) block (in the above example, these are a and d_0), is to be defined.

We deal with this issue by letting the analyst define this time span, i.e., a time span between the latest and the earliest event of a block O and, if a minimum occurrence of zero was chosen, a time span between the events "surrounding" O.

4.3.5.2.3 Arbitrary events

Arbitrary events are events of the predefined event type *Arbitrary* which does not declare any event attributes (except of the event header with event id and time stamp) and cannot occur in the operational business environment. Instead, they are used as tools for enhanced similarity searching: As part of a pattern sequence, arbitrary events are considered compatible to all events of any given target-sequence. We depict arbitrary events with a diamond shape and question mark inside as illustrated in Figure 30. Also, we will refer to the overall set of arbitrary events as $\mathbb{X}$.

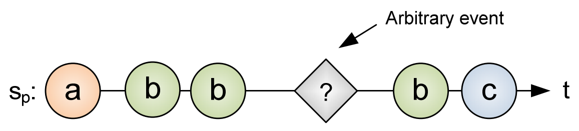


Figure 30: Illustration of arbitrary events

Arbitrary events can only be created “artificially”, i.e., defined by the business analyst. With a certain, user-defined “time of occurrence”, an arbitrary can then be inserted into a given pattern-sequence. Therewith, different solutions are considered valid, which may affect the overall costs of a target-sequence. Note, however, that for mappings to arbitrary events all attribute similarities are omitted. We will show in the implementation section, that the therewith left unconsidered cost-factors require an adaption of the cost model in terms of computing a correct weighted average by omitting these factors.

5 Similarity computation

In the previous section the similarity assessment model has been presented from a general viewpoint, independent from any algorithmic considerations. In this section, we propose how to apply this model by introducing our algorithmic models for basic event sequence similarity, event sequence constraints and time-series similarity for event attributes.

5.1 The base algorithm

In his thesis, Obweger [37] presents a base algorithm for evaluating event sequence similarity which has been designed in collaboration. Here, the cornerstones of this base algorithm are summed up in order to understand subsequent deliberations especially on event sequence constraint.

5.1.1 Finding the best solution: an assignment-based approach

In section 4.3.1 we have introduced the term *solution* as a function $s: S_p \to S_t \cup \{\varepsilon\}$ with S_p being the pattern sequence and S_t the target sequence. Thus, implicitly we have already introduced an assignment-based approach towards similarity: A *solution* maps events from the pattern to events in the target sequence or assigns them as missing (null mapping). Depending on all similarity factors and constraint blocks presented above, each solution has a certain quality. The similarity assessment model defined how to compute this quality. Yet, the remaining challenge is how to efficiently discover the solution with the best quality. Mathematically, a huge number of possible solutions exist. This number can be computed based on the length of the pattern and target sequences as:

$$s = \sum_{k=0}^{min(|S_p|,|S_t|)} \left(\frac{(|S_p|)!}{(|S_p| - k)!} \right) * \binom{|S_t|}{k}$$

Formula 17: Theoretical number of solutions for matching two event sequences

For instance, for a pattern sequence with $|S_p| = 10$ and a target sequence with $|S_t| = 12$ in total 2,581,284,541 solutions exist.

Luckily, some natural limitations exist for the set of solutions. For instance, not each event can be mapped to each other event. Intuitively, events of different event types are not *compatible* to each other. Formally, we define *compatibility* as function $comp: S_p \times (\mathbb{E} \cup \{\varepsilon\}) \to \{0,1\}$. Two entities e and f, $e \in S_p$, $e \in \mathbb{E} \cup \{\varepsilon\}$ with $comp(e, f) = 1$ we refer to as *compatible* with respect to $comp$. Otherwise, if $comp(e, f) = 0$, we refer to e and f as *incompatible* with respect to $comp$.

5.1.2 Implementation model

The base algorithm for finding similar event sequences can be counted to the family of *Branch & Bound* algorithms. Using a tree-based structure, valid solutions are discovered with respect to a given compatibility. A dynamic threshold helps to reduce the number of investigated solutions and identify the best solution as fast as possible.

The tree structure is build up incrementally: To the tree's root node first a set of child-nodes is added representing all matches for the first event of the pattern sequence. To each of these nodes, all matches for the second event in the pattern sequence are added and so forth. Yet, it is important to notice that we build the tree in a depth-first fashion.

5.1.2.1 Building the solutions tree

Let us consider an example to clarify the approach. In the following, we assume to have a pattern sequence S_p and a target sequence S_t as depicted in Figure 31.

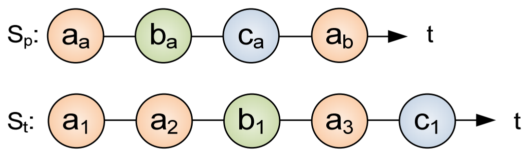


Figure 31: Exemplary pattern and target sequence

Furthermore, for the example we will define a simple compatibility which assumes that only events of the same type are compatible to each other and we omit null-mappings for a moment. Based on these assumptions the dynamic tree can now be build. Adding to a virtual start node N_s the first compatible event from S_t to build a mapping, a_1, is added to the tree. As the tree is built up depth-first, this process continues to the last event in the pattern sequence a_b. The contemporary result is illustrated in Figure 32.

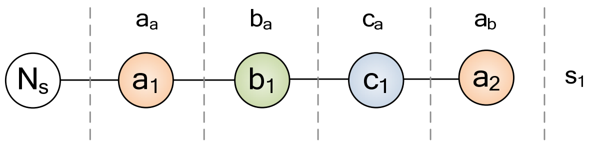


Figure 32: First possible solution in the dynamic tree

In this branch, the alternative mapping for a_b is a_3 instead of a_2. Thus, this node is added to the tree, as shown below. Please note that no node can be "reused" in a branch. Intuitively, we assume that one event cannot be counted several times within the same solution. Therefore, $a_1$is not added as an additional leaf node to this branch.

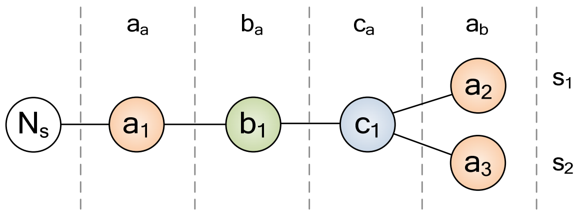


Figure 33: Continuing to build the tree of solutions

Continuing the process, a full tree of possible solutions will be build up. In the given case, 6 possible solutions exist, each depicted in the tree by a branch from the start node to a leaf node.

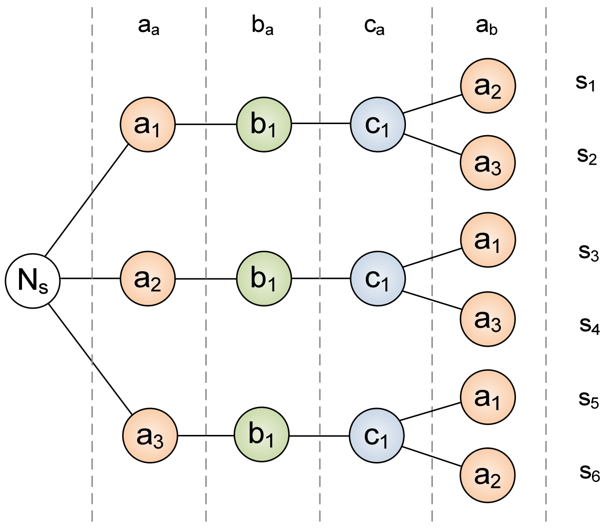


Figure 34: Full tree of solutions

For this small example and the assumption that null-mappings are not allowed and thus every event must be present in the target sequence, the number of solutions is still manageable. Yet, allowing null-mappings already increases the number of solutions to 52. In such a case, for each mapping it is possible to either use one of the target-sequence events or insert a special node, the so-called null-node into the tree. Figure 35 shows a subset of the resulting tree. Named null-nodes are shaded in light-grey.

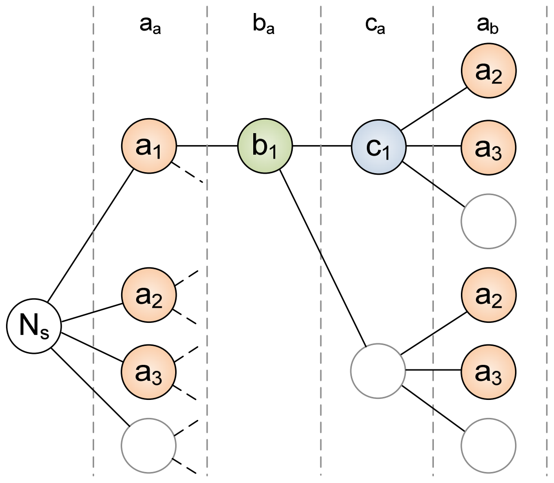


Figure 35: Excerpt of the solutions tree in case of null-mappings

5.1.2.2 The dynamic threshold

The simple example given in the previous section demonstrated that the solutions tree grows huge in case of longer event sequences. Thus it is crucial to limit a branch as early as possible. In section 4 we presented, based on which features costs for a solution are computed. Building up the tree in a depth-first manner allows us to compute the first costs c_{s_1}immediately when reaching the first leaf node. From this point only solutions need to be considered with total costs below c_{s_1}. If a new solution is found with still lower costs, these costs make up the new threshold.

An example is provided in Figure 36. For the sake of simplicity, the applied cost model only considers type deviations, and a deviation in terms of event position counts 1 per event that has to be “jumped over”. As can be seen from the figure, the first 4 matches have to be build up to the last event. Though s_3 exceeds the prior dynamic threshold of $c_{s_2} = 3$ this is only clear after adding the last mapping. In the last branch, after adding a_3 and b_1 costs are already higher than the dynamic threshold and the rest of this branch may be omitted.

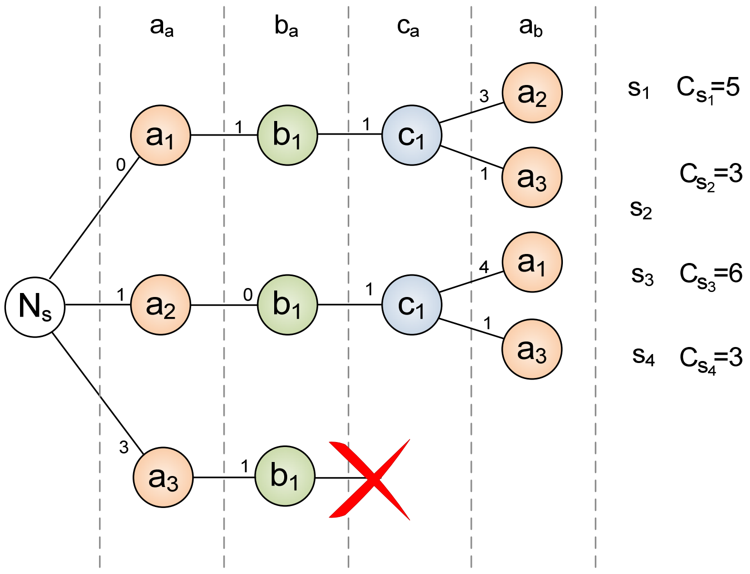


Figure 36: Threshold example

In case of assigning high costs to null-mappings and considering also event-level similarities it is obvious that this dynamic threshold omits a huge set of possible, but bad solutions.

5.2 Enhanced search pattern building blocks

5.2.1 Integration into the base algorithm

In order to integrate the additional search pattern building blocks into the base algorithm, we first designed a generic structure for their integration. Each block is represented by a block object, holding a separate state during the matching process. The block implementation as such is responsible for computing the costs for mappings, if an event is contained in the block. Basically, the interface each block has to implement provides the following operations:

- *AddMapping()* – This function is called for each object in the block and returns a *BlockResult* wrapping the costs. For instance, an arbitrary order block would return only the single event similarity costs but omit the order deviation costs. In addition, the *BlockResult* object returns allowed indices for the subsequent mappings. These indices indicate which mappings are valid for the next event. This is decisive in case of the occurrence number block, which enables to jump over certain events. Restrictive blocks return *null* if the mapping is not allowed, for instance in case of an order deviation within a strict order block.
- RemoveMapping() – This function is called when stepping out of a recursion. For performance reasons, blocks are not totally recalculated for every branch of the solutions tree, but adjusted

dynamically. Below, we will see that most blocks internally use a stack structure to dynamically add and remove the mappings.

- *SetSucceedingMapping()* – This function is called when the first mapping after a block is build. It is required for blocks which can in part be computed only after all events of the block are present, for instance in case of the occurrence number block.

Algorithm 1: Integration of search pattern building blocks into the base algorithm

Input: TreeNode parent (the parent node), int index (the current level of the tree)

Output: -

Variables: *i, j*, indices; *prevPEvent*, the previous pattern-sequence event; *pEvent*, the current pattern-sequence event; *prevBlock*, the block instance surrounding *prevPEvent*; *block*, the block instance surrounding *pEvent*; *match*, a match at the current level of the tree; *child*, a tree node representing the current match. *indices*, the pattern-sequence indices to continue the tree with; *weightedCosts*, the costs as calculated from the current and the previous mapping; *cont*, a flag indicating whether the current path can be continued or has to be aborted due to constraint violations.

State: *pattern,* a field of events representing the pattern sequence; *matches,* a field containing sets of matches in the order of the corresponding pattern-sequence events; *threshold,* the current threshold, initialized with a used-defined value $t_{initial}$.

```
// Get current pattern event and previous pattern event
Event prevPEvent = pattern[index - 1];
Event pEvent = pattern[index];

// Get block instance for current pattern event . Can be null if the event is not surrounded by a block
ConstraintBlock block = blocks[index];

// Get block instance for previous pattern event if different instance than "block"
ConstraintBlock  prevBlock = blocks[index – 1];
if  ( prevBlock = block) then
        prevBlock = null;
end

// Iterate through the matches for the corresponding pattern-sequence events
for i = 1 to matches[index].length step 1
        Event match = matches[index][i];

        // Check whether match is already part of the so-far path
        if ((match ≠ $\varepsilon$) and (parent.IsInPathToRoot(match)))  then
                continue;
        end

        // Calculate costs via the previous block, the current block
        // or as in the original base algorithm if the event is not part of a block
        int[] indices = new int[] { index + 1 };
        Double weightedCosts = null;
        bool cont = true;

        // If prevBlock is set, call SetSucceedingMapping…
        if (prevBlock ≠ null) then
                weightedCosts = prevBlock.SetSucceedingMapping(
                                        prevPEvent, parent.Match, pEvent, match, index);
        end
```

```

    // If prevBlock is null or prevBlock.SetSucceedingMapping returned null…
    if (weightedCosts = null) then
        // If inside a constraint block, call AddMapping(…), use result for further calculations
        if (block ≠ null) then
            BlockResult blockResult = block.AddMapping(
                prevPEvent, parent.Match, pEvent, match, index);

            cont = (blockResult ≠ null);
            if (cont) then
                weightedCosts = blockResult.Costs;
                indices = blockResult.Indices;
            end
        // Otherwise, if block is null, calculate costs as usual
        else
            weightedCosts = CalculateCostsAsUsual(
                prevPEvent, parent.Match, pEvent, match, index);
        end
    else
        // Call AddMapping and set “cont” and “indices”, but ignore costs since they are
        // overruled by prevBlock .SetSuccedingMapping
        BlockResult blockResult = block.AddMapping(
                prevPEvent, parent.Match, pEvent, match, index);

        cont = (blockResult ≠ null);
        if (cont) then
            indices = blockResult.Indices;
        end
    end

    // Check whether so-far costs are below the current threshold
    if (cont) and (parent.Sum + weightedCosts < threshold) then
        // Create child node and add to parent
        TreeNode child = new TreeNode(match);
        parent.add(child);
        // Set costs calcuated up to this point to child node
        child.Sum = parent.Sum + weightedCosts;

        // Do recursive method call or set threshold if a leaf is reached
        if (index < matches.length) then
            for j = 1 to indices.length step 1
                CreateSolutionsTree(child, indices[j]);
            end
        else
            threshold = child.Sum;
        end
    end

    // If inside block, call RemoveMapping
    if (block ≠ null) then
        block.RemoveMapping(pEvent, match, index);
    end
end
```

5.2.2 Restrictive blocks

5.2.2.1 Attribute constraints

In section 4.2.10.1 attribute constraints have been introduced as conditions required to be fulfilled in order to form a valid solution. Hence, it is not sufficient to set the event similarity to zero in case of an un-fulfilled attribute constraint. Instead, it must be guaranteed that the complete solution is omitted.

This can be achieved by introducing an extended compatibility which guarantees each attribute constraint to be fulfilled. Given a pattern-sequence S_p, a set of attribute constraints $attr_{\mathbb{T}_1}, \dots, attr_{\mathbb{T}_n}$ with $attr_{\mathbb{T}_i}: \mathbb{T} \to \{0,1\}\ \forall\, i = 1 \dots n$ on events of type $\mathbb{T}$ and a compatibility $c: S_p \times (\mathbb{E} \cup \{\varepsilon\}) \to \{0,1\}$, we define an adapted version $c': S_p \times (\mathbb{E} \cup \{\varepsilon\}) \to \{0,1\}$ of c as follows:

$$c'(e,f) = \begin{cases} c(e,f), & f \notin \mathbb{T} \\ c(e,f) * \prod_{i=1}^{n} attr_{\mathbb{T}_i}(f), & f \in \mathbb{T} \end{cases}$$

Formula 18: Extended compatibility function for attribute constraints

This means, plainly spoken, that an event is only compatible for a mapping, if all attribute constraints are fulfilled.

5.2.2.2 Required block

Given a pattern-sequence S_p, a “required”-block $R \subseteq S_p$ and a compatibility $c: S_p \times (\mathbb{E} \cup \{\varepsilon\}) \to \{0,1\}$, we define an adapted version $c': S_p \times (\mathbb{E} \cup \{\varepsilon\}) \to \{0,1\}$ of c as follows:

$$c'(e,f) = \begin{cases} 0, & e \in R \wedge f = \varepsilon \\ c(e,f), & otherwise \end{cases}$$

Formula 19: Extended compatibility function for “required”-blocks

Thus, when using c' instead of c, null-mappings are considered invalid for all those events that are part of the “required”-block.

Example. Consider two event sequences S_p and S_t and a “required”-block as shown below:

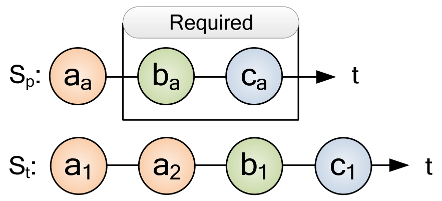


Figure 37: Example for a “required” block

Given above defined, adapted compatibility c' the following solutions of S_t are considered valid:

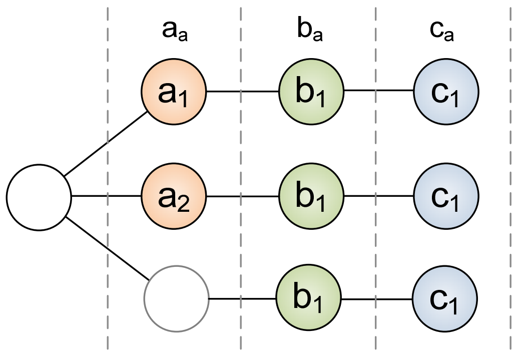


Figure 38: Tree of valid solutions in case of a “required”-block

As a “required” block has no effect on the calculation of a solution similarity, but instead only excludes certain solutions, we omit the similarity computation n the given example.

5.2.2.3 Time of occurrence constraints

Given a pattern-sequence S_p, a “time of occurrence”-block $O \subseteq S_p$ with a time interval T reaching from T_{min} to T_{max} as well as and a compatibility $c: S_p \times (\mathbb{E} \cup \{\varepsilon\}) \rightarrow \{0,1\}$, we define an adapted version c' of c as follows:

$$c'(e,f) = \begin{cases} 0, & (e \in O) \wedge (v_t(f) \notin T) \\ c(e,f), & otherwise \end{cases}$$

Formula 20: Extended compatibility function for “time of occurrence”-blocks

Thus, when using c' instead of c, all mappings between pattern-sequence events in O and target-sequence events “outside” T are considered invalid.

Example. Consider two event sequences S_p and S_t and a “time of occurrence”-block as shown below:

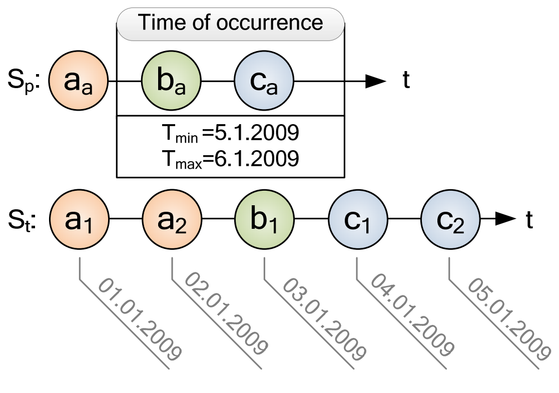


Figure 39: Example for a “time of occurrence” block

In the given example, b_1 and c_1 are outside of T. Therefore, with a base event-type compatibility c with $c(e,f) = 1$ for all $f = \varepsilon$, the following solutions of S_t are considered valid:

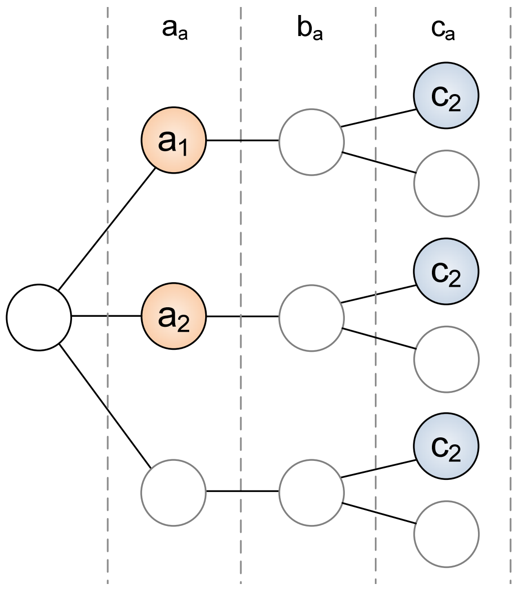


Figure 40: Tree of valid solutions in case of a "time of occurrence"-block

5.2.2.4 Maximal time span constraints

Given a solution $s: S_p \rightarrow S_t \cup \{\varepsilon\}$ and a "maximal time span"-block $M \subseteq S_p$ with a maximal time span of ts_{max}, let us define a *maximal time-span function* $maxTSpan: {S_p}^* \rightarrow \{0,1\}$ with

$$maxTSpan(e_1, e_2, \ldots, e_n) = \begin{cases} 1, & t(\{s(e_1), s(e_2), \ldots, s(e_n)\}) < ts_{max} \\ 0, & otherwise \end{cases}$$

Formula 21: Maximal time span function

We now consider s valid only if $maxTSpan$ results in 1 for all events in M.

Note that an evaluation of whether the maximal time span in a set of target-sequence events exceeds the given threshold ts_{max} is valuable each time a new target-sequence event gets known: From $t\left({M_{S_t}}'\right) > ts_{max}$ it follows that $t\left(M_{S_t}\right) > ts_{max}$ for each ${M_{S_t}}' \subseteq M_{S_t}$. We therefore integrate the "maximal time span"-block as follows into the base algorithm: When adding a node that represents a mapping for the nth event in a "maximal time span"-block M, i.e., a mapping $\left(e, s(e)\right)$ with $e \in M$ and $pos(e, M) = 1$, we evaluate $maxTSpan$ for the node and its $n-1$ predecessors. If $maxTSpan$ returns 1, the recursive algorithm is continued; otherwise, if $maxTSpan$ results in 0, the algorithm is cancelled for the concerned path.

In pseudo-code, an efficient implementation of a "maximal time span"-block can be described as follows:

Algorithm 2: Processing of maximal time span constraints - *AddMapping()*

Input: Event prevPatternEvent, Event prevMatch, Event patternEvent, Event match, int index

Output: A *BlockResult* object if the mapping *(patternEvent, match)* is valid with respect to the given block *M*, null otherwise.

Variables: Pairs of time stamps (i.e. temporal ranges) *lastRange* and *newRange*. Time stamps *earliest* and *latest*.

State: The maximal time span *timespan*; *ranges*, a stack of maximal temporal ranges between target-sequence events, with the *i*th element representing the temporal range between the first *i* target-sequence events in *M*.

```
// Get “as yet” range of timestamps or “null”-range if stack is empty
Pair<TimeStamp, TimeStamp>  lastRange;
if  (ranges.length == 0)  then
        lastRange = new  Pair<TimeStamp, TimeStamp>(null, null);
else
        lastRange = ranges.peek();
end

// Calculate new range…
Pair<TimeStamp, TimeStamp> newRange;
if (match = $\varepsilon$) then
        // Leave range unchanged in case of a null-mapping
        newRange = lastRange;
else
        // Otherwise, adapt range if necessary
        TimeStamp earliest = lastRange.First;
        TimeStamp latest = lastRange.Second;
        if (earliest = null or earliest > $v_t$(match)) then
                earliest = $v_t$(match);
        end
        if (latest = null or latest < $v_t$(match)) then
                earliest = $v_t$(match);
        end
end

// Add new range to stack
ranges.push(newRange);

// Evaluate range after earch new mapping, return “null” if illegal
if ( lastRange.First $\neq$ null              and
   lastRange.Second $\neq$null             and
    (lastRange.Second - lastRange.First) > timespan)) then
        return null;
end

// Return default costs otherwise
return new BlockResult (
        new int[] { index  + 1},
        CalcDefaultCosts(prevPatternEvent, prevMatch, patternEvent, match));
```

In the block’s *RemoveMapping()*-function, the stack’s top-element is removed via *ranges.pop()*. The *SetSucceedingMapping()*-function is irrelevant and returns null.

Example. Consider two event sequences S_p and S_t and a „maximal time span“-block $M \subseteq S_p$ with a maximal time span $ts_{max} = 20s$ as shown below:

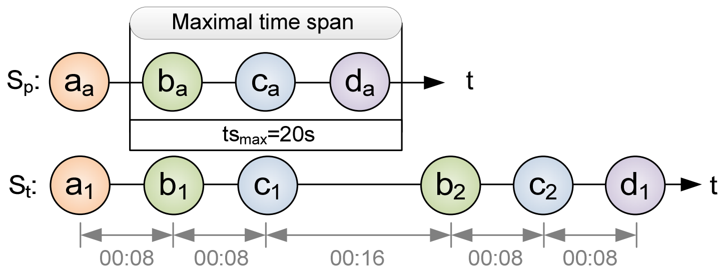

Maximal time span
S_p: a_a b_a c_a d_a t
ts_{max}=20s
S_t: a_1 b_1 c_1 b_2 c_2 d_1 t
00:08
00:08
00:16
00:08
00:08

Figure 41: Example for a "maximal time span" constraint block

With the above described block implementation and an event-type compatibility c with $c(e, f) = 0$ for all $f = \varepsilon$, the following tree is calculated; here, those nodes that are not continued due to the maximal time-span constraint are marked with a red border:

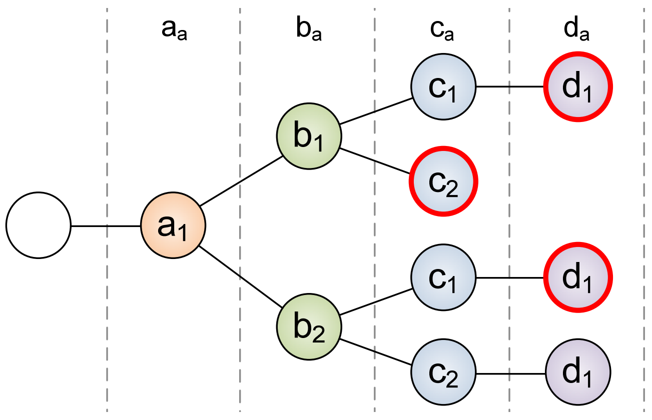


Figure 42: Tree of solutions for a "maximal time span" constraint block

5.2.2.5 Minimal time span constraints

Given a solution $s: S_p \to S_t \cup \{\varepsilon\}$ and a "Minimal time span" constraint $M \subseteq S_p$ with a minimal time span ts_{min}, let us define a *minimal time-span function* $minTSpan: {S_p}^* \to \{0,1\}$ as

$$minTSpan(e_1, e_2, \dots, e_n) = \begin{cases} 1, & \max_{1 \le i \le n} v_t(s(e_i)) - \min_{1 \le j \le n} v_t\left(s(e_j)\right) > ts_{min} \\ 0, & otherwise \end{cases}.$$

Formula 22: Minimal time span function

We now consider s valid only if $minTSpan$ results in 1 for the events in M.

Unlike in case of the maximal time-span constraint, evaluating whether the minimal time span in set of target-sequence events is greater than a certain threshold ts_{min} is only possible as soon as the complete set M_{S_t} of target-sequence events is known: From $t({M_{S_t}}') < ts_{min}$ it does not follow that $t(M_{S_t}) < ts_{min}$ for an ${M_{S_t}}' \subset M_{S_t}$. We therefore integrate the minimum-time-span functionality as follows into the base algorithm: When adding a node that represents a mapping for the last event of a "minimal time span"-block M, i.e., a mapping $(e, s(e))$ with $e \in M$ and $pos(e, M) = |M|$, we evaluate $minTSpan$ for the node and its $|M| - 1$

predecessors. If $minTSpan$ results in 1, the recursive algorithm is continued; otherwise, if $minTSpan$ results in 0, the algorithm is cancelled for the given certain path.

In pseudo-code, an efficient implementation of a “maximal time span”-block can be described as follows:

Algorithm 3: Processing of minimal time span constraints - *AddMapping()*

Input: Event prevPatternEvent, Event prevMatch, Event patternEvent, Event match, int index

Output: A *BlockResult* object if the mapping *(patternEvent, match)* is valid with respect to the given block *M*, null otherwise.

Variables: Pairs of time stamps (i.e. temporal ranges) *lastRange* and *newRange*. Time stamps *earliest* and *latest*.

State: The position *endPosition* of the last pattern-sequence element in *M*. The minimal time span *timespan*. A stack *ranges* of maximal temporal ranges between target-sequence events, with the *i*th element representing the temporal range between the first *i* target-sequence events in *M*.

```
// Get “as yet” range of timestamps or “null”-range if stack is empty
Pair<TimeStamp, TimeStamp> lastRange;
if (ranges.length == 0) then
        lastRange = new Pair<TimeStamp, TimeStamp>(null, null);
else
        lastRange = ranges.peek();
end

Pair<TimeStamp, TimeStamp> newRange;
if (match = $\varepsilon$) then
        // Leave range unchanged in case of a null-mapping
        newRange = lastRange;
else
        // Otherwise, adapt range if necessary
        TimeStamp earliest = lastRange.First;
        TimeStamp latest = lastRange.Second;
        if (earliest = null or earliest > $v_t$(match)) then
                earliest = $v_t$(match);
        end
        if (latest = null) or latest < $v_t$(match)) then
                earliest = $v_t$(match);
        end
end

ranges.push(newRange);

// Evaluate range when a last mapping  is added, return “null” if illegal
if (patternEvent.Position = endPosition        or
  lastRange.First == null                      or
  lastRange.Second == null                     or
  (lastRange.Second - lastRange.First) < timespan) then
        return null;
end

// Return default costs otherwise
return new BlockResult(
        new int[] { index  + 1},
        CalcDefaultCosts(prevPatternEvent, prevMatch, patternEvent, match));
```

In the block's *RemoveMapping()* function, the stack's top-element is removed via *ranges.pop()*. The *SetSucceedingMapping*() function is irrelevant and returns null.

Example. Consider two event sequences S_p and S_t and a „minimal time span"-block $M \subseteq S_p$ with a maximal time span $ts_{min} = 20s$ as shown below:

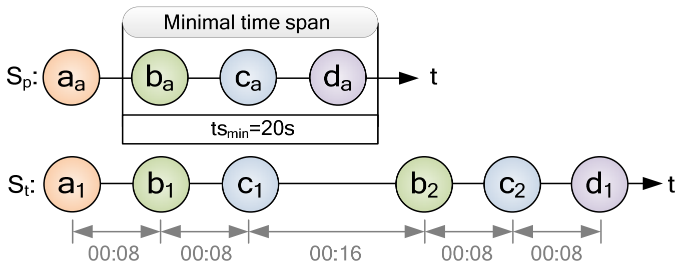


Figure 43: Example for a "minimal time span" constraint block

With the above block implementation and an event-type compatibility c with $c(e, f) = 0$ for all $f = \varepsilon$, the solutions tree is generated as shown in Figure 44. Here, those nodes that are not continued due to the minimal time-span constraint are marked with a red border:

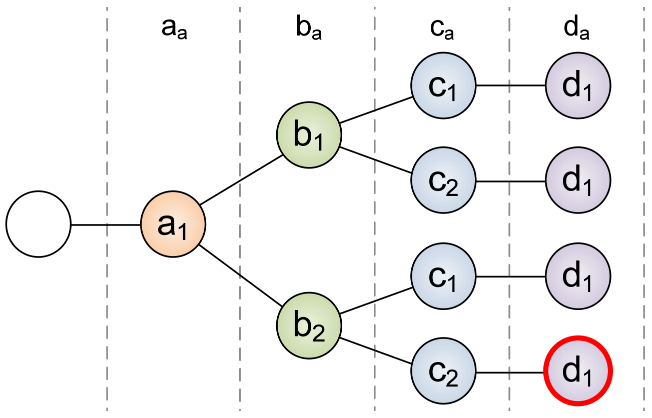


Figure 44: Tree of solutions for a "minimal time span" constraint block

5.2.2.6 Strict order constraint block

Given a solution $s: S_p \to S_t \cup \{\varepsilon\}$ and a "strict order"-constraint block $O \subseteq S_p$, let us define an *order-function* $order: S_p \times S_p \to \{0,1\}$ as

$$order(e, f) = \begin{cases} 0, & (e, f \in O) \wedge \big(pos(e, s) > pos(f, s)\big) \\ 1, & otherwise \end{cases}.$$

Formula 23: Strict order function

We now consider s valid only if $order$ results in 1 for each pair of successive pattern-sequence events (e, f), $e, f \in S_p, d(e, f, S_p) = 1$.

Strict-order constraints are integrated as follows into the base algorithm: When adding a node (i.e., a mapping) to the tree, we evaluate the order-functions for the node and its predecessor. Only if all order-functions result in 1, the recursive algorithm is continued.

In pseudo-code, an efficient implementation of the presented constraint can be expressed as follows:

Algorithm 4: Processing of strict order constraints – *AddMapping()*

Input: Event prevPatternEvent, Event prevMatch, Event patternEvent, Event match, int index

Output: A *BlockResult* object if the mapping *(patternEvent, match)* is valid with respect to the given block *M*, null otherwise.

Variables: Positions in the target-sequence *lastPosition* and *nextPosition*.

State: *positions,* a stack of positions of target-sequence events in the target sequence, with the *i*th element representing the last position throughout the first *i* target-sequence events in *M*.

```
// Get last position or "null" if stack is empty
Integer  lastPosition;
If  (position.length == 0)  then
        lastPosition =  null;
else
        lastPosition = position.peek();
end

// Calculate new position…
Integer  newPosition;
if (match = $\varepsilon$) then
        // Use last position in case of a null-mapping
        newPosition = lastPosition;
else
        // Otherwise, use match-position in the target-sequence
        newPosition = GetPositionInTargetSequence(match);
end

// Add new position to stack
positions.push(newPosition);

// Check wheter newPosition is greater than lastPosition
if ((lastPosition != null) && (lastPosition > newPosition)) then
        // Return null if invalid
        return null;
else
        // Return default costs otherwise
        return new BlockResult(
                new int[] { index  + 1},
                CalcDefaultCosts(prevPatternEvent, prevMatch, patternEvent, match));
end
```

Example. Consider two event sequences S_p and S_t and a „strict order"-constraint block as shown below:

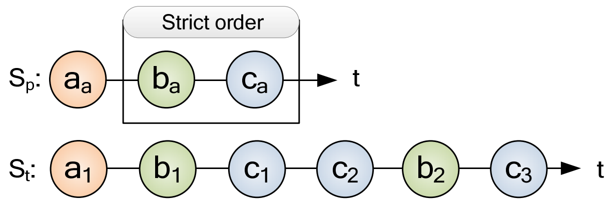


Figure 45: Example for a “strict order” constraint block

With an order-function $order$ as defined above and an event-type compatibility with $c(e,f) = 0$ if $f = \varepsilon$, the following tree of solution is generated. Again, those nodes that are not continued are marked with a red border.

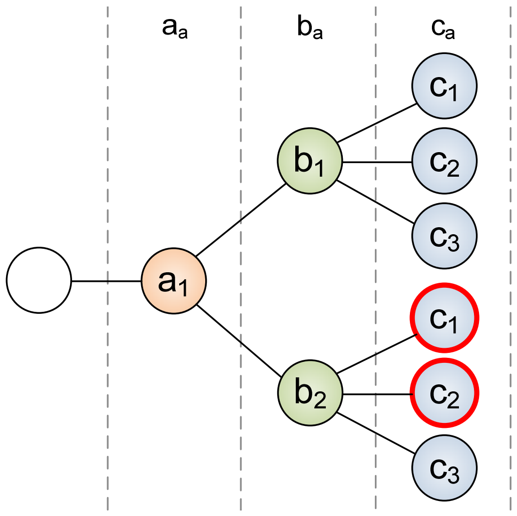


Figure 46: Tree of solutions for a “strict order” constraint block

5.2.3 Widening blocks

5.2.3.1 Arbitrary order block

In section 4.3.5.2.1 we introduced the concept of *temporal permutations* of a pattern sequence S_p with respect to a sub-sequence $A \subseteq S_p$ as an event sequence $S_p{}'$ where the times of occurrence (and, consequently, the positions in $S_p{}'$) are permutated for all events in A. All other event attributes remain equal across the events in S_p and $S_p{}'$.

Obviously, one possible approach for implementing “arbitrary order”-blocks would be to perform the base algorithm several, using the various permutations of S_p and choosing the cheapest solution from all pattern-sequences. This, however, is impracticably slow as most calculations are redundant.
Therefore, we implement “arbitrary order”-blocks as follows: Given a target sequence S_t, we find solutions “as usual”, i.e., for the original pattern sequence S_p only. Though, when calculating the (order- and temporal-structure-related) costs of a solution $s: S_p \to S_t \cup \{\varepsilon\}$, we instead consider a virtual solution $s: S_p{}' \to S_t \cup \{\varepsilon\}$, with $S_p{}'$ being the best-possible permutation of S_p with respect to s.

Example: Consider two solutions $s_1: S_p \to S_t \cup \{\varepsilon\}$ and $s_2: S_p \to S_t \cup \{\varepsilon\}$ as shown below:

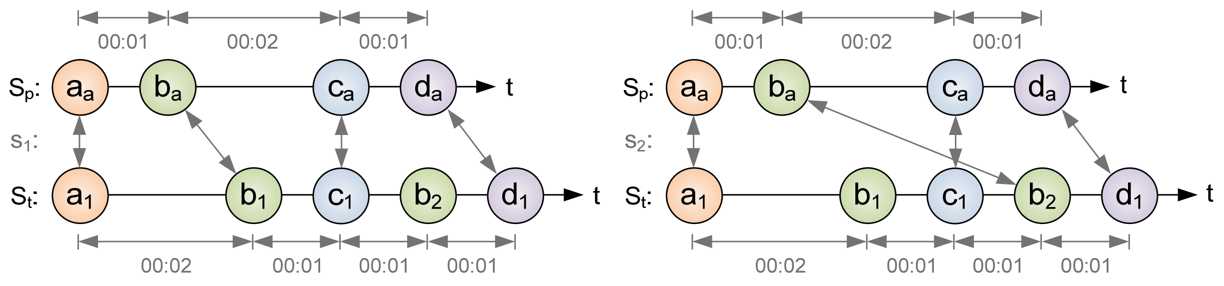


Figure 47: Example of two possible solutions for a short event sequence

Now, consider an "arbitrary order"-block $A \subseteq S_p$ as shown below:

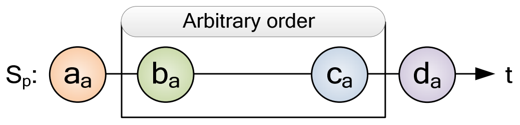


Figure 48: Example on an "arbitrary order" block

Taking A into account, calculating overall costs remains unchanged for s_1; here, S_p itself is the "best" permutation of S_p. For s_2, however, overall costs (regarding the order and the temporal structure) are calculated as for an imaginary solution $s_3: S_p{}' \to S_t \cup \{\varepsilon\}$ as shown below, with $S_p{}'$ being the optimal of permutation of S_p with respect to s_2:

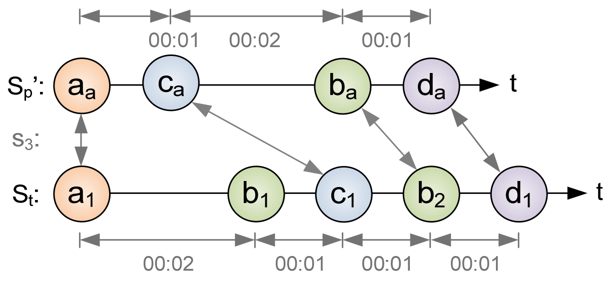


Figure 49: Optimal permutation of a pattern sequence in case of an "arbitrary order" block

5.2.3.1.1 Adapting the base algorithm

We have stated that an "arbitrary order"-block requires a conceptual adaption of the pattern-sequence depending on the given solution. As the proposed algorithm builds upon the idea of certain, fixed pattern-sequences, an implementation of the described block requires an adaption of the basic structure of the algorithm, and is thus much more difficult than for previous blocks.

Consider a pattern sequence S_p, a target sequence S_t and an "arbitrary order"-block $A \subseteq S_p$, and let e_i address the i^{th} event in A. When a tree node representing a match for e_j, $j < |A|$, is added to the tree, single-event similarities are evaluated as usual. Cost-factors regarding the order and the temporal structure, however, are *not* calculated and therefore *not* added to the costs of the current solution.

Finally, when a tree node representing a match for $e_{|N|}$ is added to the tree, we read the last $|N|$ matches from the node an its $|N| - 1$ predecessors, and get a "partly" solution $s: N \to S_t \cup \{\varepsilon\}$, $s(e_i) \in S_t \forall 1 \le i \le |N|$. "In memory", i.e., without adapting the actual structure of the tree, we now rearrange the mappings in s, so that

the earliest target-sequence event is mapped to the earliest pattern-sequence event, the second-earliest pattern-sequence event is mapped to the second-earliest target-sequence event, and so on. As all permutations of the pattern-sequence can be considered "valid", and single-event similarities have been calculated before, we do not have to care about compatibilities here. Thus, for the adapted list of mappings s', $pos(e, N) > pos(f, N) \rightarrow pos(e, s') > pos(f, s')$ holds for all $e, f \in N$.
In the last step, we calculate those cost-factors that have been omitted before, but apply the according cost-functions on the mappings in s' instead of the mappings in s. Finally, we weight these cost-factors accordingly and add them to the overall costs of the original solution.

Algorithm 5 and Algorithm 6 list these steps in pseudo-code.

Algorithm 5: Processing of arbitrary order blocks – *AddMapping()*

Input: Event prevPatternEvent, Event prevMatch, Event patternEvent, Event match, int index
Output: A *BlockResult* object containing single-event similarity costs.
Variables: -
State: A stack *matches* of previous matches in the given arbitrary-order block *A*.

```
// Add mapping to stack of mappings
mappings.Add(new Pair<Event, Event>(patternEvent, match);

// Store pre-block mapping
if ( pos(patternEvent, A) = 1) then
        preBlockPatternEvent = prevPatternEvent;
        preBlockMatch = prevMatch;
end

// Return "block result" containing (only) single similarity costs
return new BlockResult(
        new int[] { index + 1},
        CalculateSingleSimCosts(patternEvent, match));
```

Algorithm 6: Processing of arbitrary order blocks – *SetSucceedingMapping()*

Input: Event prevFinalPatternEvent, Event prevFinalMatch, Event finalPatternEvent, Event finalMatch, int index
Output: The order- and temporal-structure costs for the given, succeeding mapping.
Variables: -
State: *matches, a stack* of current matches in the given arbitrary-order block *A*.

```
// Calculate ordered list of matches
List<Event> orderedMatches = new List<Event>();
for each Pair<Event, Event> mapping in mappings
        if (mapping.Second ≠ $\varepsilon$) then
                orderedMatches.Add(mapping.Second);
        end
end

SortByTimeOfOccurence(orderedMatches);

Double costs = 0;

// Find best-possible mappings and calculate corresponding order- and temporal-structure costs
Event prevPatternEvent = preBlockPatternEvent;
Event prevMatch = preBlockMatch;
for each Pair<Event, Event> mapping in mappings
        Event match = mapping.Second;
        if (match $\neq \varepsilon$) then
                match = orderedMatches.GetFirst();
                orderedMatches.RemoveFirst();
        end

        costs += OrderAndTemporalStructureCosts(
                        prevPatternEvent, prevMatch, mapping.First, match);

        prevMatch = match;
        prevPatternEvent = mapping.First;
}

costs += OrderAndTemporalStructureCosts(
                prevPatternEvent, prevMatch, finalPrevPatternEvent, finalMatch);

return costs;
```

In the block's *RemoveMapping()* function, the stack's top-element is removed via *mappings.pop()*.

5.2.3.1.2 Example

Consider two solutions $s_1: S_p \to S_t \cup \{\varepsilon\}$ and $s_2: S_p \to S_t \cup \{\varepsilon\}$ and an "arbitrary order"-block $A \subseteq S_p$ as shown below:

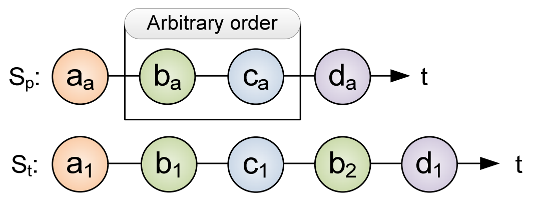
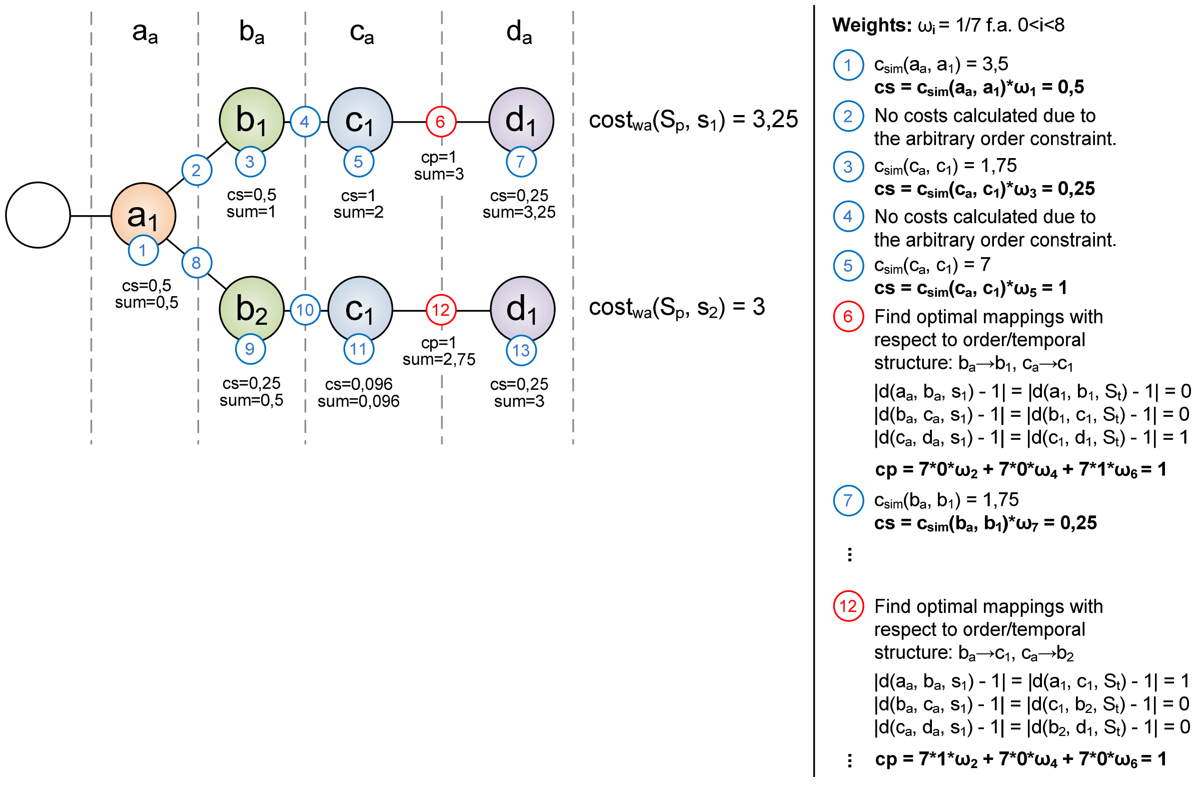


Figure 50: Example on an "arbitrary order" block

With a event-type compatibility c with $c(e,f) = 0$ for all $f = \varepsilon$, uniformly distributed weights, a however defined single-event similarity cost-function c_{sim} and an order cost-function c_{order} as based upon $c_{order}'(e, s(e), f, s(f)) = 7 * (|1 - d(e,f,s)|)$, costs are calcualted as presented below. Those calculation steps that are particularly interesting regarding the presented block, 6 and 12, are highlighted in red.



Weights: $\omega_i = 1/7$ f.a. $0<i<8$

1. $c_{sim}(a_a, a_1) = 3{,}5$
   $cs = c_{sim}(a_a, a_1)*\omega_1 = 0{,}5$
2. No costs calculated due to the arbitrary order constraint.
3. $c_{sim}(c_a, c_1) = 1{,}75$
   $cs = c_{sim}(c_a, c_1)*\omega_3 = 0{,}25$
4. No costs calculated due to the arbitrary order constraint.
5. $c_{sim}(c_a, c_1) = 7$
   $cs = c_{sim}(c_a, c_1)*\omega_5 = 1$
6. Find optimal mappings with respect to order/temporal structure: $b_a \rightarrow b_1$, $c_a \rightarrow c_1$
   $|d(a_a, b_a, s_1) - 1| = |d(a_1, b_1, S_t) - 1| = 0$
   $|d(b_a, c_a, s_1) - 1| = |d(b_1, c_1, S_t) - 1| = 0$
   $|d(c_a, d_a, s_1) - 1| = |d(c_1, d_1, S_t) - 1| = 1$
   $cp = 7*0*\omega_2 + 7*0*\omega_4 + 7*1*\omega_6 = 1$
7. $c_{sim}(b_a, b_1) = 1{,}75$
   $cs = c_{sim}(b_a, b_1)*\omega_7 = 0{,}25$

⋮

12. Find optimal mappings with respect to order/temporal structure: $b_a \rightarrow c_1$, $c_a \rightarrow b_2$
   $|d(a_a, b_a, s_1) - 1| = |d(a_1, c_1, S_t) - 1| = 1$
   $|d(b_a, c_a, s_1) - 1| = |d(c_1, b_2, S_t) - 1| = 0$
   $|d(c_a, d_a, s_1) - 1| = |d(b_2, d_1, S_t) - 1| = 0$

⋮ **$cp = 7*1*\omega_2 + 7*0*\omega_4 + 7*0*\omega_6 = 1$**

Figure 51: Cost calculation example in case of an “arbitrary order” block

Note that because of the “arbitrary order” block and slightly better single-event similarities, the apparently uncommon solution s_2 is considered optimal:

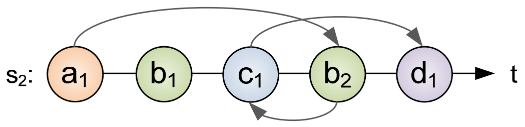


Figure 52: Optimal solution for “arbitrary order” block example

5.2.3.2 Occurrence number blocks

In section 4.3.5.2.2 we introduced the concept of *foldings* of a pattern sequence S_p which contains multiple subsequent occurrences of a subsequence $O \subseteq S_p$ at the original position of O in S_p. We further noted that the *order* of all events in a folding S_n is naturally defined, whereas the exact temporal structure is undetermined so that we require the user to configure how to compute the timespan between foldings of O. Figure 53 again illustrates the foldings S_0 through S_3 for a simple example.

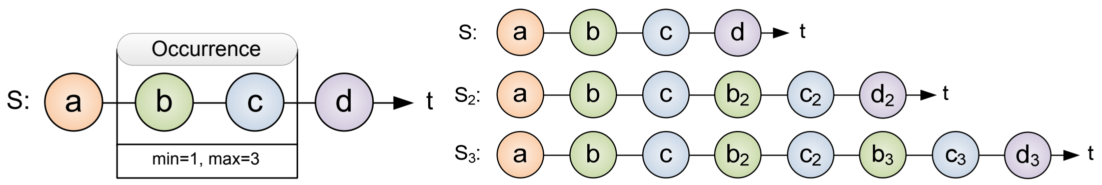


Figure 53: Foldings of a sample sequence for an occurrence number block

5.2.3.2.1 Adapting the base algorithm

Consider a pattern sequence S_p, a target sequence S_t and an "occurrence number"-block $O \subseteq S$ with a minimal occurrence of min and a maximal occurrence of max, and let e_i address the ith event in the max-folding of S, $S_{p_{max}}$. Consequently, let e_{k+i*n} address the nth event in the ith iteration of O in $S_{p_{max}}$.

When calculating the overall costs between S_p and S_t, we start creating a tree as if the maximal folding $S_{p_{max}}$ was the pattern-sequence we search solutions for. Thereby, we derive the weights for $S_{p_{max}}$ from the set of weights for O by proportionally distributing the normal weights without any folding within the maximal folding of O. E.g., when a weight of ω_{k+l} was originally chosen for the order cost-factor of (e_{k+l}, e_{k+l+1}), we assume weights of $\frac{\omega_{k+l}}{max}$ for the order cost-factors of $\left(e_{k+(i-1)*|O|+l}, e_{k+(i-1)*|O|+l+1}\right)$, $i = min, \ldots, max$.

Now, when adding a tree-node that represents $e_{k+i*|O|}$, $i \geq min$, we allow creating a "short-cut" for exiting the iterations part of $S_{p_{max}}$: To the node representing $e_{k+i*|O|}$, we allow adding "additional" tree-nodes representing matches for $e_{k+max*|O|+1}$, i.e., the event that succeeds the last iteration of O in $S_{p_{max}}$. If no such event is available, we allow reaching a (virtual) leaf: Here, the node representing $e_{k+i*|O|}$ is considered the last mapping of a certain solution s, but nonetheless serves as an origin for further solutions. For calculating cost-factors for $\left(e_{k+i*|O|}, e_{k+max*|O|+1}\right)$, we assume $t\left(e_{k+i*|O|}, e_{k+max*|O|+1}\right) = t\left(e_{k+max*|O|}, e_{k+max*|O|+1}\right)$ and $d\left(e_{k+i*|O|}, e_{k+max*|O|+1}, S_p\right) = t\left(e_{k+max*|O|}, e_{k+max*|O|+1}, S_p\right) = 1$, and also chose weights as if we were calculating cost-factors for $\left(e_{k+max*|O|}, e_{k+max*|O|+1}, S_p\right)$.

Keep in mind, however, that for "short-cut" solutions, the sum of weights may be smaller than 1. Therefore, when adding a short-cut to a node representing $e_{k+i*|O|}$, we read the cost-factors for e_j, $k < j \leq i * |O|$, and for (e_l, e_{l+1}), $k < l < i * |O|$, i.e., the cost-factors for those parts of the current solution that comprise elements of the iterations part of $S_{p_{max}}$ and add them to the solution's costs so that *a.)* the sum of weights is correct with respect to the current pattern-sequence event and *b.)* the proportions between the inner-block weights are maintained.

Algorithm 7, Algorithm 8 and Algorithm 9 express the implemented strategy in pseudo-code.

Algorithm 7: Processing of arbitrary order blocks – *AddMapping()*

Input: Event prevPatternEvent, Event prevMatch, Event patternEvent, Event match, int index

Output: A "BlockResult" object.

Variables: *c,* representing the costs for the current mapping *(patternEvent, match)*. *relativePosInBlock,* the relative position of *patternEvent* in *O*. *costFunctions*, the given cost functions. *costFunction*, a cost function.

State: *costs*, a stack of "so far" weighted cost factors calculated in the given occurrence-number block *O*. *sumOfWeights,* the "so far" weights assigned in *O*.

```
// Generate costs for the added mapping, add cost factors and weights to stack/"sum of weights" for
// later normalization
Double c = 0;
for each CostFunction costFunction in costFunctions
        Double costFactor = costFunction.GetCosts(
                prevPatternEvent, prevMatch, patternEvent, match);
        Double weight = GetWeight(patternEvent, costFunction);

        costs.push(new WeightedCostFactor(costFactor, weight);

        c += weight * costFactor;
        sumOfWeights += weight;
end

// Check relative position in block and decide wether a shortcut can be added
Integer relativePosInBlock = index – startIndex;
if ((relativePosInBlock + 1) % 1 length = 0 and
    relativePosInBlock + 1) / length >= minIterations)) then
        // Return block result including shortcut
        return new BlockResult(c, new int[] { index + 1, endIndex + 1 });
else
        // Return block result not including a shortcut
        return new BlockResult(c, new int[] { index + 1 });
end
```

Algorithm 8: Calculation of arbitrary order blocks – *SetSubsequentMapping()*

Input: Event prevPatternEvent, Event prevMatch, Event postBlockPatternEvent, Event postBlockMatch, int index

Output: The costs for the given mapping *(postBlockPatternEvent, postBlockMatch)*.

Variables: *totalSumOfWeights*, holding the maximal sum of weights to be assigned in the given block *O. lastBlockPatternEvent,* the last pattern event in the maximal folding of *O. costs,* the return value.

State: *costs*, a stack of “so far” weighted cost factors calculated in the given occurrence-number block *O*. *sumOfWeights,* the “so far” weights assigned in *O*.

```
// Call “AddMapping” (with the block’s last pattern-sequence event instead of the “correct” previous
// pattern-sequence event) to calculate the mapping’s basic costs
BlockResult blockResult = AddMapping
                (lastBlockPatternEvent, prevMatch, postBlockPatternEvent, postBlockMatch, index);

Double costs = blockResult.Costs;

// Normalization of prior costs so that overall sum of weights is reached
for each WeightedCostFactor weightedCostFactor in costs
        costs += weightedCostFactor.Costs *
                        (weightedCostFactor.Weight / sumOfWeights) *
                        (totalSumOfWeights – sumOfWeights);
end

// Remove previously added mapping
RemoveMapping(postBlockPatternEvent, index);

return costs;
```

Algorithm 9: Calculation of arbitrary order blocks – *RemoveMapping()*

Input: EventWrapper patternEvent, int index

Output: -

State: *costs*, a stack of “so far” weighted cost factors calculated in the given occurrence-number block *O*. *sumOfWeights,* the “so far” weights assigned in *O*.

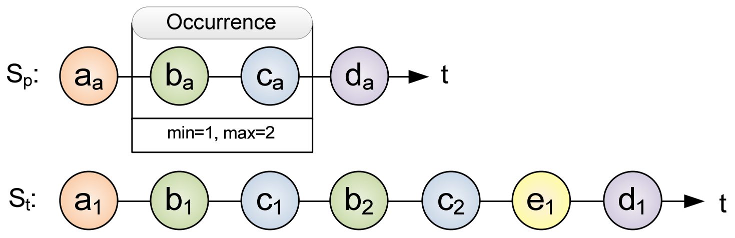
Variables: *costFunction*, the given cost functions. *costFunction*, a cost function. *weightedCostFactor*, the *costs*-stack’s current top-level element.

```
// Remove corresponding cost factors from stack, reduce “total costs”
for each CostFunction costFunction in costFunctions
        WeightedCostFactor weightedCostFactor = costs.pop();
```

```
4:              totalCosts -= weightedCostFactor.Weight;
5:    end
```

5.2.3.2.2 Example

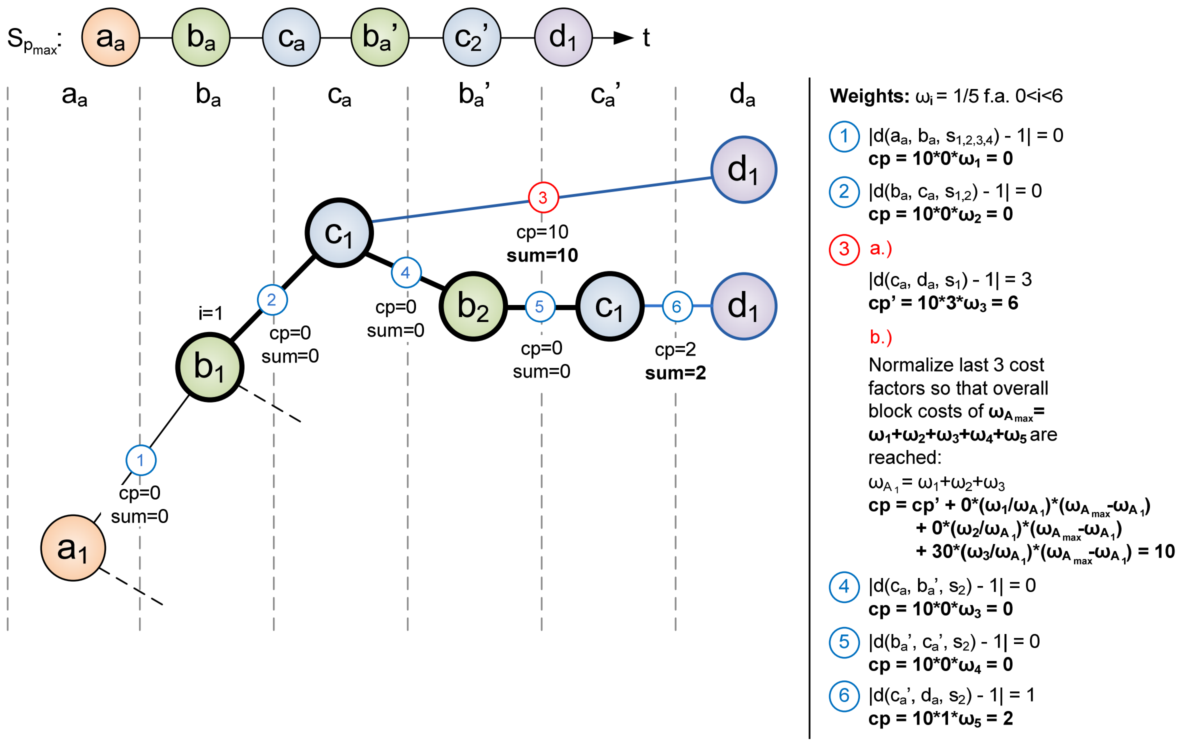
Consider a pattern sequence S_p, a target sequence S_t, and an "occurrence number"-block $O \subseteq S_p$ as shown below:



Occurrence
S_p: a_a b_a c_a d_a t
min=1, max=2
S_t: a_1 b_1 c_1 b_2 c_2 e_1 d_1 t

Figure 54: Example of an "occurrence number" block

With a event-type compatibility c with $c(e,f) = 0$ for all $f = \varepsilon$, weights uniformly distributed over the adapted pattern sequence $S_{p_{max}}$ and an order cost-function c_{order} as based upon $c_{order}'(e, s(e), f, s(f)) = 10 * (|1 - d(e,f,s)|)$, costs are calcualted as presented below. Being particularly interesting with respect to the presented block, calculation step 3 is marked red. Also, for the sake of brevity, we decided to focus on a smaller part of the (rather larger) overall tree.



$S_{p_{max}}$: a_a b_a c_a b_a' c_2' d_1 t
a_a b_a c_a b_a' c_a' d_a
a_1 b_1 c_1 b_2 c_1 d_1 d_1
i=1
1 cp=0 sum=0
2 cp=0 sum=0
3 cp=10 sum=10
4 cp=0 sum=0
5 cp=0 sum=0
6 cp=2 sum=2
Weights: $\omega_i = 1/5$ f.a. 0<i<6
1 $|d(a_a, b_a, s_{1,2,3,4}) - 1| = 0$ cp = $10*0*\omega_1 = 0$
2 $|d(b_a, c_a, s_{1,2}) - 1| = 0$ cp = $10*0*\omega_2 = 0$
3 a.) $|d(c_a, d_a, s_1) - 1| = 3$ cp' = $10*3*\omega_3 = 6$
b.) Normalize last 3 cost factors so that overall block costs of $\omega_{A_{max}} = \omega_1+\omega_2+\omega_3+\omega_4+\omega_5$ are reached:
$\omega_{A_1} = \omega_1+\omega_2+\omega_3$
cp = cp' + $0*(\omega_1/\omega_{A_1})*(\omega_{A_{max}}-\omega_{A_1})$ + $0*(\omega_2/\omega_{A_1})*(\omega_{A_{max}}-\omega_{A_1})$ + $30*(\omega_3/\omega_{A_1})*(\omega_{A_{max}}-\omega_{A_1}) = 10$
4 $|d(c_a, b_a', s_2) - 1| = 0$ cp = $10*0*\omega_3 = 0$
5 $|d(b_a', c_a', s_2) - 1| = 0$ cp = $10*0*\omega_4 = 0$
6 $|d(c_a', d_a, s_2) - 1| = 1$ cp = $10*1*\omega_5 = 2$

Figure 55: Solution tree (excerpt) and calculated costs for an "occurrence number"-block

5.2.3.3 Arbitrary events

5.2.3.3.1 Implementation approach 1

One possible implementation of arbitrary events is that of using an adapted version of the base compatibility as shown above. Therewith, all events of the target sequence are considered compatible to events of type *Arbitrary*.

5.2.3.3.2 Implementation approach 2

The described approach on arbitrary events is in full accordance with the base algorithm and treats arbitrary events just like any normal one: An arbitrary event may, for instance, be placed at an exact point in time and thus be considered in a temporal-structure cost-function, or it may be part of constraint block. Yet, by massively extending the overall set of matches, arbitrary events may result in a notable growth of the tree, and thereby slow down the calculation.

We therefore propose an alternative, block-based implementation of arbitrary events that builds upon the (in fact, simplistic) assumption that for a (sub-)pattern-sequence $S' = (e, x_1, x_2, \dots, x_n, f)$, $x_1, x_2, \dots, x_n \in \mathbb{X}$, $e, f \notin \mathbb{X}$, $x_1, x_2, \dots, x_n$ are mapped to target-sequence events somewhere "in between" the target sequence events mapped to e and f in the best-possible solution $s: S \rightarrow T \cup \{\varepsilon\}$. In other words, if $pos(e,s) \geq pos(f,s)$, we assume that $pos(e,s) \geq pos(x_i) \geq pos(f,s) \forall 1 \leq i \leq n$. Otherwise, if $pos(e,s) \leq pos(f,s)$, we assume that $pos(e,s) \leq pos(x_i) \leq pos(f,s) \forall 1 \leq i \leq n$.[5]

In the pattern editor, we let the user create an "arbitrary events"-block around two successive pattern-sequence events $e, f \in S_p$ and define a range min to max of arbitrary events that shall be "between" the matches for e and f in the target sequence. When calculating the order cost-factors for a target sequence S_t, we assume a (virtual) distance between e and f in S_p that is a.) in $[min + 1, max + 1]$ and b.) optimal with respect to the distance between e and f in the given solution s, $d(e,f,s)$. Consider, for instance, an "arbitrary event"-block with $min = 3$ and $max = 5$: Here, for a solution with $d(e,f,s) = 4$, we assume $d(e,f,S_p) = 4$. For a solution with $d(e,f,s) = 7$, however, we assume $d(e,f,S_p) = 6$ as the distance must be in $[min + 1, max + 1]$.

The question arising is what happens if there are fewer events between e and f in s than minimum number of arbitrary events required? From the above assumption, it clearly follows that the minimal number of arbitrary events *must* be mapped to something in between; thus, one might consider something similar to null-matches for non-mapped arbitrary events. As arbitrary events are usually less critical then "normal" events (otherwise they wouldn't be "arbitrary"), we decided to implement a very simple null-match approach that differs from the rather complex and cost-function specific default implementations: Whenever the distance between e and f in a solution s is smaller than min, we add the according number of fixed and user-defined "pseudo null-node" costs to the (still to be weighted) order-cost factor. E.g., for a solution with $d(e,f,s) = 2$, we assume $d(e,f,S_p) = 2$ but add two times the user-defined pseudo null-node costs.

Algorithm 10 and Algorithm 11 list the calculation steps in pseudo-code.

[5] Equality is allowed here due to possible null-mappings.

Algorithm 10: Processing of arbitrary event blocks – *AddMapping()*

Input: Event prevPatternEvent, Event prevMatch, Event patternEvent, Event match, int index

Output: A "BlockResult" object.

Variables: *endIndex*, the end index of the given arbitrary event block. *minEvents*, the minimum number of arbitrary events, *maxEvents*, the maximum number of arbitrary events. *nullNodeCosts*, the costs for "arbitrary" null nodes. *c,* representing the costs for the current mapping *(patternEvent, match).*

```
// If pattern event is the second event in the given block, perform special cost calculation
if (index = endIndex) then
        // If match is a null node, calculate costs as usual
        if (match = $\varepsilon$) then
                return new BlockResult(
                        CalculateDefaultCosts(prevPatternEvent, prevMatch, patternEvent, match),
                        index + 1);
        else
                // Calculate single sim & temp structure costs as usual
                Double regularCosts = CalculateSingleSim&TempStructureCosts(
                        prevPatternEvent, prevMatch, patternEvent, match);

                // Get/update "last position" in the order cost-function.
                // For more details on the order cost-function, refer to Obweger [37].
                Integer lastPosition = orderCostFunction.CalculateLastPosition(
                        prevPatternEvent, prevMatch, patternEvent, match);

                // Calculate the distance between match and the last previous non-null match ("last
                // position") in the target sequence.
                Integer targetSeqDist = match.Position – lastPosition;

                // Find a pattern sequence distance "optimal" with respect to patternSeqDist and
                // maxEvents.
                Integer patternSeqDist;
                if (targetSeqDist > 0) then
                        patternSeqDist = Min(maxEvent, targetSeqDist);
                else
                        patternSeqDist = 1;
                end

                // Calculate order costs as if the distance between prevPatternEvent and
                // patternEvent was "optimal", i.e., patternSeqDist.
                Integer orderCosts = orderCostFunction.CalculateCosts(
                        targetSeqDist, patternSeqDist);

                // Finally, if there are too few events between match and prevMatch regarding
                // minEvents, add an according number of "null node costs" to orderCosts.
                Integer missingInTargetSeq = _minEvents – Abs(targetSeqDist);
                if (missingInTargetSeq > 0) then
                        orderCosts += missingInTargetSeq * nullNodesCosts;
                end

                return new BlockResult(
                        regularCosts + orderCosts * GetWeight(orderCostFunction, patternEvent),
                        index + 1);
        end

// Otherwise, calculate costs as usual
```

```
else
    return new BlockResult(
        CalculateDefaultCosts(prevPatternEvent, prevMatch, patternEvent, match),
        index + 1);
end
```

Algorithm 11: Retrieving/Updating the last target-sequence position in the order cost-function

Input: Event prevPatternEvent, Event prevMatch, Event patternEvent, Event match, int index

Output: The last successfully mapped target-sequence position, null if not available.

Variables: *lastPosition,* the return value.

State: *lastPositions,* a field holding the last successfully mapped target-sequence position for each pattern sequence index.

```
Integer lastPosition = null;
if (prevMatch = $\varepsilon$) then
    // If prevMatch is a null match, get position from index - 1
    lastPosition = lastPositions[index - 1];
else
    // Otherwise, get prevMatch's position
    lastPosition = GetPositionInTargetSequence(prevMatch);
end
lastPositions[index] = lastPosition;

return lastPosition;
```

Both the block's *SetSucceedingMapping()* function and the block's *RemoveMapping*() function are irrelevant: The former returns *null*, the latter does nothing at all. Note that with the alternative approach, the user can define the pure existence of arbitrary events, yet he or she cannot define an exact time stamp. Thus, the proposed blocks only affect order cost-factors. Also, for several reasons, the algorithm does not necessarily calculate correct overall solution costs:

- In cases where null-mappings are considered, it may be more efficient to map an event that is outside the surrounding non-arbitrary mappings.
- As the algorithm does not mark those target-sequence events that are considered as mapped to arbitrary events as part of the given solution(s), those may "re-mapped" to later pattern-sequence events. This clearly conflicts with the algorithm's general approach and may result in solutions with too low costs.

Example. Consider a pattern sequence S_p, a target sequence S_t, and an "arbitrary events"-block $A \subseteq S_p$ with $min = 2$ and $max = 4$ as shown below:

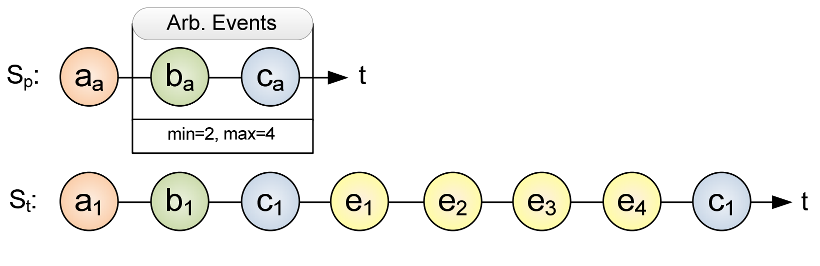
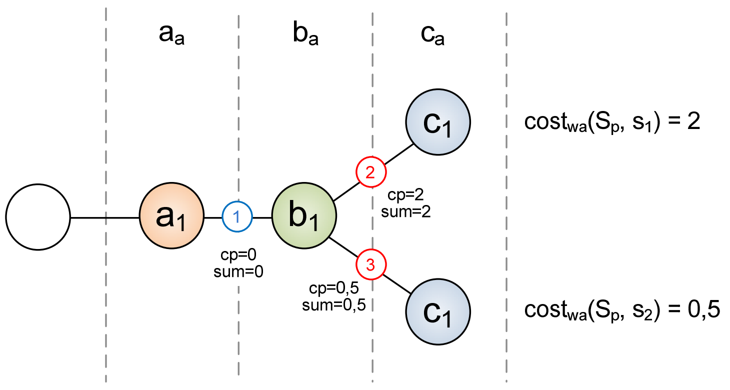


Figure 56: Solution tree (excerpt) and calculated costs for an "occurrence number"-block

With a event-type compatibility c with $c(e,f) = 0$ for all $f = \varepsilon$, uniformly distributed weights and an order cost-function c_{order} as based upon $c_{order}'(e, s(e), f, s(f)) = (|d(e,f,S_p) - d(e,f,s)|)$, costs are calculated as presented below. Again, those calculation steps that are particularly interesting with respect to the presented block, 2 and 3, are marked red.



Weights: $\omega_i = 1/2$ f.a. $0<i<3$

1 $|d(a_a, b_a, s_{1,2}) - 1| = 0$
cp = $0*\omega_1 = 0$

2 As $d(b_a, c_a, s_1)$ is smaller than *min* + 1,
min + 1 - $d(a_a, b_a, s_1)$ null-mappings must be considered. We therefore calculate order costs as follows:

n = min + 1 - $d(a_a, b_a, s_1) = 2$
cp = $n*c*\omega_2 = 2*2*0.5 = 2$

3 As $d(b_a, c_a, s_2)$ is larger than *min* + 1, all arbitrary events could be mapped to events of the target sequence. Thus, we find an optimal virtual distance of b_a and c_a in S_p with respect to s_2, *min* and *max*, $d_{s_2, min, max}(b_a, c_a, S_p) = 4$, and calculate the order cost-factor as follows:

$|d(a_a, b_a, s_{1,2}) - d_{s_2, min, max}(b_a, c_a, S_p)| = 1$
cp = $1*\omega_2 = 0,5$

Figure 57: Solution tree (excerpt) and calculated costs for an "arbitrary events"-block

5.2.4 Asymptotic runtime

In his thesis, Obweger [37] discusses the complexity of the base algorithm, which is in the worst case directly proportional to the number of possible solutions $\sum_{k=0}^{min(|S_p|,|S_t|)} \left(\frac{(|S_p|)!}{(|S_p|-k)!}\right) * \binom{|S_t|}{k}$ (see section 5.1.1). At this stage, we refer the interested reader to this thesis, and limit the discussion to the impact of presented enhanced pattern sequence building blocks. It is still important to note that in practice, we avoid the worst-case runtime through

- compatibilities, restricting the set of valid solutions, and the
- dynamic threshold, allowing us to skip costly solutions early in the calculation.

In summary, the basic runtime may be influenced by the additional blocks in one of the following ways:

- Increase or decrease the number of compatible events.
- Influence the probability of solutions being omitted due to exceeding the threshold.
- Add to the total number of possible solutions.

5.2.4.1 Restrictive blocks

Restrictive blocks have been introduced as blocks which do not decrease a similarity score proportionally to a deviation, but omit solutions if the so-defined restrictions or constraints are violated. Thus, using such blocks in general reduces the number of valid solutions and therewith the runtime. We count the following blocks to the family of restrictive blocks. All restrictive blocks influence the runtime positively by decreasing limiting compatibilities.

- **Attribute constraints** – Certain matches will not be compatible any longer due to violating an attribute constraint.
- **Time of occurrence constraints** – Similar to attribute constraints: Matches outside the allowed time span are not compatible any longer.
- **Maximal time span constraints** – Also decrease the compatibility with the only difference that these constraints can be evaluated only after two or more events in a block have been mapped. The asymptotic runtime remains equal though.
- **Minimal time span constraints** – Decrease the compatibility in the same way as the maximal time span constraints, with the difference that evaluation is possible only after *all* events in the block are mapped.
- **"Strict order" constraint block** – Mappings which are not in the correct order are incompatible within the scope of the block.
- **"Required"-block** – Decreases the compatibility as for events in the block null-mappings are not compatible any longer.

5.2.4.2 Widening blocks

Intuitively, widening blocks have negative impact on the runtime by weakening the matching or allowing additional solutions to be built. We count the following blocks to the family of widening blocks:

- **Arbitrary order block** – Arbitrary order blocks do neither influence compatibilities nor the total number of solutions. Yet, such blocks decrease the probability of solutions being omitted by the threshold, as within the block order deviations are not scored so that solutions in general have lower costs.
- **Occurrence number blocks** – Occurrence number blocks have the worst influence in terms of the runtime. Such a block increases the number of possible solutions by the maximum number of foldings plus the maximum number of iterations at the solution tree level of the first match after the block. Considering a small example makes it clearer: Having an occurrence number block which allows one to five occurrences of a subsequence, this subsequence is now added five times to the pattern and in addition solutions are possible that map the first occurrence only and then take the described "short-cut" to the event following the block, or two occurrences and so forth.
- **Arbitrary events (implementation approach 1)** – This approach increases the compatibility. An arbitrary event is an event with a very high compatibility, and thus can be mapped to any event. In general this causes more solutions to be valid.
- **Arbitrary events (implementation approach 2)** – This implementation model increases or decreases the probability for solutions being omitted due to exceeding the threshold. An increase is possible in combination with a defined minimum number of arbitrary events. In that case, many null-mappings have to be build if the target sequence does not contain sufficient events, and thus the costs increase faster. Typically yet, it will decrease the probability and slow down the matching process.

In total, positive influence on the compatibilities or hitting the threshold early is very desirable. Thus, where possible the user should try to restrict the search in order to omit considering a large bulk of bad solutions.

5.3 Time series similarity for event attributes

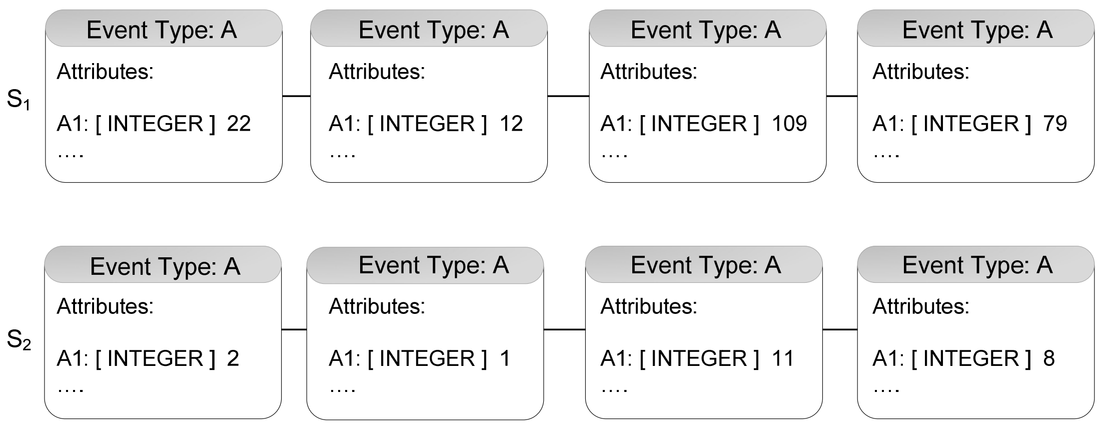
5.3.1 Overview and requirements

From the application examples named in section 3 it becomes obvious that simple event-by-event comparisons with a distance metric such as normalized absolute difference may not be sufficient in all application areas. Typical examples requiring other models are chart pattern discovery in financial market analysis or numeric series of machine precision measures, sensor logs and similar series occurring in manufacturing applications. Therefore, additional similarity techniques are proposed. In the following, these are referred to as

- Normalized sequence similarity and
- Normalized relative sequence similarity.

5.3.1.1 Normalized sequence similarity

The normalized sequence similarity technique is used for normalizing attribute values prior to assessing similarity. Normalization is performed relatively to any reference value. For instance, the first value of a sequence of values may be used as a reference. In Figure 58, 2 event sequences S_1 and S_2 are given. The sequences of values are shown before and after normalization. The normalized values can then be compared to each other with any similarity matching algorithm.

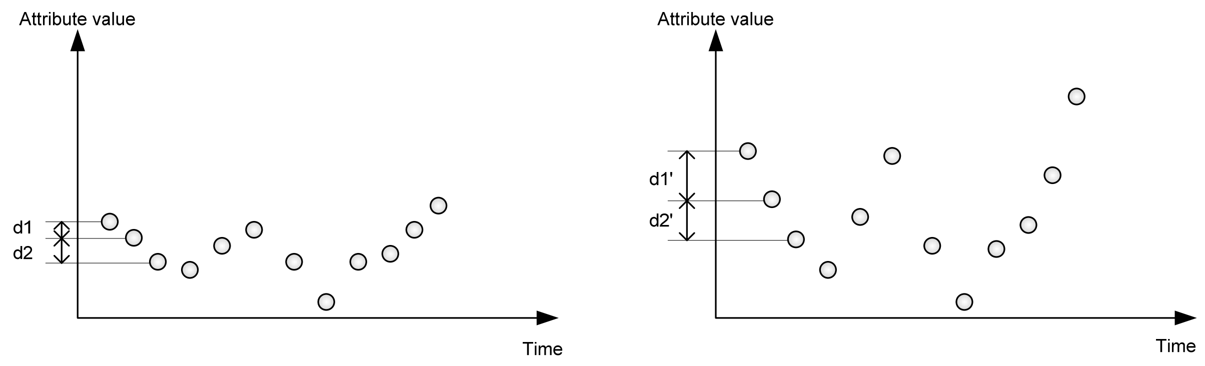


	Before normalization				After normalization			
Value sequence 1	22	12	109	79	1	0,54	4,95	3,95
Value sequence 2	2	1	11	8	1	0,5	5,5	4

Figure 58: Normalized sequence similarity

5.3.1.2 Normalized relative sequence similarity

The normalized relative sequence similarity technique considers the relative distance between subsequent values in a value series. An example is given in Figure 59.



	Before normalization			After normalization		
Distance values sequence 1	2	2,3	(…)	1	1,15	(…)
Distance values Sequence 2	4	4,5	(…)	1	1,125	(…)

Figure 59: Attribute value series for normalized relative sequence similarity

In the example, the normalized sequence similarity technique is applied to the distances between the values instead of applying it to the values itself. In this way, the two value series shown in Figure 59 are considered as being similar to each other.

In each of these two cases, after a transformation/data extraction step a series of ordered value pairs consisting of a time stamp and an according value has to be compared to a corresponding series in the reference pattern. Hence, an appropriate time-series similarity model has to be found to perform this comparison, and it must be compatible for being integrated into the base algorithm. In the following, the applied time-series similarity model is discussed in detail, before its integration into the event similarity algorithm is described.

5.3.2 Applied time-series similarity model

5.3.2.1 Overview and requirements

This section describes the time-series similarity approach which is applied and integrated into the similarity matching of event sequences. From the requirements derived from named application examples, the time-series approach has to support the following parameters:

- The time-scaling of target sequence and reference pattern may be distinct, i.e., these are not equally sampled time-series.
- Time-series do not necessarily have a constant sampling rate, the time between single data points may vary strongly.
- The algorithm must support full sequence matching as well as subsequence matching and the intermediates of a matching starting at the first data point or ending with the last data point.
- The output of the algorithm must be a ranking of matches, so that these can be combined with further characteristics of the complete event sequence. This means in particular that the time-series comparison of an attribute can be only one of several considered characteristics, and the best match of the time-series comparison is not automatically the best match overall.

Due to these requirements, many existing approaches fall short in one or the other way. For instance, many approaches rely on constant value sampling, or do not support subsequence matching. Therefore, a new approach is proposed, which adopts some of the existing ideas and extends them to be applicable in the given environment.

5.3.2.2 Method summary

The applied time-series model is based on the idea of utilizing the slopes between the single data points for comparing time-series data [50] [51]. The existing model of Toshniwal and Joshi assumes that data points are sampled regularly in order to subdivide the value series into regular slices. In the given context, the problem is that the time span between events, and therefore between the data points of the time-series, may vary strongly. Therefore, the question of how to subdivide the series into several slices in order to compare the slopes, arises. The second problem of the existing slope model is that it does not support subsequence matching and different scaling on the time-axis of target sequence and reference pattern. Considering a simple example from the stock chart pattern domain underlines the problem: One of the classical problems in this domain is the “W-pattern”: A chart formation which looks like a “W” or a reversed “W” may be an indication of a trend reversal.

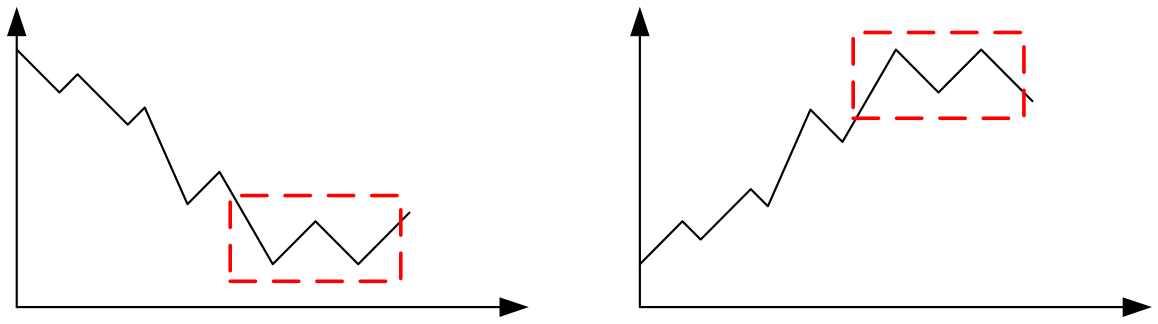


Figure 60: W-Formation in stock charts

In practice, of course, such patterns may not always be that clear and intermediate, outlying data points may be present. Therefore, the detection of such a pattern is a good example for applying fuzzy similarity techniques.

Considering this example it becomes clear that a fixed-scale similarity model fails in the pattern detection: A W-formation may occur at the scale of years, but also within one day. Having the pattern defined as a reference pattern, the similarity comparison model must be flexible enough to detect it at different scales. Also, the relative height of the “W” might vary, depending on the prior history and first of all the volatility of the stock.

These aspects are considered in the proposed similarity model. It is based on the idea of subdividing the time-series into slices at those points of the series where the movement trend of the series is turning and using these turning points as the basis for comparing the slopes of the series. As a consequence, the scale at which a pattern is detected strongly depends on the turning points selected. When extracting only the turning points of the long-term movement, short-term movements will be smoothed implicitly and remain unconsidered.

The determination of the turning points utilizes well-known and simple techniques from the financial market analysis. Here, so-called trend following strategies rely on mechanisms to detect changes in the overall direction of a price movement over a certain period. Among these mechanisms, the moving average (MA) intersection technique is one of the most simple and most popular ones. The simple moving average (SMA) with a period of n is the unweighted arithmetic mean of the previous n data points. It is used to smooth-out short-term fluctuations and highlighting long-term trends or cycles. Mathematically, building the moving average is simply an example of a convolution, so it is similar to low-pass filters in signal processing.

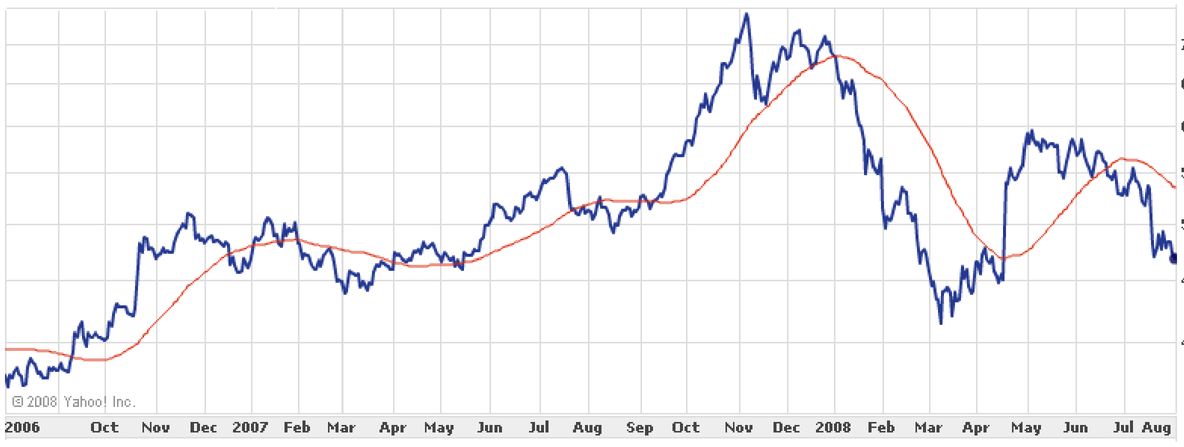


Figure 61: Example of stock chart with moving average[6]

Figure 61 provides an example of a 2-years stock chart and the moving average (red) with a period of 50. For generating trading signals based on such a MA curve, the following rules are applied:

- When the price curve crosses the MA line from top to bottom, an up-trend turns into a down-trend. This gives a sell-signal.
- When the price curve crosses the MA line from bottom to top, a down-trend turns into an up-trend. This gives a buy-signal.

These simple rules are based on the idea that the smoothed MA-line always moves behind the price curve. As long as the price does not change the major direction, they do not cross. When they cross, it is an indication of a trend change.

Exactly these crossing points can also be utilized to extract the turning points in the trend of the overall chart with respect to a given MA period. The algorithm first computes the MA and determines the crossing points. In the next step, the slopes between these points are computed and finally they are compared to each other. These steps are repeated in multiple passes with varying MA periods. In this way a search pattern can be detected at different scaling levels. At the end, the result contains a list of possible matches for the series, each match having a sum of slope deviations and a starting point in the series. Algorithm 12 summarizes the steps in pseudo code.

Algorithm 12: Base algorithm for comparing two time series

Input: TimeSeries pattern, TimeSeries targetSeries

Output: Sorted list of similarity matches as tuples of time-series subsequence and similarity score

Variables: Arrays of time-series data points *patternPoints*, *targetPoints*; *deviation* (sum of slope deviations for a certain match); Arrays of slope values *patternSlopes*, *targetSlopes*; Indizes *MAPeriod*, *startpoint*, *index*

```
1:  //iterate in multiple passes through the data with varying MA period
2:  for MAPeriod = minPeriod to maxPeriod step precision
```

[6] Chart created with YAHOO Finance, finance.yahoo.com. Copyright: YAHOO Inc.

```
        DataPoint[] patternPoints = GetTurningPoints(pattern, MAPeriod);
        DataPoint[] targetPoints = GetTurningPoints(targetSeries, MAPeriod);
        patternSlopes = GetSlopesBetweenTurningpoints(patternPoints);
        targetSlopes = GetSlopesBetweenTurningpoints(targetPoints);
        //start from different points in the longer series and evaluate the resulting match
        deviation = 0;
        if (patternPoints.Length>=targetPoints.Length)
            for startingPoint = firstPointInTarget to lastPointInTarget step 1
                for index = 0 to patternLength step 1
                    deviation += Difference (targetSlopes[index+startPoint],
                                                                patternSlopes[index]);
                next
                StoreDeviationForThisMatch();
                deviation = 0;
            next
        else
            //iterate over the pattern sequence instead of the target sequence
    next
```

In the following the method, several parameters and introduced variations are described in greater detail. For instance, in Algorithm 12 the search pattern and the source series are both smoothed with the same MA period which is not generally desirable.

5.3.2.3 Determination of turning points

In the above overview, it was said that the crossing points between the MA curve and the original curve are used as the turning points between which the slopes are computed and subsequently compared. In fact, this has the downside that the turning points are set after the actual turn of the trend. Especially for longer MA periods, this can be very decisive. The rule is that the greater the MA period, the later can the actual reversal be detected. The advantage is that we of course have the possibility to track back the series to the actual point where the trend reversed: in case of an up-trend turning into a down-trend this is the highest data point since the last trend reversal or the beginning of the series, for a down-up trend reversal, it is the lowest point.

Out of these considerations, several modes are introduced for determining the turning points. Ultimately, the user can choose which mode to apply. The modes available are:

- **Extremum** – Take the extremum between the current and the last crossing point.
- **CrossingPoint** – Take the value of the crossing point directly.
- **AvgExtremumAndCrossingPoint** – Build an average between the extremum and the crossing point. Averages the value as well as the timestamp.
- **ExtremeValuesAverage** – Compute an average of the highest/lowest n percent of values. Set the timestamp to the point of the absolute extremum. This technique is used to straight out extreme outliers.

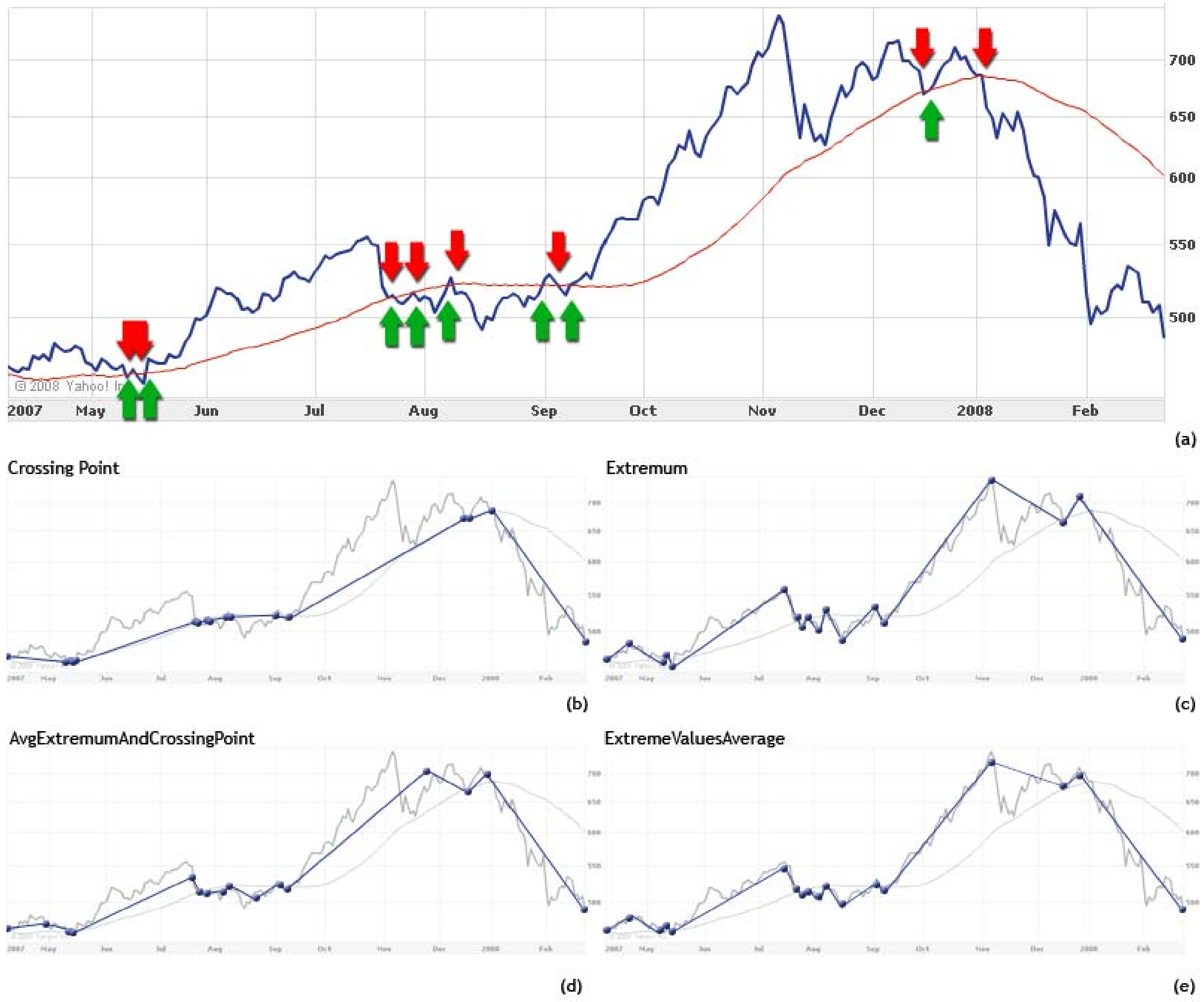


Figure 62: Impact of different turning point modes

Figure 62 illustrates the impact of these turning point modes. In Figure 62a, the MA crossing points are marked for an MA period of 50. It can be seen that such a relatively large MA period smoothes out big movements, such as the price high in November and the subsequent temporary decline.

In Figure 62b, the series of turning points resulting from the turning point mode “CrossingPoint” is shown. The resulting curve is very smooth; many smaller movements nearly disappear completely. Reversal points are mostly far behind the actual turning point of the curve.

Figure 62c shows the use of the turning point mode “Extremum”. The mode emphasizes the original movements of the curve, but in case of many short subsequent changes such as in August/September, the result is a sequence of rough spikes, which may be undesirable.

In Figure 62d, the “AvgExtremumCrossingPoint” turning point mode is applied. In this case the resulting turning points must not necessarily be points on the original curve. Turning points are set later than the actual curve turns. Smaller up/down alterations are smoothed better.

Finally, Figure 62e shows the result of applying the “ExtremeValuesAverage” technique. The result is similar to the “Extremum” technique. Yet, extreme outliers are not used directly. This may be of advantage in case of extreme short term variations which should not be considered for the overall trend.

Algorithm 13 demonstrates the combined computation of the moving average and the extraction of turning points depending on the turning point mode.

Algorithm 13: Computation of turning points based on MA crossing points

Input: TimeSeries searchedSequence, int period (the MA period), TurningPointMode mode, int extremeValuesAveragePercentage

Output: TimeSeries turningPoints

Variables: Indices *indexOverall* (index of the current data point in the sequence), *indexInternal* (index within the internal array of last values for MA computation): *lastValues* (array of last values used for MA computation), *curSum* (current sum of MA values), *replacedValue* (last value still used for the MA's sliding time window - the value that is replaced when moving the time window forward), *maPoints* (calculated series of MA points), *pointsFromLastCrossingPoint*, *penultimatePoint, intersection*;

```
for each point in searchedSequence
        //always add the first and the last point
        if  indexOverall = 0 or indexOverall = searchedSequence.Lenth - 1
                turningPoints.AddDataPoint(point);

        if  indexInternal = period
                indexInternal = 0;

        //update the next values for the MA period computation
         replacedValue = lastValues[indexInternal];
        lastValues[indexInternal] = point.Value;

         //prior to reaching the first MA period, simply sum up all values
        if  indexOverall  <= period – 1
                curSum +=point.Value;

        if  indexOverall  = period – 1
                maPoints.AddDataPoint(point.TimeStamp, curSum / period);

        //for subsequent MA value remove the last value and add the new one
        else if  indexOverall >= period
                curSum = curSum - replacedValue + point.Value;
                maPoints.AddDataPoint(point.TimeStamp, curSum / period);

         //check if the ma line intersects with the data point series. the function Intersects(a,b,c,d)
         //returns the intersection of two lines a-b and c-d
         intersection = Intersects(maPoints[penultimatePoint],maPoints[point.TimeStamp],
                                searchedSequence [penultimatePoint], point.Value);

        if  intersection = Intersection.FromBottomToTop
                if  mode = TurningPointMode.Extremum
                        turnPointToAdd = FindLastMaxValue(pointsFromLastCrossingPoint);
                if  mode = TurningPointMode. CrossingPoint
                        turnPointToAdd = point;
                if  mode = TurningPointMode. AvgExtremumAndCrossingPoint
                        turnPointToAdd =Avg(FindLastMaxValue(pointsFromLastCrossingPoint),
                                                point);
                if  mode = TurningPointMode. ExtremeValuesAverage
                        turnPointToAdd = (point.TimeStamp, GetAvgOfHighestValues(
                                pointsFromLastCrossingPoint, extremeValuesAveragePercentage));

        if  intersection = Intersection.FromBottomToTop

                 //equivalent to intersection from top to botton, yet, use functions
                //"FindLastMinValues" and  "GetAvgOfLowestValues" respectively, instead of the
                //max value functions
        if intersection ≠ Intersection.None
            turningPoints.AddDataPoint(turnPointToAdd);
            pointsFromLastCrossingPoint.Clear();

        pointsFromLastCrossingPoint.AddDataPoint(point);
        penultimatePoint = point.TimeStamp;
        indexInternal++;
        indexOverall++;
    next
    return turningPoints;
```

5.3.2.4 Modes and parameters for MA smoothing

In Algorithm 12 on page 71 both the pattern series and the searched series are smoothed with the same MA period in each pass. This technique may be appropriate when comparing two independent and potentially unknown time-series. In other cases, for instance when the pattern is a well-known formation such as the "W-pattern" illustrated in Figure 60 on page 70 which also has a known time scaling, the analyst may wish to set the MA period so that the turning points exactly emphasize the most important characteristics of the input pattern. Hence, separate control of the determination of turning points in the reference pattern sequence and the target sequence is required. In order to provide utmost flexibility, the algorithm in addition provides two modes:

- **EqualPeriodAlways** – Method as used in Algorithm 12. In different passes both series are smoothed with the same MA period.
- **VaryingPeriod** – The MA periods for reference pattern and target sequence are varied independently and compared to each other. The user can set the minimum and maximum MA periods and the precisions (step lengths) for both series independently. Algorithm 14 illustrates the resulting iteration loops.

Algorithm 14: Iterations for varying period MA smoothing

Input: TimeSeries pattern, TimeSeries targetSeries

Variables: Arrays of time-series data points *patternPoints*, *targetPoints*; Indizes *MAPeriodPattern*, *MAPeriodTarget*

```
     //iterate in multiple passes through the data with varying MA period
    for MAPeriodTarget = minPeriodTarget to maxPeriodTarget step precisionTarget
        for MAPeriodPattern = minPeriodPattern to maxPeriodPattern step precisionPattern
            DataPoint[] patternPoints = GetTurningPoints(pattern, MAPeriodPattern);
            DataPoint[] targetPoints = GetTurningPoints(targetSeries, MAPeriodTarget);
        next
        //Get slopes and compute deviations
        …
    next
```

With this mode it is also possible to set the MA of the reference pattern to a fixed value, in case of minimum period equaling maximum period.

5.3.2.5 Anchoring the match

For certain applications it might be of interest to anchor the match, i.e., to guarantee that the match either starts with the first value of the searched sequence or ends with the last value or both. Especially in finance, anchoring the end of the match is important, as users may want to find stocks or other instruments where a search pattern is *currently* emerging, and not those where the pattern occurred somewhere in the past but now the price moved to other directions.

Anchoring the match at the start can be achieved by just comparing the match starting with the first turning point. Practically this just leaves out the loop shifting the shorter series over the longer one. Instead, it only compares the first n slopes between the turning points, whereby n corresponds to the length of the shorter sequence, which is mostly the search pattern.

Equivalently, anchoring the match at the end can be done by leaving $m - n$ values at the beginning of the longer sequence unconsidered, whereby m is the length of the longer sequence, n the length of the shorter sequence.

Anchoring both, start and end of the match is yet a special case. Trying to apply the same algorithm as above fails, because the number of turning points may vary, and it is impossible to choose which turning points should be used. Also, forming combinations of all possible turning point usages is inappropriate. Due to these considerations this setting is treated completely separate, using a different technique for the matching. In this case, the original algorithm of Toshniwal and Joshi [50][51] is applied.

The idea is that when wanting to anchor the start and the end of the match, the search pattern should be virtually stretched over the compared sequence, and differences should be assessed. Even if this is the simplest case of time-series comparison it is still necessary to find an efficient way to perform this comparison. The slope-model is applied as follows: First, the two time-series are normalized on the time-axis. Then both series are subdivided into equally-sized slices. The density of these slices is configurable. In a next step, the slopes between the curve points at each slice are computed, and finally pair-wisely compared.

Figure 63 illustrates the described process. The simple example also underlines a major problem when using this approach: The slicing is a kind of resampling to reduce the complexity and number of slopes to compare. It has the purpose of emphasizing the overall movement instead of small fluctuations, which always decrease the similarity score. As a consequence, the success of the comparison strongly depends on the chosen sampling rate for slicing. If it is not appropriate, important parts of the curve may just be smoothed out. As can be seen in the example, the W-formation is less clear after the slicing. For the red line it nearly vanishes. The effect can be reduced by again iterating in multiple passes over the series with varying sampling rates, but not completely omitted.

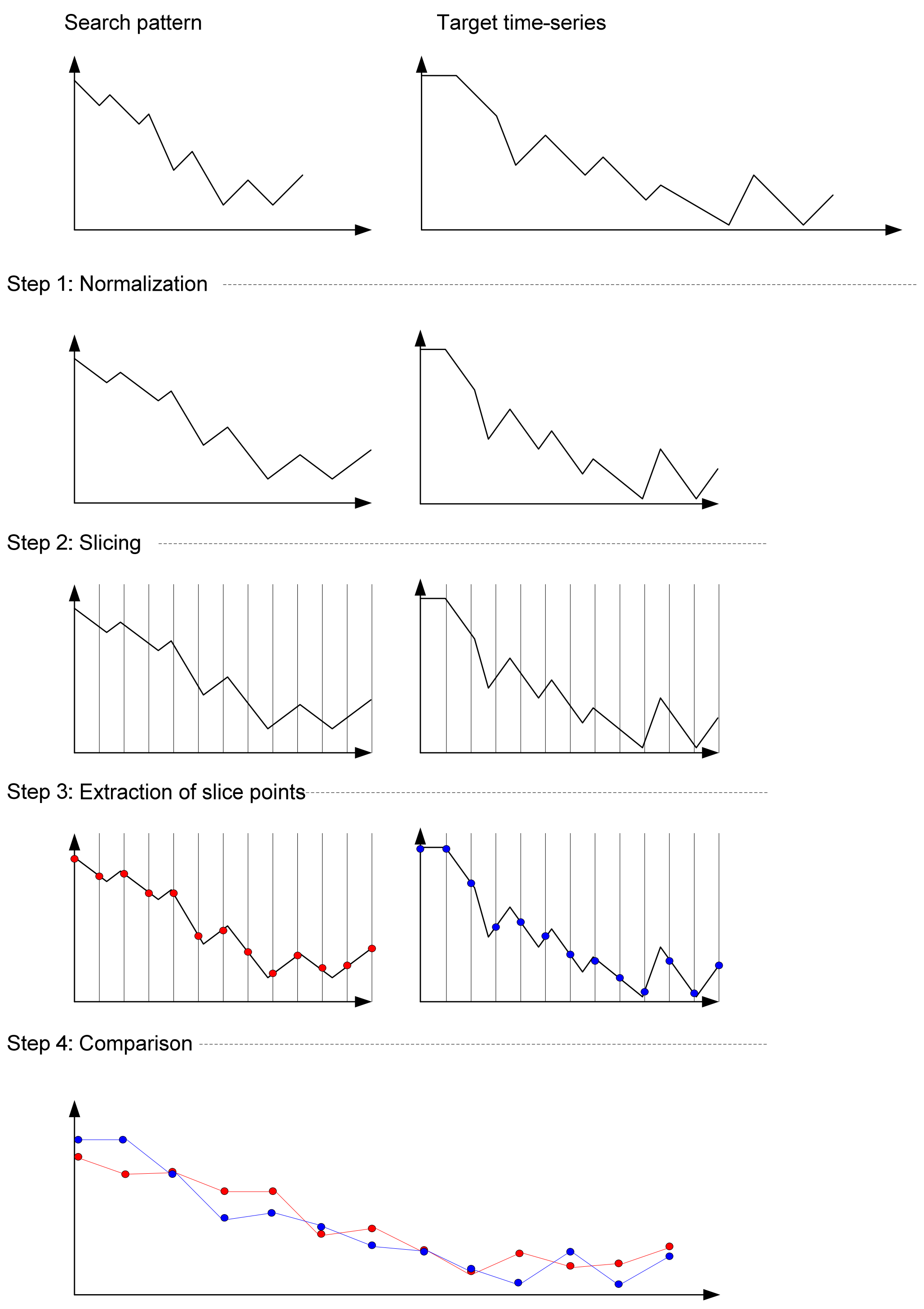


Figure 63: Process of slope comparison with regular slicing

5.3.2.6 Modes for comparing the slopes

In case of the default mode for the matching, i.e. the computation of turning points based on MA crossings and the comparisons of slopes between them the next question arising is which slopes to compare to each other. The simplest method is to always compare the slopes between subsequent points only. This technique is also applied by Toshniwal and Joshi in their original publication. One problem arising from these slop-by-slope comparisons is the following: In combination with varying distances between the turning points, it is possible that the method leads to false positives, i.e., matches that are in fact not equal or even similar. An example makes it clear. In Figure 64 two series are shown, which have exactly matching slopes from one point to the other. Yet, due to the fact that they are not equally sampled, they are not guaranteed to be equal, as the example shows. A value-by-value comparison would assess these series as being equal.

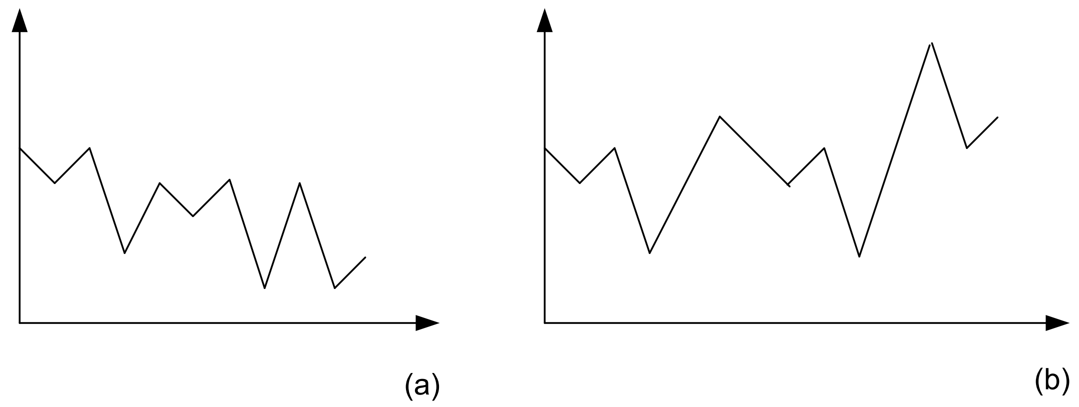


Figure 64: Two unequal series with equal slopes between turning points

In consequence of these considerations, different locality modes for the slope comparison are introduced. The offered modes are:

- **None** – In this mode only neighbouring turning points are compared, with the above said shortcoming. Requires the fewest comparison operations.
- **Local** – In this mode, slopes between each point and n subsequent points are compared, whereby n is a user-configurable parameter.
- **Global** – In this mode the slopes from each point to each other point are considered. This causes the most global view onto the series.
- **WeightedGlobal** – In this mode, also all slopes are considered as in the global mode, but the closer two points are, the higher is the slope deviation weighted.

The modes virtually span different grids of slope lines over both the reference pattern and the compared sequence. Figure 65a illustrates this virtual grid of considered slopes for local comparison mode with the number of points taken into account set to 3. In comparison, Figure 65b shows the considered slopes in case of global comparison mode.

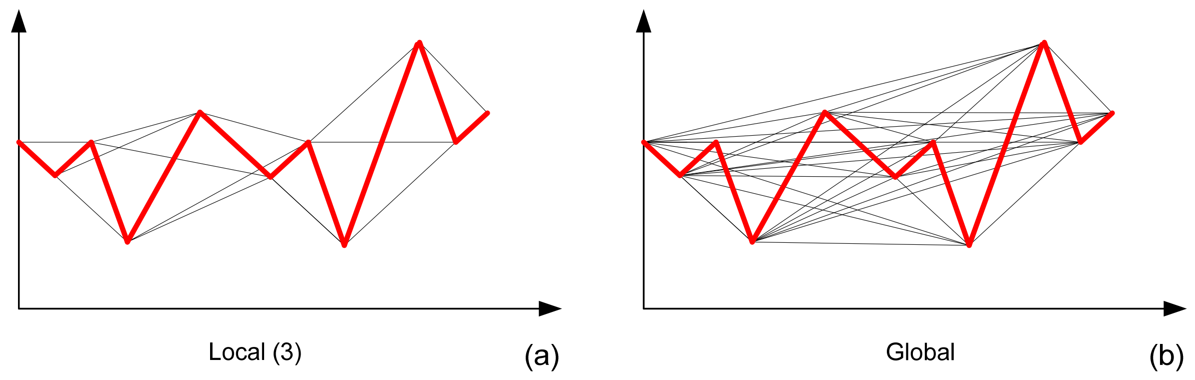


Figure 65: Considered slopes in local and global mode

The figure underlines the major difference and value of both techniques: In the example sequence, the two lowest points in the series have the same value. In the finance domain, one of the main application areas for these techniques, such a pattern could be interpreted as a resistance level, a point where a down-side movement stops more than once. In the local mode, the slope of 0 degrees between these 2 points remains considered, while in global mode it gets equally weighted as all other slopes and is taken into account. In other application areas of course this might not be desirable.

Please note that for the special example of resistance levels mentioned above it would be possible to go even one step further and allow the user to emphasize such levels in the search pattern. Yet, a specific modelling of time-series constraints is out of scope for the on-hand thesis.

5.3.2.7 Time weighting

The example in Figure 62 on page 73 shows that with the turning point extraction technique it might occur that the time spans between the extracted points vary strongly. For instance, for a longer subsequence that moves straight in one trend direction, only one turning point at the beginning and end will be extracted, whereas in other sequence areas with short-term fluctuations many consecutive turning points with only minimal time spans in between may emerge from the extraction process. In Figure 66 two series of this kind are shown. In fact, sequences (a) and (b) only vary in the second curve point, which means only two of eight slopes show a deviation.

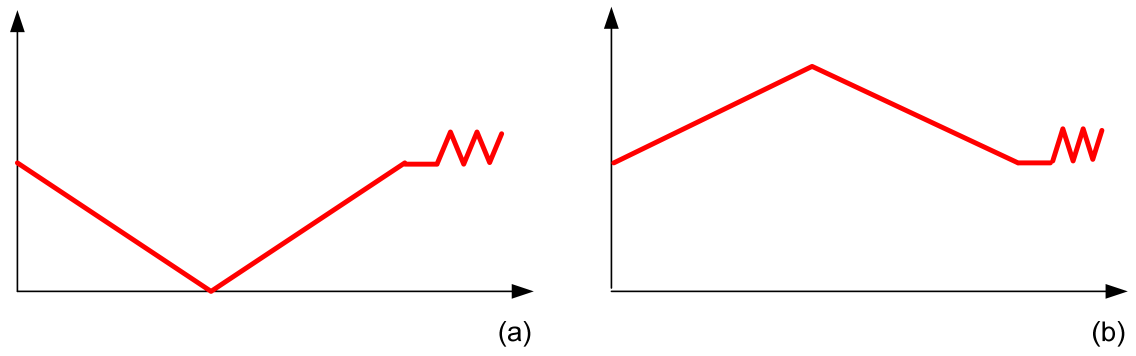


Figure 66: Two sequences with many equal slopes

Yet, for the assessment of similarity this might be misleading, as the overall curve movement is not similar. Therefore, we introduce an optional time weighting as an additional parameter for the algorithm. If time weighting is activated, each deviation is weighted proportionally to the relative length of a slope in relation to the complete sequence's length. For the above example, this would result in a very high weight for the first 2 slopes and very low weights for the subsequent slopes, resulting in a high dissimilarity of the two sample series.

5.3.2.8 Search patterns

Implicitly, throughout the above considerations it was assumed that the input for a similarity search is a time-series on its own. This results from the fact that when using an event sequence as the pattern, it is natural that it's interpreted as a time-series. Anyway, in many cases the user will want to search for a specific and most often simplistic pattern such as for example the "W-formation". It is unnecessary in this case though, to use a time-series as pattern and compute turning points based on different MA periods. Instead, the data points as such should be interpreted as the turning points forming the input pattern, and this pattern should be searched at different scales (by varying the MA) in the searched series.

Therefore, the time-series comparison library implemented in the course of this work offers also a method to compare such a pattern directly. The base algorithm remains the same, but instead of computing any turning points for the reference pattern, the pattern's data point are passed directly to the slope computation.

In addition, such a simplified pattern can be attached with a minimum and maximum duration it may span. The rationale is that it may be useful to restrict the search in a way that for instance a certain pattern must span at least a couple of weeks but not more than several years.

5.3.2.9 Similarity computation

Up to this point, we discussed slope deviations rather than similarity. Yet, in order to integrate the time-series model with the event sequence similarity matching, results have to be translated to a unified measure. This measure is a similarity score between 0 and 1. In case of slope deviations, it is possible to determine for each slope the maximum possible deviation, which is 180 degrees.
Let $slopes_p$ be the slope values in degrees between the turning points extracted from the reference pattern and $slopes_t$ be the slope values in degrees between the turning points in the compared target sequence. A similarity score sim can then be computed as an inverse of the ratio of a match's sum of slope deviations to the sum of theoretical maximum deviations with the following formula:

$$sim(ref, seq) = 1 - \frac{\sum_{i=1}^{n} \left| slopes_p[i] - slopes_t[i] \right| * w_i}{\sum_{i=1}^{n} 180 * w_i}$$

Formula 24: Time series similarity computation

Formula 24, w_i refers to a specific weight for each pair of slopes. Depending on the algorithm settings, its value is computed from the length of the respective slopes in relation to the complete sequence length (time weighting, see section 5.3.2.7) and, in case of weighted global matching mode a linear factor that weights deviations of distant slope pairs less high than directly corresponding pairs.

5.3.2.10 The STSimilarity library

The implementation was done in C# and the resulting .dll can be integrated into other applications rather than the on-hand event similarity as well. The most important interface methods and configuration parameters are summarized in Appendix A – The STSimilarity library.

5.3.2.11 Possible extensions

This section briefly discusses some possible extensions and improvements that could be made in order to enhance the applied time-series model.

Volatility indicator
A volatility indicator such as the rate of change (ROC) or Bollinger bands could be used to avoid falsely detected trend reversals. In this way smaller fluctuations could potentially be omitted for the matching if desired. Thus, it depends on what the user wants to find and if these movements might be relevant.

Constraint modeling
It was already briefly discussed that in order to emphasize certain characteristics in a search patter like a resistance level, enhanced possibilities for search pattern modeling would be a plus. Such constraints could be

certain slopes which are required to be in a given value range, or also tolerance levels for single data points or slopes.

Enhancements for anchored start and anchored end

In case of a configuration set to anchor the match at the start *and* at the end of the series, the algorithm has to be adapted. Currently, a simplified version of the algorithm is used, that performs a constant slicing of the time-series and compares the slopes pair-wise. It shows that this method is very sensitive to the configuration parameter of the slicing rate and may omit important aspects of a time-series. Originally, this was the reason for the introduction of the trend-reversal method.

A possible enhancement for this special case of full-sequence matching could be to use the MA smoothing and turning point extraction also in this case, but to intelligently thin-out the turning points to an equal number of points. Hereby, turning points which are positioned nearly on the straight line between the prior and the succeeding point could be omitted. Alternatively it would be possible to match subsequences of turning points onto the target sequence and omit those turning points not required. It is yet a challenging task to find efficient ways for discovering the best partial matches.

5.3.3 Asymptotic runtime

The following considerations illuminate the asymptotic runtime of the proposed slope-based time-series similarity comparison algorithm between two time-series T_p (pattern sequence) and T_t(target sequence).

5.3.3.1 Turning point computation

The first operation to consider is the extraction of turning points. Algorithm 13 shows the turning point extraction algorithm, taking a time-series T of length n, the period for the MA smoothing p_{ma} and turning point mode input. With the loop on line (1) it iterates once over the complete series. The MA period p_{ma} does not influence the runtime, as the algorithm continuously remembers the last value to be subtracted from the MA sum and solely adds the new ones rather iterating again over the last p_{ma} number of values. The selected turning point mode is also not decisive for the runtime. The turning point modes “AvgExtremumAndCrossingPoint” and “ExtremeValuesAverage” add, compared to “CrossingPoint” and “Extremum” an additional constant factor to the runtime for the computation of the average value. In total, this still results in an asymptotic runtime of $\Theta(n)$ for the extraction of turning points based on MA curve crossings.

5.3.3.2 Slope computation

The runtime of the slope computation depends upon which slopes should be extracted. In case of locality mode “None”, i.e. only subsequent slopes should be compared to each other, the runtime is $\Theta(n)$. For the comparison mode “Local”, always the slopes between a point and m subsequent points are computed. As m is a constant (user-configurable) factor, the asymptotic runtime is still $\Theta(n)$. For the modes “Global” and “WeightedGlobal”, for each point the slopes to all other points are required, i.e. $\frac{1}{2} * n^2$ slopes and thus the runtime is in $\Theta(n^2)$.

5.3.3.3 Total runtime

MA smoothing mode “EqualPeriodAlways”

Algorithm 12 summarizes the algorithmic steps in case of comparing T_p and T_t with the MA smoothing mode “EqualPeriodAlways”. It means that the MA period is period is varied, but always equally for T_p and T_t. With the loop on line (1), the algorithm iterates $\frac{p_{ma\_max}-p_{ma\_min}}{precision}$ times, whereby $precision$ is a user-configurable step-length and p_{ma_min} as well as p_{ma_max} are the lower and upper bounds for the moving average periods, defined by the user. On lines (3) to (6) the turning point and slope computations are done in each loop. Subsequently, the algorithm generates multiple matches, by “shifting” the shorter sequence of turning points over the longer sequence of turning points and computing the underlying slope deviations. In the following, n_p denotes the number of points in the T_p and n_t the number of data points in T_t. The total runtime r of the algorithm is

$$\begin{aligned} r = \frac{p_{ma\_max} - p_{ma\_min}}{precision} * \left(n_p + n_t + f_p n_p + f_t n_t + \left|f_p n_p - f_t n_t\right| * \min\left(f_p n_p, f_t n_t\right)\right) \\ = \Theta(n_p + n_t + \left|n_p - n_t\right| * \min\left(n_p, n_t\right)) \end{aligned}$$

Formula 25: Asymptotic runtime of base time-series similarity algorithm

with f_p and f_t denoting the factor by which the original sequences are “thinned out” by the turning point extraction process. This factor depends on the MA period (the smaller the period, the more turning points in typical data sets) and the characteristics of the data. In case of the locality mode for slope comparison set to “Global” or “WeightedGlobal”, according to above considerations the asymptotic runtime is given as:

$$r = \Theta({n_p}^2 + {n_t}^2 + \left|n_p - n_t\right| * \min\left(n_p, n_t\right))$$

Formula 26: Asymptotic runtime when comparing slopes globally

5.3.3.3.1 Best case

The best case of the algorithm is when pattern and target sequence both have data points on a straight line, as the given example in Figure 67.

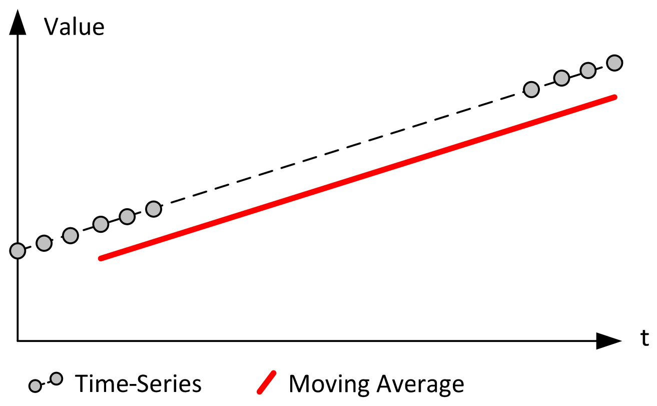


Figure 67: Best case for time-series similarity runtime

In this case, the MA line never crosses the curve, and only the start and end points will be added. In total, only one pair of slopes needs to be compared, and the runtime is solely the time for the turning point extraction, i.e. $\Theta(n)$. Thus, we can say that the complete algorithm has a lower bound of $\Omega(n)$.
A second best-case scenario is given if $f_p n_p = f_t n_t$, i.e. the numbers of turning points extracted from both sequences are equal.

5.3.3.3.2 Worst case

The worst case is the case where the utmost possible number of turning points is extracted. An example is sketched in Figure 68. The maximum number of turning points for a series of length n and an MA period p_{ma} is $n - p_{ma} + 1$.

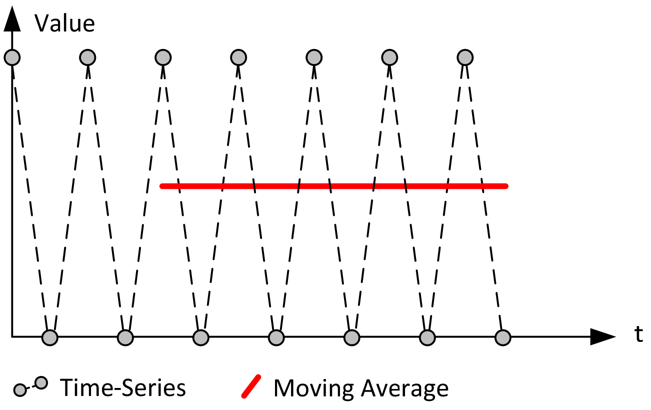


Figure 68: Worst case for time-series similarity runtime

In addition to the worst case in terms of the number of turning points, for comparing the slopes, the worst case is given when $\frac{|f_p n_p - f_t n_t|}{2} = \min(f_p n_p, f_t n_t)$, i.e. the number of turning points extracted from one sequence is exactly twice as high as the number of turning points from the second sequence. If n is the length of the longer sequence, the loop for shifting the shorter sequence over the longer one and comparing the slope deviations requires $\frac{1}{2}n * \frac{1}{2}n$ iterations. Thus, the upper bound of the algorithm is $\mathrm{O}(n^2)$.

MA smoothing mode "VaryingPeriod"

In Algorithm 14, the iterations in case of the MA smoothing mode "VaryingPeriod" have been introduced. In terms of the runtime, this means that the steps of turning point extraction, slope computation and slope deviation assessment are not executed $\frac{p_{ma\_max} - p_{ma\_min}}{precision}$ times, but $\left(\frac{p_{ma\_max} - p_{ma\_min}}{precision}\right)^2$ times.[7] In terms of the asymptotic runtime, this is still a constant factor which does not depend on the number of data points, so the algorithm's upper and lower bounds are not influenced. In practice however, for typical data series these modes have high impact on the performance.

5.3.4 Results and performance

The performance and matching results of the time-series model have been evaluated separately from the event similarity matching process in order to simplify testing and enable targeted assessment of outcomes. The results are summarized in Appendix B – Evaluation results time-series similarity model.

[7] To be precise, it is $\frac{p_{ma\_\max\_pattern} - p_{ma\_\min\_pattern}}{precision\_pattern} * \frac{p_{ma\_\max\_target} - p_{ma\_\min\_target}}{precision\_target}$ times, as we allow specifying separate MA period ranges for both the target and the pattern sequence. Yet, it is irrelevant for the considerations on the asymptotic runtime.

5.3.5 Integration into base similarity algorithm

In order for the time-series model to be fully integrated into the base similarity algorithm, the following requirements have to be met:

- Time-series similarity is just one of many attribute similarity factors. It must be possible to add the matching results with equal weight as other attribute similarities and event sequence characteristics to the overall similarity.
- Multiple attributes may use time-series similarity as the attribute similarity technique. This means in particular, that also the results of multiple time-series similarity comparisons must be combinable.

We propose two modes of execution for performing an event sequence similarity search with at least one attribute utilizing numeric sequence similarity.

5.3.5.1 Mode 1 - Post-matching execution

In mode 1, post-matching execution, first the base algorithm is applied, including the processing of all constraint blocks. For each ranked match above the similarity threshold, the series of attribute values from the chosen events for the match is extracted. This series is compared to the attribute series extracted from the search pattern, and the similarity result is weighted and added to the overall similarity.

For this approach, the time-series algorithm is configured to perform full-sequence matching as we assume that the base algorithm selects the events of the sequence that must be taken into consideration. Figure 69 illustrates the process. As can be seen from the illustration, the approach guarantees that for the time-series matching an equal number of data points is available in the searched sequence and the reference sequence after executing the base algorithm. Therefore, these points can be understood directly as the turning points of the series and only the slopes need to be compared.

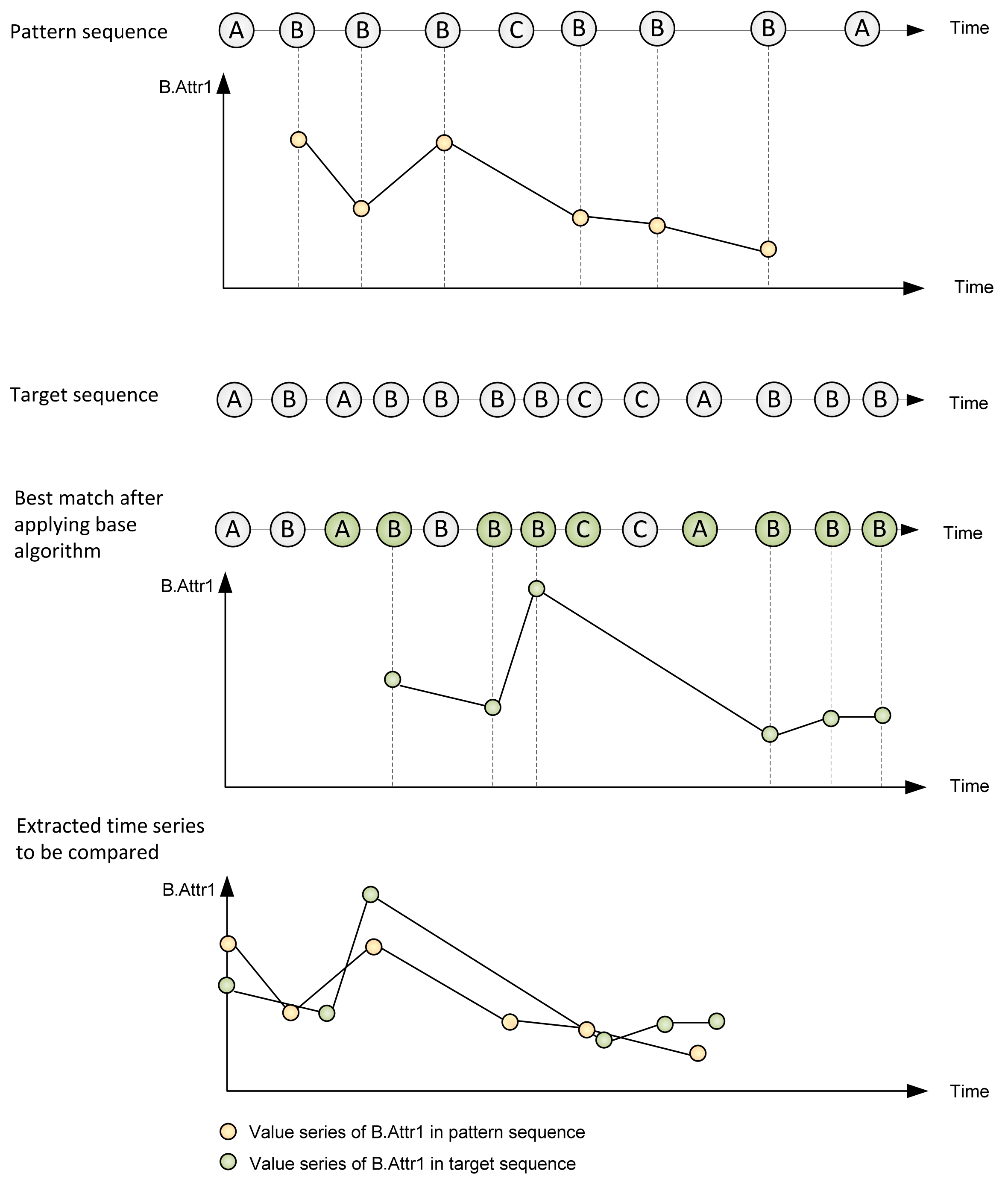

Pattern sequence
A B B B C B B B A
Time
B.Attr1
Time
Target sequence
A B A B B B B C C A B B B
Time
Best match after applying base algorithm
A B A B B B B C C A B B B
Time
B.Attr1
Time
Extracted time series to be compared
B.Attr1
Time
Value series of B.Attr1 in pattern sequence
Value series of B.Attr1 in target sequence

Figure 69: Post-matching mode for numeric sequence similarity attribute technique

The post-matching mode implies a certain limitation in the execution of complex similarity searches with type matching and time-series similarity. The problem at hand is that in case of a high weight of the attribute to be evaluated with time-series similarity, cases might exist where a chosen subsequence from the time-series matching with the costs of types to be omitted could still be better than a subsequence chosen by the base algorithm and all other constraint blocks. In order to avoid these effects, it would be required to provide an implementation of the time-series algorithm that is able to process constraint blocks and type deviations, in order to execute the type matching coupled with time-series similarity and constraint blocks all in one. On the other hand this contradicts with our approach to keep attribute techniques atomic and exchangeable. Another issue in that case is the impossibility to apply the extended time-series algorithm for many attributes at the same time. We have therefore chosen to take this limitation into account and provide the alternative pre-matching execution mode which is able to cover many of these cases.

5.3.5.2 Mode 2 – Pre-matching execution

Mode 2, pre-matching execution assumes that type deviations as well as related constraint blocks such as order constraints are less important for the matching compared to the attributes for which time-series similarity is applied. Considering an event sequence search pattern from the finance domain proves this assumption to be adequate in many cases.

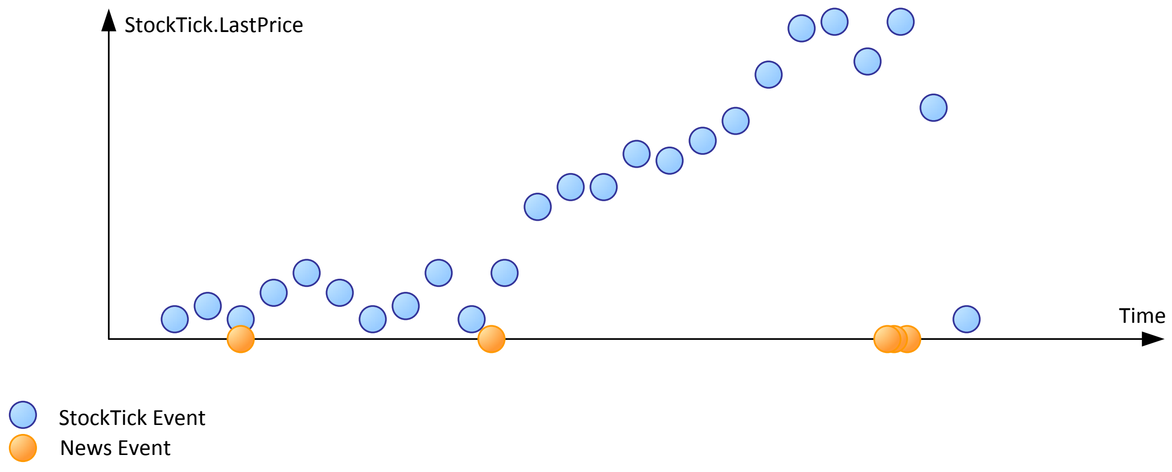
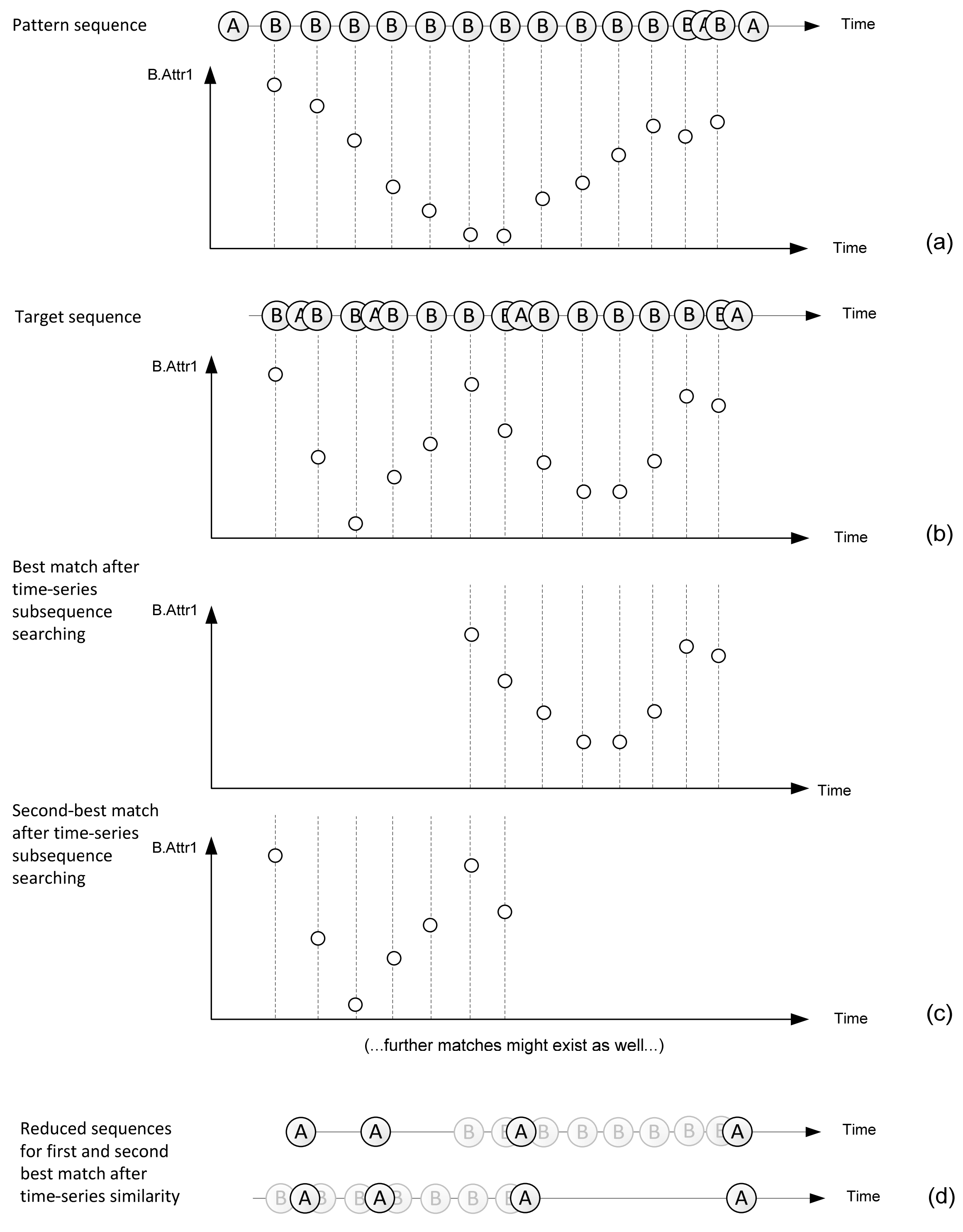


Figure 70: Example of a search pattern from finance domain

For the pattern in Figure 70, the type order for stock tick events is practically irrelevant, as these events occur regularly, in the frequency of the data captured. The only aspect important regarding the event types is the occurrence of news events in relation to the pattern of the *LastPrice* attribute of the *StockTick* events.

5.3.5.2.1 Pre-matching one time-series attribute

In many cases exactly one attribute of one event type is chosen for the time-series matching. Figure 71 illustrates the matching process in that case. First the series of attribute values is extracted from both the reference pattern (Figure 71a) and the target sequence (Figure 71b) by taking the attribute values of all events of the concerned event type. For these two series, the time-series similarity algorithm is applied. It is configured to return a list of possible matches (Figure 71c). Matches below the similarity threshold are omitted. Subsequently, for each of these matches the events are chosen from the searched sequence (Figure 71d), and passed to the base algorithm to perform the matching process. Results from the base algorithm and the time-series similarity are then weighted and combined.



Pattern sequence
A
B
Time
B.Attr1
Time
(a)
Target sequence
Time
B.Attr1
Time
(b)
Best match after time-series subsequence searching
B.Attr1
Time
Second-best match after time-series subsequence searching
B.Attr1
Time
(c)
(...further matches might exist as well...)
Reduced sequences for first and second best match after time-series similarity
Time
Time
(d)

Figure 71: Pre-matching mode for numeric sequence similarity attribute technique

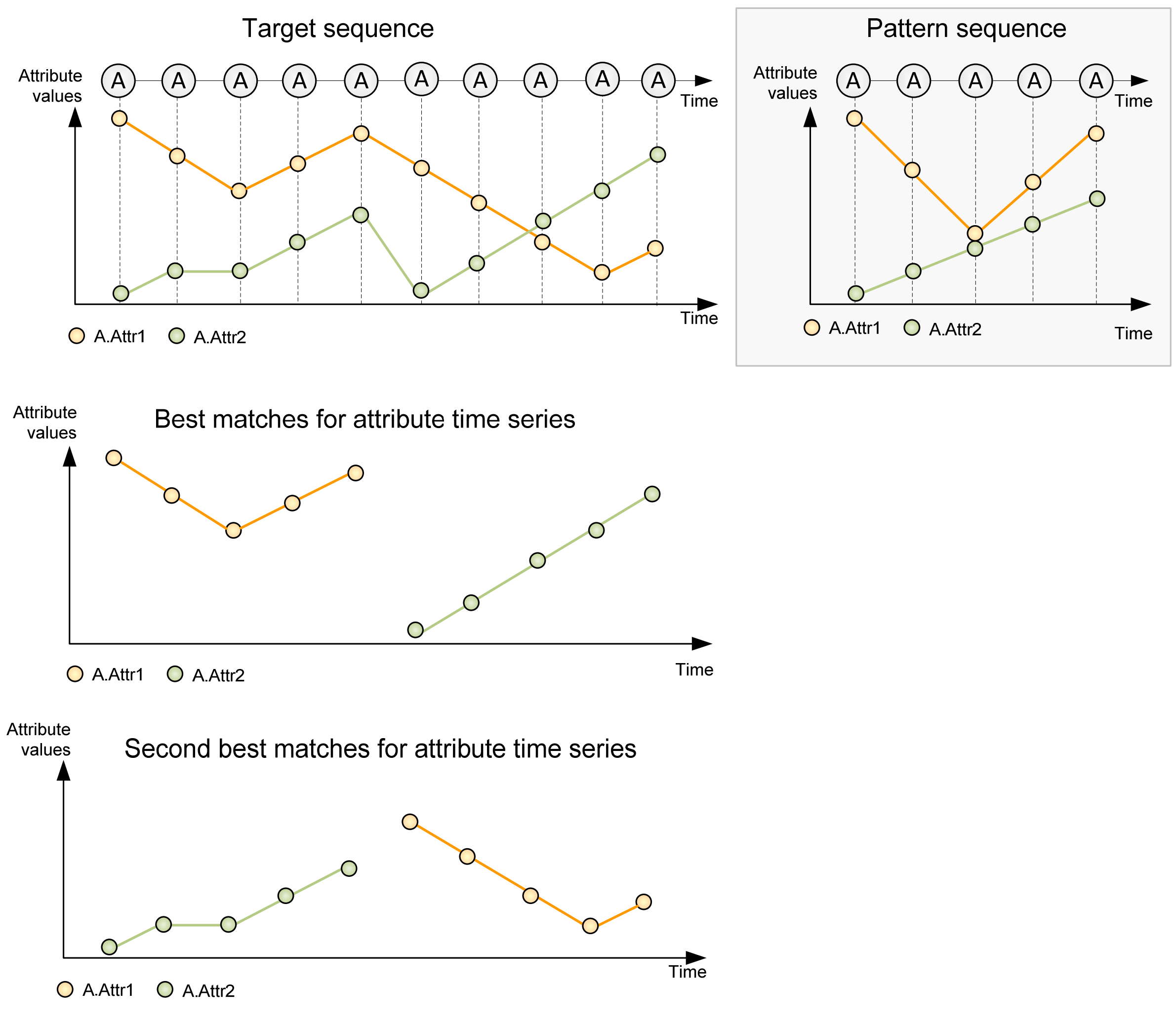
Typically, in case of pre-matching the weight of type deviations for the concerned event type is set to zero. Otherwise after the subsequence selection process of the time-series algorithm the omitted events might sharply reduce the overall similarity, which is undesired in most cases. This is indicated in Figure 71d by greying out events of type B which are subsequently indecisive for the type matching process. Yet, other attribute similarities might be evaluated on these events now as well.

Figure 71d shows that events of other event types are not cut-off when selecting a subsequence according to the results of the time-series matching. The rationale of the approach is not to implicitly omit other events that are not directly concerned by the time-series matching. Rather, the fact that they might now occur at points in time having a big deviation to the found time-series match will decrease the similarity score when performing the base algorithm, but all information is considered.

5.3.5.2.2 Pre-matching multiple time-series attributes of the same event type

The user might select numeric sequence similarity as the attribute technique for multiple event attributes of the same event type. For instance, *StockTick* events might contain the attributes *LastPrice* and *Volume*. A similarity search may consider both attributes at the same time in order to find sequences where both the price and the traded volume are correlating. In that case, first multiple results of the time-series matching have to be combined with each other. The challenge about this combination is that of course only the same subsequences of events are allowed to be combined. Otherwise a virtual match is created that utilizes one set of events *or* another at the same time. Please note that all of the following considerations apply only in case of subsequence searching. If the user requires matches to be anchored at the first and last data points, the matching process will return exactly one attribute similarity score per attribute, and from these scores, a weighted average is computed.

The problem in case of subsequence searching is illustrated in Figure 72. For two event attributes of event type *A*, namely *Attr1* and *Attr2* the time-series similarity matching is applied. In the example, the best and second best matches from the time-series comparison are shown. Apparently, the two sequences of best matches cannot be combined directly, as they result from attribute value sequences extracted from distinct sets of events.



(…)

Figure 72: Pre-matching in case of multiple time-series attributes

In order to figure out the combined best matches, we apply the following process:

(1) Build m-tuple wise combinations of solutions from the m sets of subsequence matches, whereby m is the number of different attributes for which time-series similarity is applied (e.g. $m = 2$ in the previous example).

(2) For each tuple: Get the set of common events S_{common} based on the subsets of events from which originally the time-series attribute values have been extracted (in the following denoted as S_1 to S_n). $S_{common} = S_1 \cap S_2 \ldots \cap S_n$. Depending on the event sets, apply one of the following operations:

a. $S_{common} = \emptyset$ (The event sets are disjoint): Select the attribute series with the highest similarity score. Extract the attribute values for all other attributes and recompute time-series similarity in terms of full-sequence matching.

b. $S_1 = S_2 .. = S_n = S_{common}$ (The event sets are equal): Compute the similarity as a weighted average from the m matches in the tuple.

c. $S_{common} \neq \emptyset \; and \; S_{common} \neq S_1 \; or \; S_2 \ldots or \; S_n$ (The event sets are overlapping): For each entry i in the tuple perform one of the following operations:

   i. $S_i = S_{common}$: The already computed similarity score is added to the overall similarity score (as one additional factor in a weighted average).

   ii. $S_i \neq S_{common}$: The original value set for which the similarity score has been computed is different to the common value set. Extract the values of the concerned attribute and recomputed the similarity in the sense of full event sequence matching.

S_{common}, together with all events of other event types is then passed to the base similarity algorithm for the rest of the matching process.

5.3.5.2.3 Pre-matching multiple time-series attributes in different event types

The execution in case of multiple time-series attributes chosen for multiple different event types is similar to the case of one event type. In addition to the technique presented in the previous section 5.3.5.2.2, also the results from different event types have to be combined by simply forming permutations of matches.

As denoted above, in case of time-series similarity applied to the events of one event type, for a possible match all events of the other event types are added and not cut-off. This implies for each permutation, that it unions the subsequence of events of each event type. For types for which no time-series pre-matching has been performed, all events are added to the permutation. A temporary similarity score is computed based on the similarity of each subsequence (weight average).

5.4 Generic similarity

In this section we discuss the integration of custom event similarities into the existing framework. In the SENACTIVE InTime system, typically a default implementation class called *EventObject* is used for events. For certain features or to ease event processing, also custom implementation classes derived from *EventObject* may be used. We provide an interface *ISimilarityComparable* with a function *Compare()*. It is expected to get two objects of the *ISimilarityComparable* type as input and return a value between 0 and 1. This applies to complete event objects, but also to attributes with custom runtime types. The purpose is mainly to keep an open, extendible and customizable character of our event processing platform.

6 Implementation

6.1 Data and memory management

In order to implement event-based similarity searching in practice, it is not sufficient to only provide an efficient search algorithm. In addition, a well-designed infrastructure is required, being able to cope with potentially millions of events. Especially in case of events holding many event attributes, physical memory limits may soon be reached when trying to load all events and holding them as runtime objects in memory.

In order to overcome these difficulties, we propose two architecture models, the incremental load architecture and a batch loading architecture.

6.1.1 Incremental load architecture

The incremental load architecture is applicable independently from the size of the dataset. In this architecture, the data management module for the similarity search engine queries the database one-by-one for sequences of events. The engine then compares the sequence to the pattern, and memorizes only the sequence id assigned with the computed similarity score. Figure 73 illustrates the incremental load architecture.

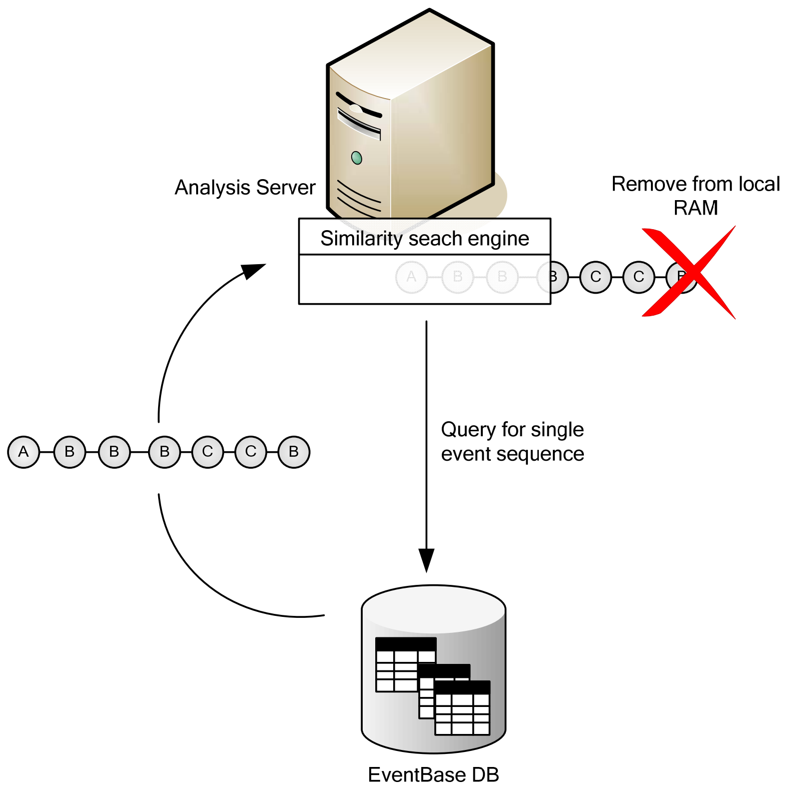


Figure 73: Incremental loading architecture

The downside of this architecture is that the database roundtrips cost performance in terms of execution speed, and also cause higher load on the database. This is critical in case of an operational system which is under constant load.

6.1.2 Bulk load architecture

In order to increase performance and reduce database roundtrips, the bulk load architecture may be applied. In this architecture model, a larger bulk of event sequences is retrieved from the database and kept in memory until all event sequences in the bulk have been searched. In case of smaller data sets up to several hundred thousand events, even the complete dataset can be loaded in bulk. Figure 74 shows the loading process in case of bulk loading.

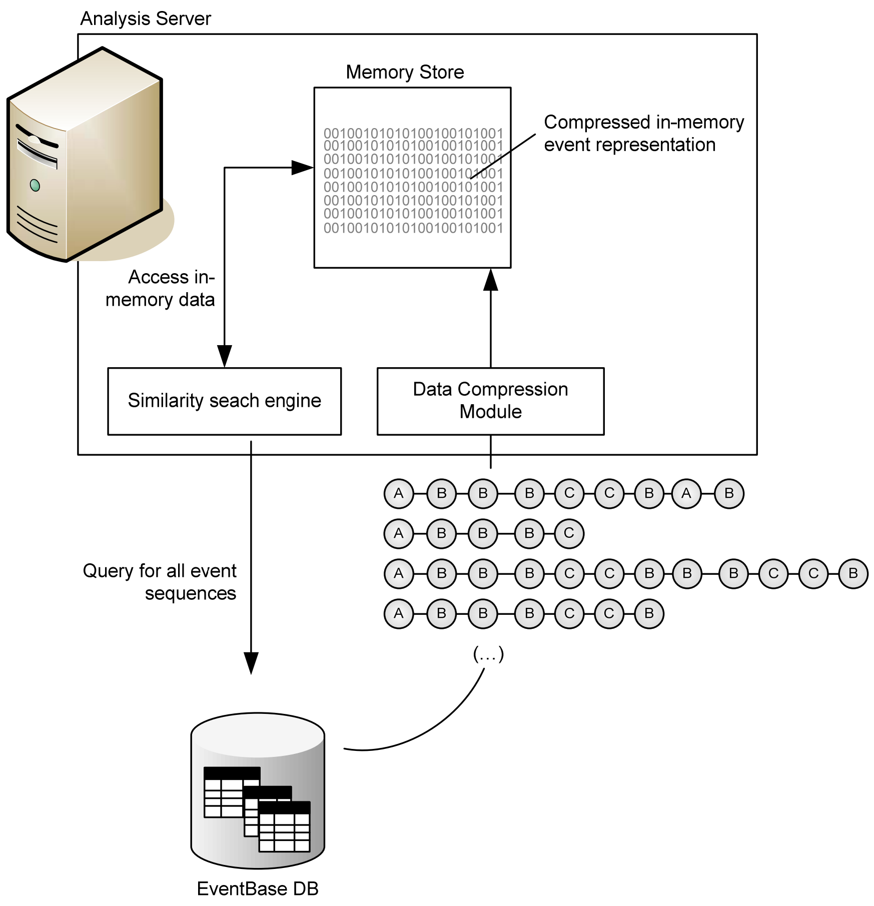


Figure 74: Bulk load architecture

In order to increase the number of events that can be hold in memory, we have implemented a specific in-memory event data store. It is capable of compressing the events by a factor of up to 1:10. This compression rate is achieved by representing the event data in efficient byte arrays. The in-memory data container also makes use of common attribute values. For instance, having “bet placed” events with a string attribute “bet type”, this attribute will always hold one value of a very narrow value set such as “goal bet”, “free throws” etc. In case of 2 different values, only 1 bit is required to store this information, compared to 8 bits per character for strings. In case of millions of events these savings are decisive.

The advantage of the approach is clearly that fewer database roundtrips are required. The time for compressing the events is negligible. The only downside is that still a lot of memory may be occupied, and one has to take care that in case of large bulk sizes and eventually multiple parallel similarity queries (requiring different data) the machine does not run out of memory during the matching process which by itself temporarily requires a higher amount of memory.

7 Providing similarity mining to the analyst

7.1 Overview

The value of a comprehensive model for event-based similarity search is only as high as the number of features that are made available, and more importantly, usable for operators of the software. Therefore it is a crucial part of this work to develop not only the algorithmic backend, but also a user interface which makes all above-presented searching opportunities and configuration options available to end-users.
In literature, discussions on user interfaces for similarity search are rare up to the date of writing this thesis. Related publications are discussed in section 2.5 on page 18. In the following sections, the key points of our user interface concepts will be presented. This includes the overall workflow, the graphical pattern editor and also the listing and presentation of results.

7.2 User workflow for similarity mining

In order to perform a similarity search, basically a reference pattern orand a data set are required. If no further information is available, a default similarity configuration model is applied. The default model weights all attributes equally (except of event object header attributes such as the unique event identifier). Thus, the user will get results without any initial configuration. Yet, the search model is rather sensitive to configuration parameters. Therefore, there are several opportunities to improve the results:

- the base similarity configuration can be refined,
- similarity priorities can be refined
- and the reference pattern can be refined.

All of these parameters are discussed throughout the subsequent sections. Figure 75 illustrates the major elements of our intended workflow model.

7.2.1 Setting the base similarity configuration and similarity priorities

The base similarity configuration defines the attribute similarity techniques and their default parameters used for different event attributes. In the user workflow, the base similarity configuration is strongly related to the event type definitions. Therefore, typically the application developer who designs the data repository, the event processing model and the event types as well as their attributes will provide meaningful default values for this configuration (Figure 75a). Based on the base similarity configuration, the business analyst which we consider as our actual end user of the similarity search may refine similarity priorities (weights of the individual features to be considered), and trigger a search.

In order to perform a similarity search, we propose two workflow models, which are discussed in the following.

7.2.2 Workflow model 1: Querying by example

Workflow model 1 assumes that the analyst first queries the data for known event sequences or discovers an event sequence in a search result. From the visualizations, it is possible to directly start the similarity search without any further refinements. In addition, the user may refine the search result in the search pattern editor. Figure 75b illustrates the workflow path.

7.2.3 Workflow model 2: Building a search pattern

Workflow model 2 illustrated in Figure 75c assumes that the user wants to trigger a search based on a new or existing reference pattern rather than querying by example. In the similarity search view of the SENACTIVE *EventAnalyzer*^{TM}, the pattern editor can be opened in order to create a blank search pattern or open an existing one. After modeling the desired event sequence and constraint blocks, the search is triggered.

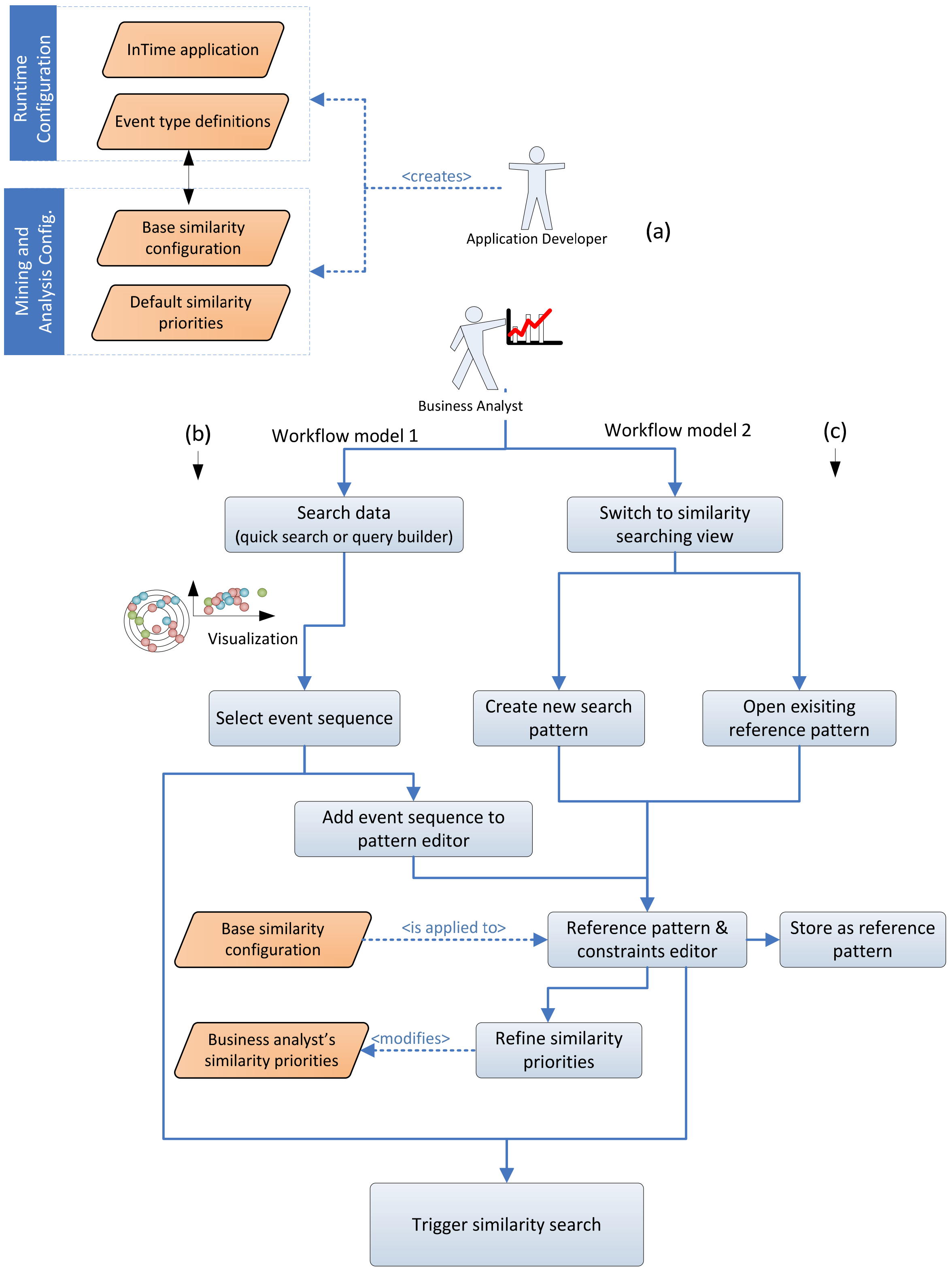

Runtime Configuration
InTime application
Event type definitions
Mining and Analysis Config.
Base similarity configuration
Default similarity priorities
<creates>
Application Developer
(a)
Business Analyst
(b)
Workflow model 1
Workflow model 2
(c)
Search data (quick search or query builder)
Switch to similarity searching view
Visualization
Select event sequence
Create new search pattern
Open exisiting reference pattern
Add event sequence to pattern editor
Base similarity configuration
<is applied to>
Reference pattern & constraints editor
Store as reference pattern
Business analyst's similarity priorities
<modifies>
Refine similarity priorities
Trigger similarity search

Figure 75: User workflow for event-based similarity mining

7.3 Similarity search pattern modeling

This section describes our search pattern editor. It enables users to refine a similarity query based on a discovered event sequence, or to build a new search pattern from scratch. The major requirements to be met by the editor are:

- Enable adding and removing events
- Allow editing attributes of events
- Allow modeling of constraint blocks
- Allow setting attribute and occurrence time constraints
- Enable excluding the events of a certain event type
- Store and load search patterns

In addition, the pattern editor must be integrated seamlessly into the SENACTIVE EventAnalyzer and should conform to the usability principles of the complete application. Therefore, we decided to build the editor upon an existing visualization module in the EventAnalyzer, the EventChart. It is a configurable 2D scatter chart for events, enabling to occupy the axis with different placement policies. These policies include time (see X-axis in Figure 76a and b), numeric event attributes (Y-axis in Figure 76b), sectors by numeric or literal attributes (X-axis in Figure 76c, and both axis in Figure 76d), ideal space filling (Y-axis in Figure 76c), and distinct axis positions for each event sequence (Y-axis in Figure 76a). As can be seen from the figures, all policies can be freely combined and configured. For further details on the placement policies the interested reader may refer to [49].

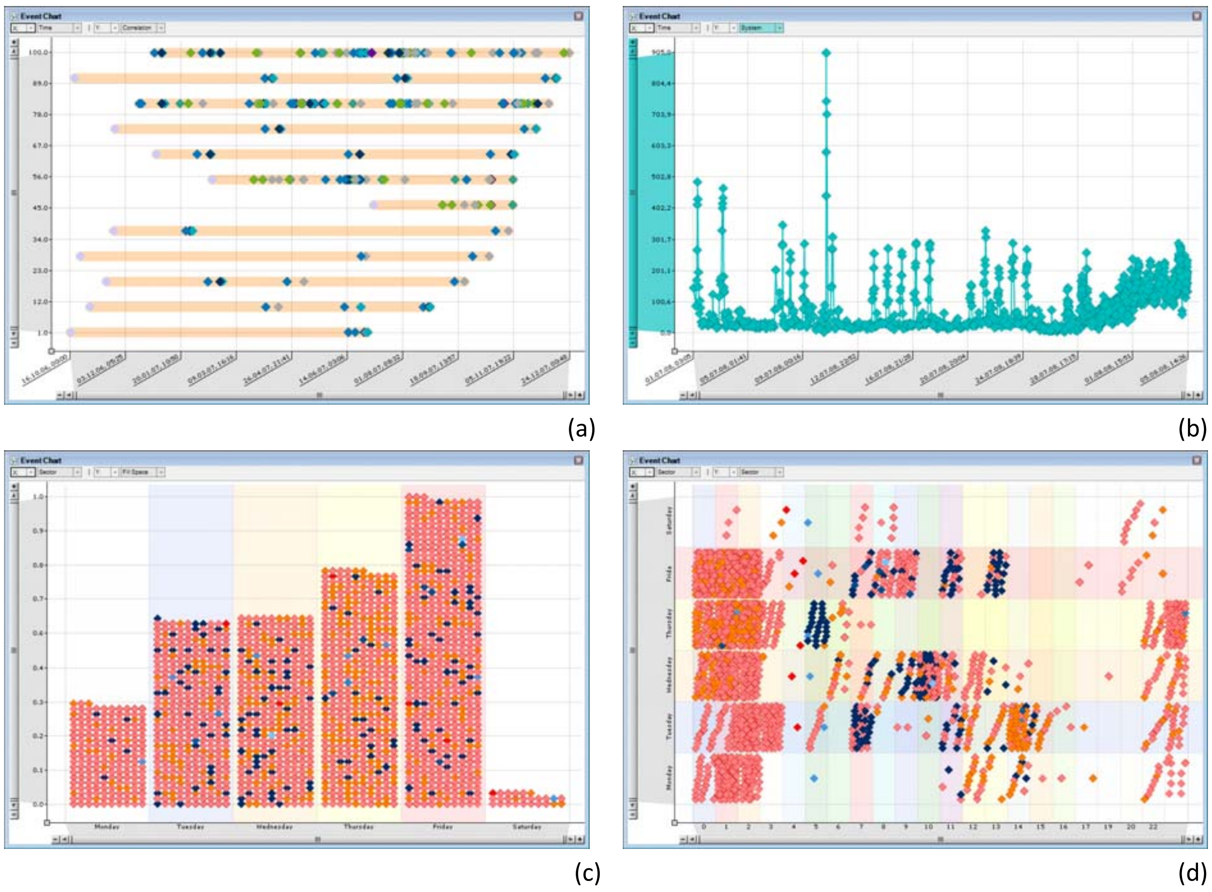


Figure 76: Event Chart with different placement policies

7.3.1 The similarity pattern editor

As discussed in the overview, the similarity pattern editor should be integrated directly into the SENACTIVE EventAnalyzer. We therefore decided to make it available in each of the existing visualizations via context menus. Having discovered an interesting event sequence, for instance in the event tunnel view, via the context menu it is possible to directly search for similar event sequences (see Figure 77). The search scope may either be the current result set (a queried and filtered data set) or the complete event repository. In addition, it is possible to open the search pattern editor. The selected event sequence is used as an input for the editor and can then be refined.

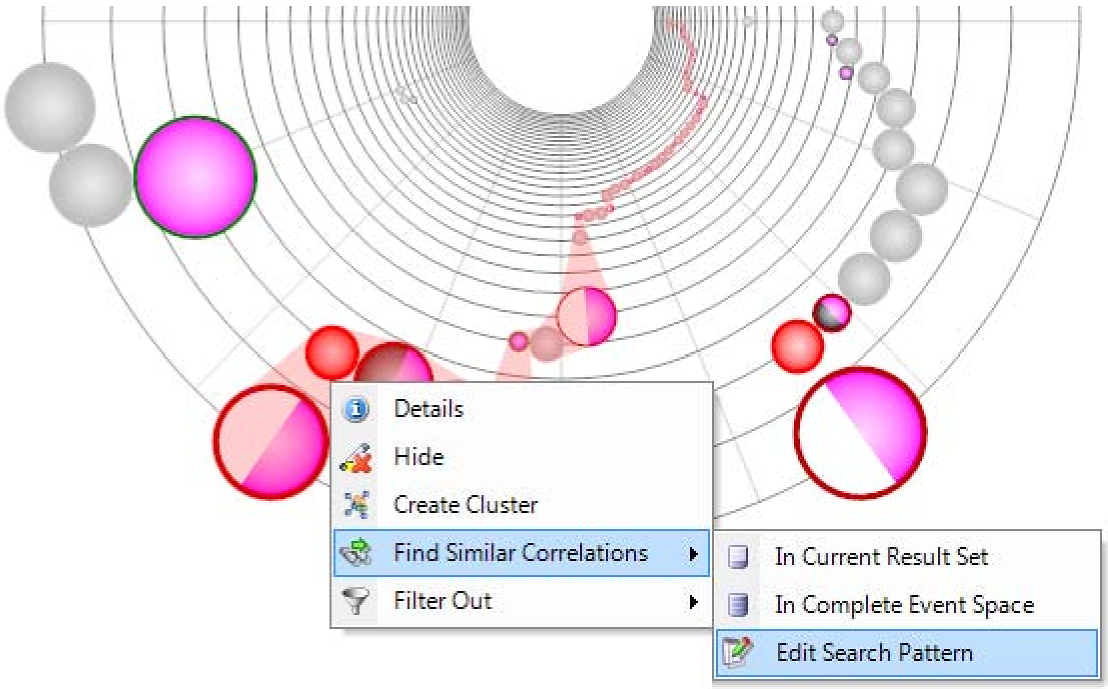
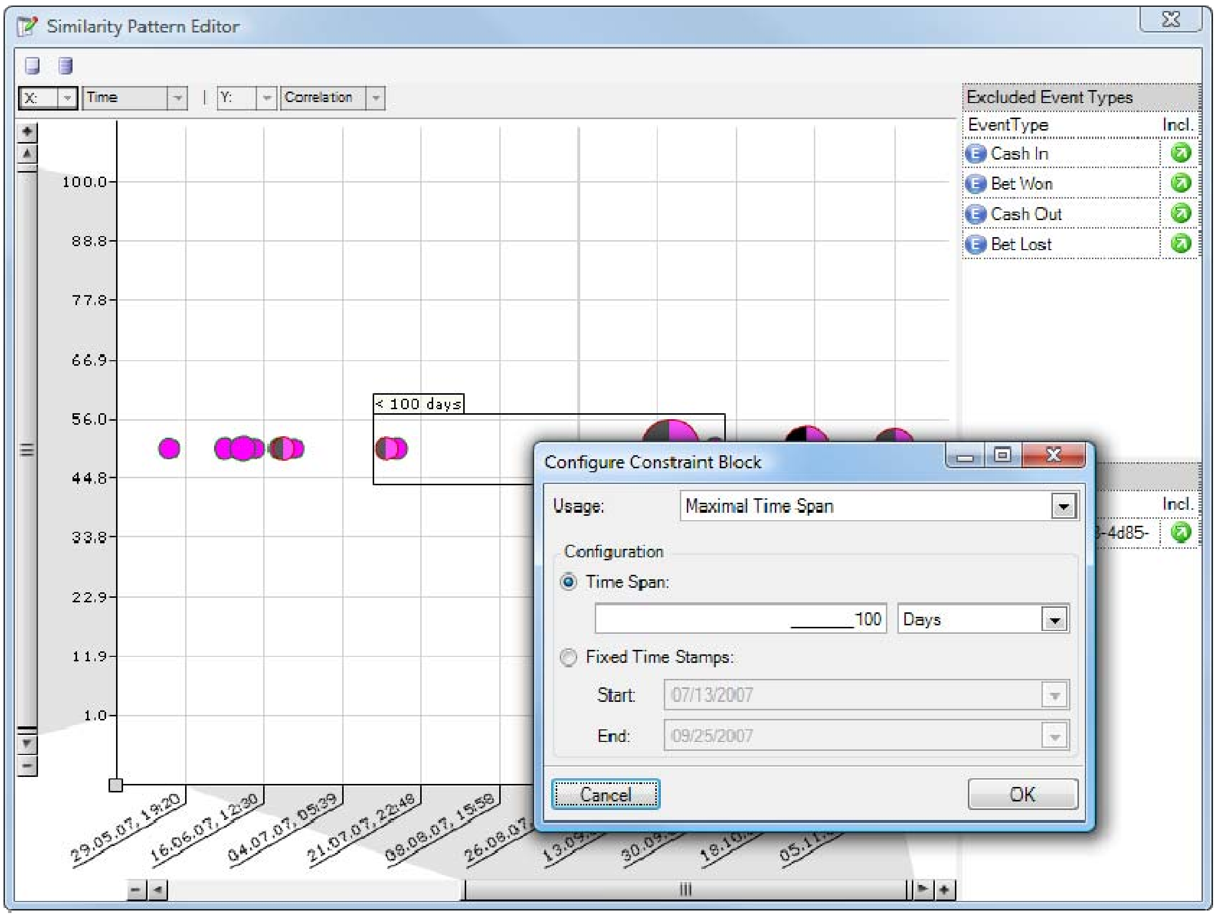


Figure 77: Integration of search pattern editor in visualizations

After invoking the edit operation, the pattern editor opens as a separate dialog window. In provides operations for adding, editing and removing constraint blocks. Figure 78 shows the pattern editor. In its default configuration, on the x-axis the user sees the time. On the y-axis all events of the sequence are plotted in center-axis position.



Configure Constraint Block

Usage: Maximal Time Span

Configuration

[✓ Time Span:] 100 Days

Fixed Time Stamps:

Start: 07/13/2007

End: 09/25/2007

Cancel OK

Figure 78: Example for constraint block configuration – time constraint blocks

In order to emphasize certain characteristics of the pattern, such as a numeric event attribute, the user could change the configuration and use other placement policies on the y-axis. The x-axis is always occupied by the event's occurrence time. This limitation results from the fact that all constraint blocks are strongly time-related. Editing them and displaying them is virtually impossible with other attributes on the axis.

Figure 79 shows the context menu for excluding single events or all events of a given event type. The figure shows events plotted with different colors and a certain shape. The reason is that the pattern editor keeps the user's color, size and shape mappings set for the other views, in order to recognize the events also in the pattern editor.

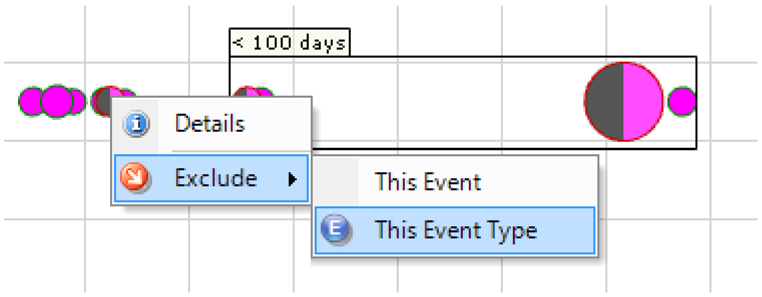
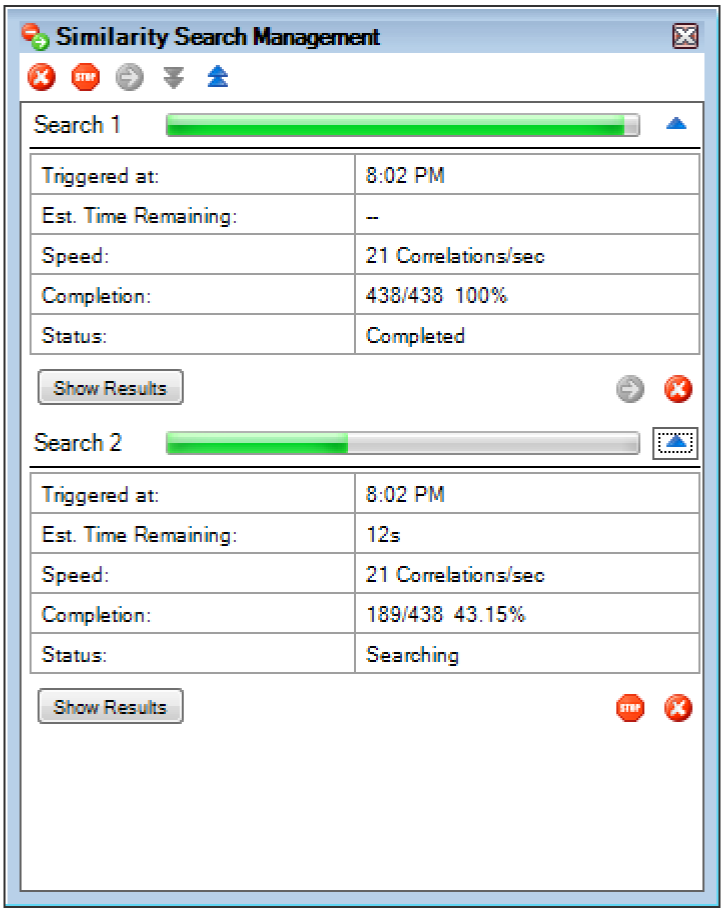


Figure 79: Excluding events and event types

7.4 Similarity search management

A similarity search might be a long-running process in case of a large data repository to be analyzed, comparable to typical data mining processes. Hence, it is intuitive to understand the search as kind of a background activity, which, triggered once executed in the background while the user is able to perform other analysis tasks. It may even be desired to trigger several searches in parallel, for instance over night, in order to view the results on the next day. Out of these considerations, we propose a similarity search management module. This module is responsible for starting searches on background threads, and it displays the progress of each search. Figure 80 shows the panel in the EventAnalyzer.


Similarity Search Management

Search 1

Triggered at:	8:02 PM
Est. Time Remaining:	--
Speed:	21 Correlations/sec
Completion:	438/438 100%
Status:	Completed

Show Results

Search 2

Triggered at:	8:02 PM
Est. Time Remaining:	12s
Speed:	21 Correlations/sec
Completion:	189/438 43.15%
Status:	Searching

Show Results

Figure 80: Similarity search management panel

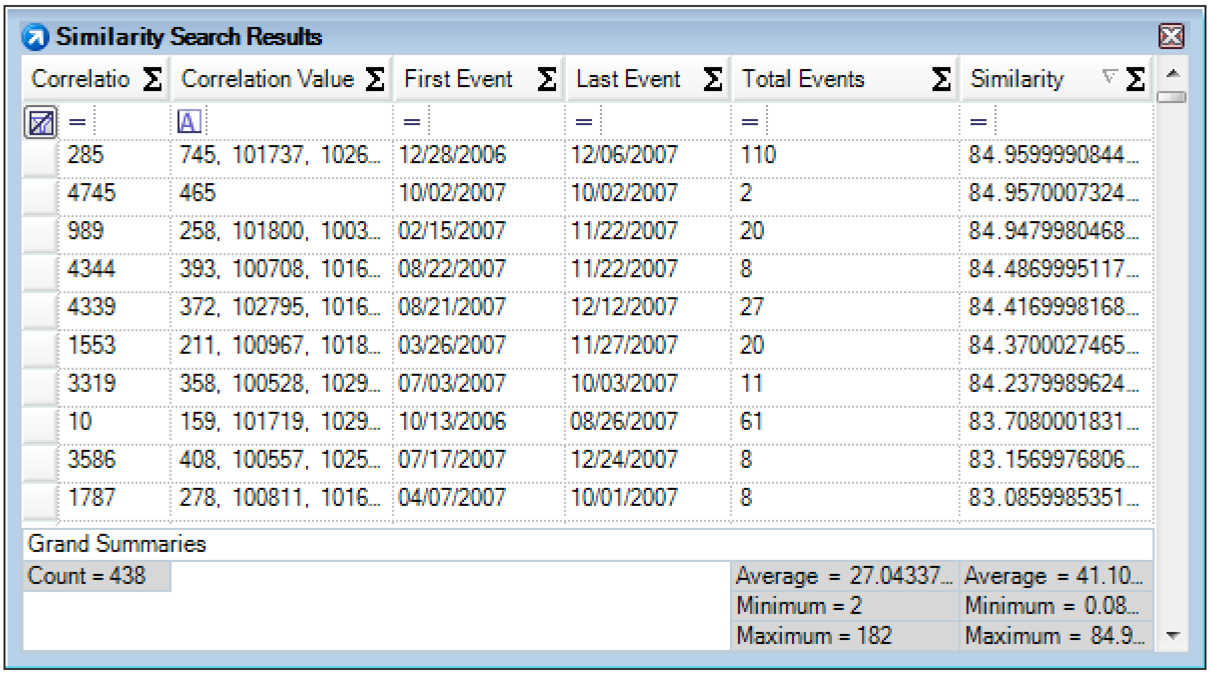
As can be seen from the figure, each search can be paused or cancelled directly from the view. From the already searched event sequences a forecasting is done on the total search time.

7.5 Visualizing similarity search results

In the EventAnalyzer, we offer two ways for displaying search results, namely a similarity ranking view and the graphical visualization of the discovered event sequences.

7.5.1 Similarity ranking view

The similarity ranking view lists the event sequences by similarity ranking in a table. It is possible to click the entries in order to highlight the event sequence in the visualization. The table also offers useful summaries such as average similarity, average number of events in the searched sequences and the like. Figure 81 shows the similarity ranking view.



Similarity Search Results

Correlatio	Correlation Value	First Event	Last Event	Total Events	Similarity
285	745, 101737, 1026...	12/28/2006	12/06/2007	110	84.9599990844...
4745	465	10/02/2007	10/02/2007	2	84.9570007324...
989	258, 101800, 1003...	02/15/2007	11/22/2007	20	84.9479980468...
4344	393, 100708, 1016...	08/22/2007	11/22/2007	8	84.4869995117...
4339	372, 102795, 1016...	08/21/2007	12/12/2007	27	84.4169998168...
1553	211, 100967, 1018...	03/26/2007	11/27/2007	20	84.3700027465...
3319	358, 100528, 1029...	07/03/2007	10/03/2007	11	84.2379989624...
10	159, 101719, 1029...	10/13/2006	08/26/2007	61	83.7080001831...
3586	408, 100557, 1025...	07/17/2007	12/24/2007	8	83.1569976806...
1787	278, 100811, 1016...	04/07/2007	10/01/2007	8	83.0859985351...

Grand Summaries

Count = 438				Average = 27.04337...	Average = 41.10...
				Minimum = 2	Minimum = 0.08...
				Maximum = 182	Maximum = 84.9...

Figure 81: Similarity ranking view

7.5.2 Graphical view

The event sequences returned from a similarity search are plotted in all EventAnalyzer views like a normal search result. In addition, similarity highlighting enables to visually distinguish most similar hits from least similar ones. This highlighting configuration is provided to the user via a set of sliders (see Figure 82). Dragging these sliders directly updates all visualizations.

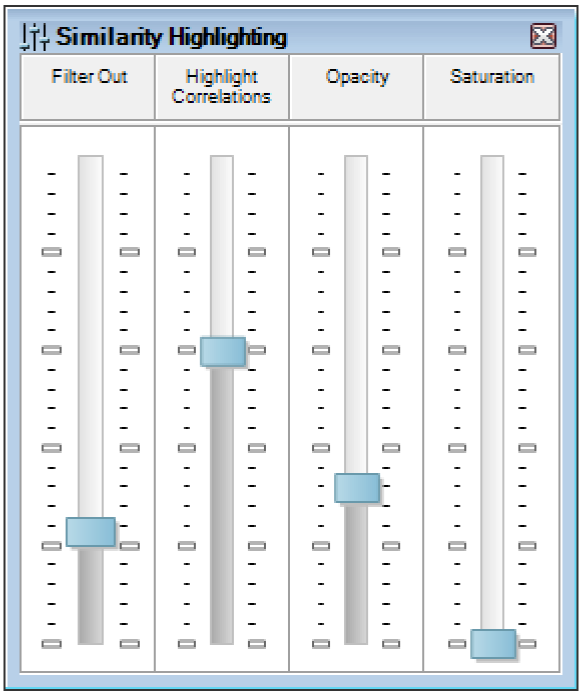


Figure 82: Similarity highlighting control panel

The following highlighting techniques are supported currently:

- **Filtering by threshold** – All sequences with a similarity less than a user selected threshold are filtered out
- **Event sequence highlighting** – Sequences which are most similar are painted as connected sequences
- **Opacity** – The less the similarity the lower the opacity of the plotted events
- **Saturation** – The less the similarity, the lower the saturation

8 Results and evaluation

8.1 Overview

In the course of this work, a comprehensive evaluation has been carried out in order to judge both algorithmic performance and accuracy of search results. We claim to provide a generic model for event sequence similarity. Hence, in order to prove the generic character of our approach, we decided to evaluate results based on strongly varying input data from different application domains. In addition for each evaluation scenario we defined different objectives, which are reasoned by the idea to cover different interests of our software's end users. For each scenario, the evaluation is spilt up into two parts, the results of performance measures and the judgment of search results including a discussion on the degree to which we see initial aims being fulfilled by the gathered results. Especially the second part is done in awareness of the fact that full objectiveness is virtually impossible when it comes to assessment of similarity search results. We therefore focus on our concrete, application specific objectives for judging the value of the results.

8.2 Case studies

8.2.1 C1 Online gambling –user activity histories

8.2.1.1 Scenario and data structure

The first evaluation scenario aims at investigating on the algorithmic performance and correctness of search results in a controlled and exactly defined environment. We achieve this environment by utilizing simulated data with controlled variations in the generated event sequences. The simulation model generates events representing the activity log of single customers of an online betting platform. Such sequences include the following activities: opening the account (i.e., registering at the platform), cashing-in and cashing-out money, placing bets, winning and losing bets and notifications on failed bet placements. The occurring event types and their attributes are depicted in Figure 83.

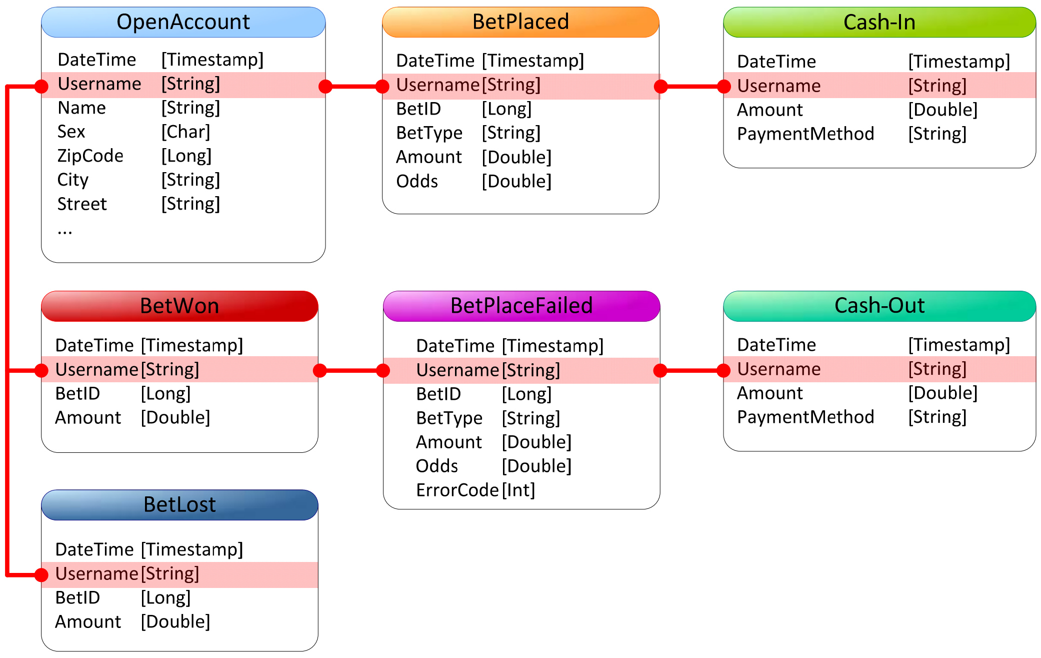


Figure 83: Event types and correlations in evaluation scenario C1 – Online gambling

The simulation model generates several arbitrary sequences of events, whereby the simulation engine takes care of correctness and validity of the sequence. For instance, the simulation keeps track on the virtual cash balance of a customer during the simulation, so that only bet placements are simulated, if money is available. In addition to the arbitrary sequences, several account histories are generated, which follow a defined template structure. These template structures have been defined based on a requirements study carried out at a large European online betting and gambling provider. In the course of this study, known, suspicious behavior pattern have been identified and described. Yet, the descriptions are fuzzy, and the concrete sequences simulated vary in the number of events occurring as well as certain event attribute's values.
For instance, one of these patterns is the sleeper pattern. Sleepers are users which, after registration and maybe a few initial bets do not bet for a long period of time. It is then remarkable, if such sleepers suddenly cash-in a large amount of money, place a very high bet, and cash-out again immediately. This is often an indication that the user had insider information on a bet or places the bet for a user who is not allowed to place it, for instance game officials such as referees or players and other participants.

8.2.1.2 Objectives and evaluation focus

For the evaluation of our similarity search algorithm in the given context, we define the following objectives:

- Among the simulated account histories, 10 are simulated based on a selected template. Using one of these 10 sequences, the other 9 sequences must be discovered with the similarity search.
- None of the other account history should be retrieved, except in case the arbitrary simulation generates a pattern similar to our template.

In addition to these measureable objectives, the focus of this evaluation case is on:

- Determining the sensitivity of the model towards the similarity configuration.
- Measuring the performance with different configuration parameters.

In the following, different combinations of search patterns and similarity configuration options are defined which have been executed for the case study.

8.2.1.3 C1.a – Type matching with subsequence searching

8.2.1.3.1 Search pattern and configuration

In this scenario, no attribute similarities are considered. Weighting of all possible type deviations and missing events are neutral and equal; matches do neither have to start with the first event nor end with the last event.

The following reference sequence is used as the search pattern, whereby the table lists the event type colors. Thus, the short pattern sequence starts with an "open account" event, followed by a placed bet and a notification that the bet was lost. At the end of the sequence, this user won a bet and cashed out directly after.

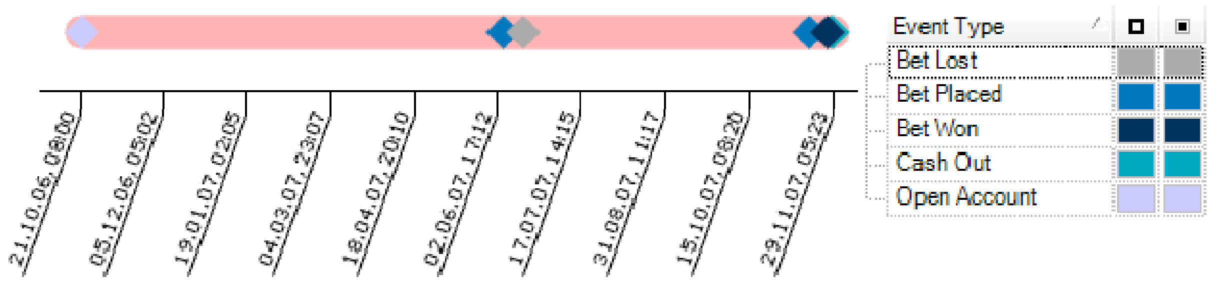


Figure 84: Search pattern for evaluation case C1.a

8.2.1.3.2 Search results and discussion

Plainly spoken scenario C1.a tries to find occurrences of the same order of event types. The time spans between the events are not considered.

The results are accordingly. Figure 85 shows the best matches in the given scenario among the searched 438 event sequences. According to the plot, these results intuitively appear inappropriate: Most matches are longer than the pattern sequence and show a completely distinct shape compared to it. Yet, this results simply from the fact that we configured subsequence searching. Thus, for most of these discovered event sequences only the first few events match, and the rest is ignored.

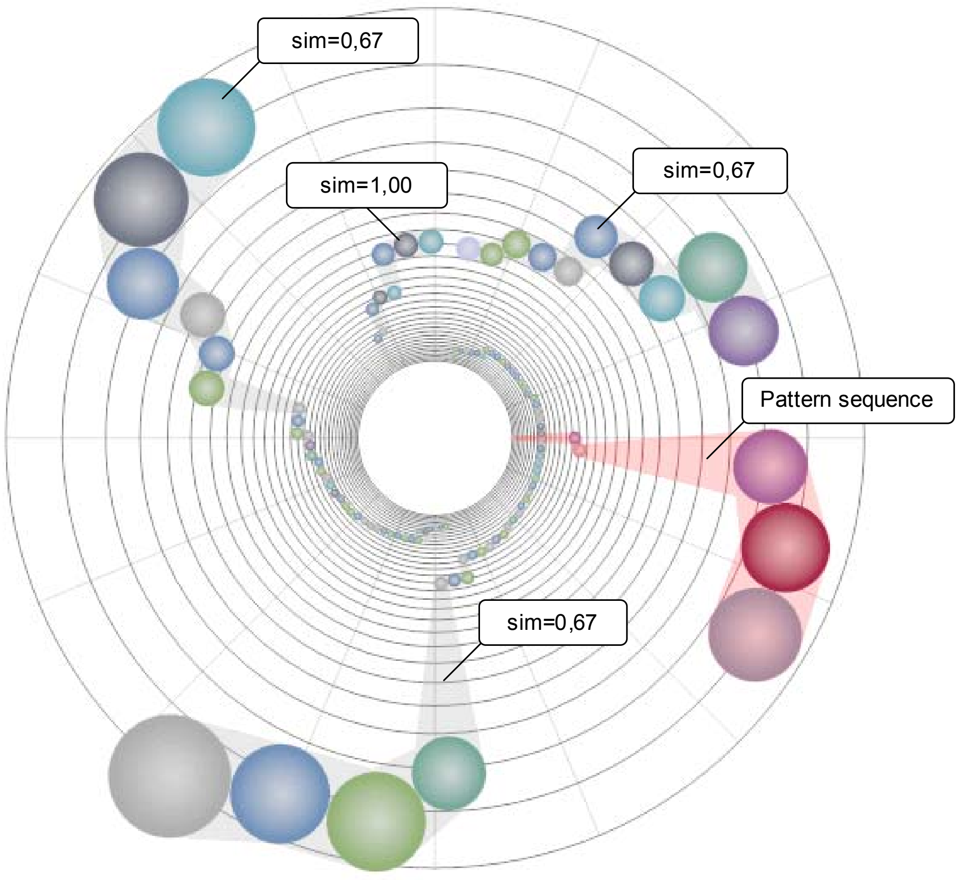


Figure 85: Best search results for scenario C1.a visualized in the Event Tunnel

Remarkable is the high ratio of “overhead” time, i.e. the time for data loading and preparation in relation to the pure algorithm time (see performance summary below). Caused by the fact that the matching is very fast in case of the short pattern event sequence, data loading and preparation make up more than 75% of the total search time in this scenario[8].

8.2.1.4 C1.b – Type matching without subsequence searching

8.2.1.4.1 Search pattern and configuration

This scenario is defined equally to scenario C1.a, but matches are anchored at the start and the end of the searched sequence.

8.2.1.4.2 Search results and discussion

Scenario C1.a showed that subsequence searching may lead to intuitively incorrect results for the given dataset. Requiring a match to start with the first event and end with the last event (everything else decreases the similarity) retrieves sequences which intuitively appear by far more similar. The best matches are depicted again in Figure 86.

This scenario fulfils already our initial requirement to retrieve a set of simulated event sequences, which all have a very similar structure concerning the occurrence of different event types.

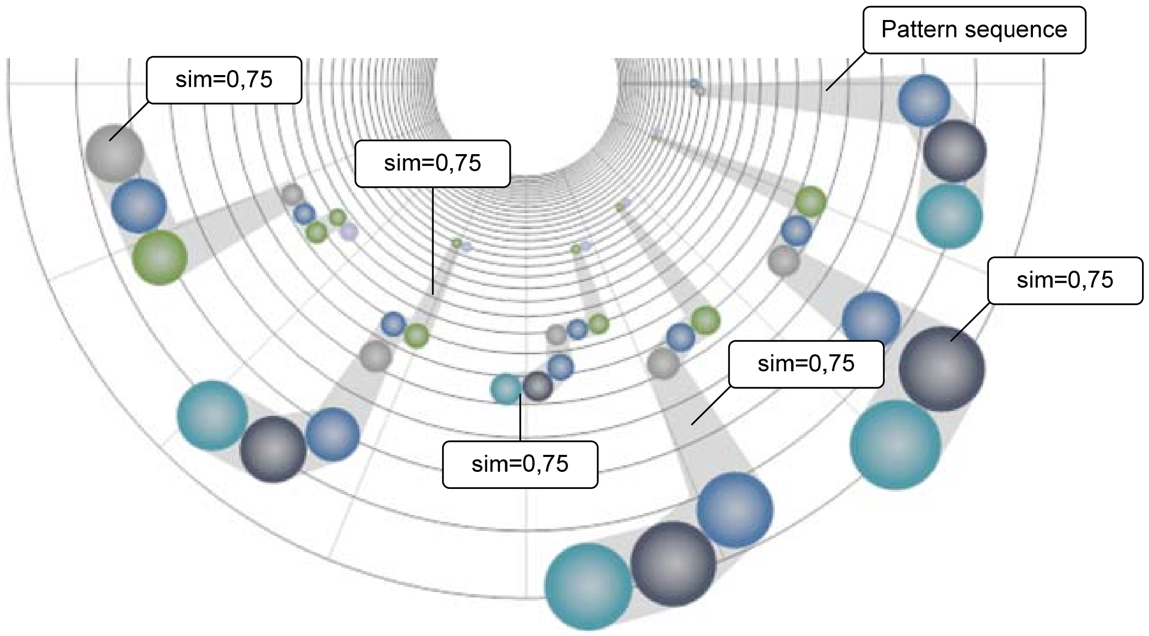


Figure 86: Best search results for scenario C1.b visualized in the Event Tunnel

[8] It is important to mention at this stage that for the evaluation the proposed reference architecture loading event sequence-by-event sequence from the database is used.

8.2.1.5 C1.c – Type matching with time deviations (full-sequence matching)

8.2.1.5.1 Search pattern and configuration

For scenario C1.c we use the same search pattern as before, but occurrence time deviations are considered in addition to the order of the events.

8.2.1.5.2 Search results and discussion

In section 4.3.3 we presented two different modes for handling time deviations, namely the absolute time difference mode and the relative time difference mode.

The given evaluation scenario showed that the absolute time difference mode is virtually inapplicable in this context. The time spans between the events in the scenario are relatively large (e.g. several hours to a couple of months). Thus, some of these deviations have huge absolute values and require a very small scaling factor in order to scale them to a range comparable to other aspects such as type deviations. In return, this scaling factor causes “minor” deviations to be almost ignored. Yet, these “minor” deviations might also be a couple of days and decisive for the search semantic.

The relative time mode works out better for the scenario. Still, the best matches in the previous scenario already had a very similar temporal structure (see Figure 86) so that again these sequences have been discovered as the best matches.

8.2.1.6 C1.d – Type and attribute matching (Numeric attributes)

8.2.1.6.1 Search pattern and configuration

In this scenario, the following event attributes are considered in addition, using the normalized absolute difference similarity technique (see section 4.2.1):

- BetPlaced.Amount
- BetPlaceFailed.Amount
- Cash-In.Amount
- Cash-Out.Amount
- BetPlaced.Odds
- BetPlaceFailed.Odds

8.2.1.6.2 Search results and discussion

The discovered sequences for this evaluation case again only slightly differ from the retrieval results in scenario C1.b. In the simulated data set, variations in terms of the selected event attributes are not significant, and thus considering these attributes in addition only has slight influence on the overall similarity score. Obviously, considering the event attributes costs some performance.

As a variation from the originally defined scenario C1.d we also tried to maximize the weight of only the selected event attributes. Using this configuration, some other event sequences consorted with the prior discovered sequences, but all in all, we found that it is hard to adjust the weights so that absolute difference similarity deviations in combination with type deviations allowing null-mappings return reasonable result. The problem is similar as with time deviations: In order for such a combination to return meaningful results, the costs of the absolute difference deviations must be well-adjusted with other similarity costs. In other words, if the absolute value differences (which the user will not know up-front) are very small, deviations will show almost no effects in combination with costs for other mappings such as null-mappings.

8.2.1.7 C1.e – Using strict order constraint blocks

8.2.1.7.1 Search pattern and configuration

This scenario is defined equally to scenario C1.c, but in addition 2 strict order blocks guarantee the sequence of bet placements and immediate cash-outs.

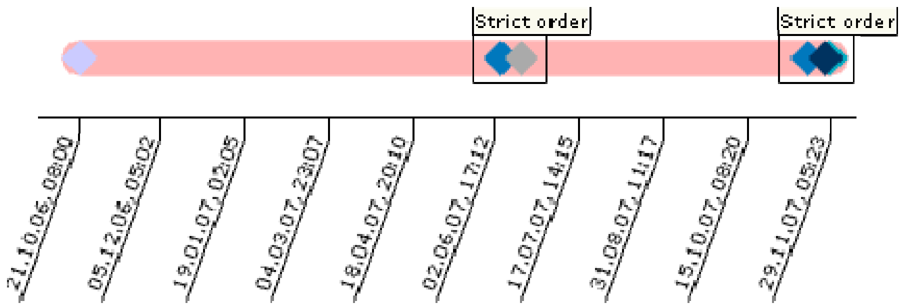


Figure 87: Search pattern for evaluation case C1.e

8.2.1.7.2 Search results and discussion

For the given data set, search results did not differ from the search results in scenario C1.c. This is caused by the fact that the simulation model used to generate these sequences follows a template where these events are always simulated in order. Only the execution speed is slightly better, as a few solutions can be omitted earlier on the way to discovering the best solution.

8.2.1.8 C1.f – Using a minimum timespan constraint block

8.2.1.8.1 Search pattern and configuration

In the evaluation scenario C1.f, a minimum timespan block is utilized to guarantee an idle period in the user activities for about 3 months before placing a new bet, winning and cashing out immediately. The search pattern is depicted in Figure 88.

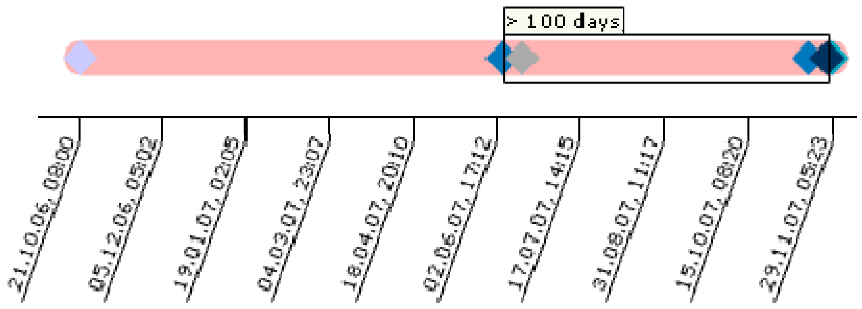


Figure 88: Search pattern for evaluation case C1.f

8.2.1.8.2 Search results and discussion

Looking at the search results for this scenario shows that the best matches in the previous scenario did not pass the constraint block. Thus, all in all the best matches are much less similar. It is well visible that all of these matches have a very long idle phase where no events occurred (see Figure 89).

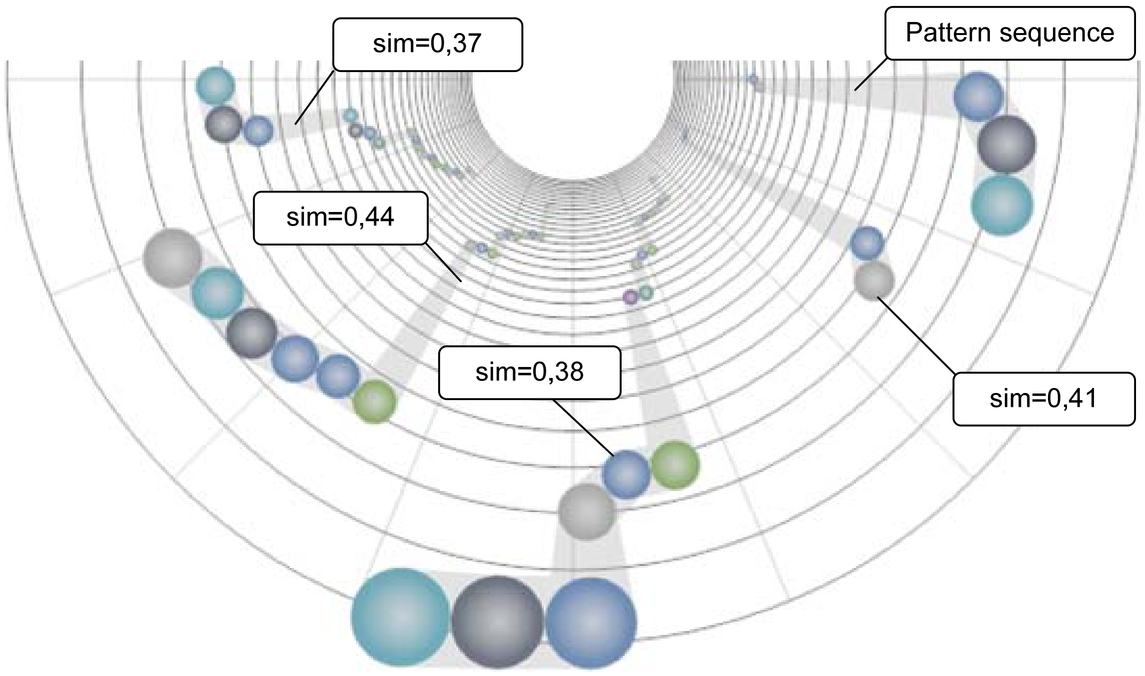


Figure 89: Best search results for scenario C1.f visualized in the Event Tunnel

8.2.1.9 Performance summary

All of the scenarios have been executed with the following data set:

Total number of events:	12455
Total number of event sequences:	438
Average number of events per event sequence:	27,043

First, the scenarios have been executed without an initial threshold. Thus, the threshold value of costs is dynamically updated with every possible solution, but initially a set of potentially bad solutions have also been build up completely, until the dynamic threshold bit by bit decreases and more and more solutions can be omitted early.

Scenario	Events in pattern	Total time	Algorithm time[9]	Events/sec total	Sequences /sec total	Events/sec algorithm	Seq./sec algorithm
C1.a	6	00:00:14.25	00:00:03.32	889,64	31,08	3663,24	128,53
C1.b	6	00:00:19.58	00:00:08.28	638,71	22,46	1500,60	52,65
C1.c	6	00:00:15.03	00:00:03.05	830,33	29,13	4151,23	146,66
C1.d	6	00:00:25.58	00:00:11.68	488,43	17,13	1073,71	37,35
C1.e	6	00:00:23.21	00:00:09.77	536,63	18,87	1274,82	44,83
C1.f	6	00:00:24.12	00:00:10.02	516,37	18,16	1243,01	43,71

Table 3: Performance results for evaluation scenario C1 without initial threshold

[9] Measure the pure algorithm execution time, i.e. the total time minus the overhead for data retrieval from database and conversion of the raw data into the processable events.

In addition, we executed the scenarios with an initial threshold, for a target similarity of 0,5 with the objective to speed up the searching process. The best matches shown above have still been discovered. Performance results are listed below.

Scenario	Events in pattern	Total time	Algorithm time	Events/sec total	Sequences /sec total	Events/sec algorithm	Seq./sec algorithm
C1.a	6	00:00:14.99	00:00:00.97	830,88	29,21	12580,81	442,42
C1.b	6	00:00:15.91	00:00:01.01	803,84	27,52	12331,68	433,66
C1.c	6	00:00:16.55	00:00:01.17	752,56	26,46	10645,30	347,36
C1.d	6	00:00:17.96	00:00:02.05	693,48	24,38	6075,61	213,56
C1.e	6	00:00:17.52	00:00:01.49	710,90	25,00	8359,06	293,95
C1.f	6	00:00:14.27	00:00:01.23	827,81	30,69	10126,02	356,06

Table 4: Performance results for evaluation scenario C1 with initial threshold

8.2.2 C2 Trouble tickets – change history sequences

8.2.2.1 Scenario and data structure

Trouble tickets in general can be understood as issues and problems reported either by customers or company-internal. Typically, a trouble ticket holds a problem or task description. It may be assigned to a certain person or support group and have a defined priority. Common trouble-ticket systems keep track of each ticket's status, whereby typical states are *open*, *assigned*, *resolved*, *incomplete* etc.

The second evaluation scenario aims at analyzing sequences of trouble-ticket traces. In contrast to the first evaluation scenario, for this case study real data from a trouble ticket system have been used instead of simulated data. The data have been provided by an international company offering, among others, IT services such as maintaining and monitoring other companies' servers and IT landscapes. With thousands of customers who all might submit issues to the trouble ticketing system, the analysis thereof becomes a demanding task.

Figure 90 depicts the relevant event types for this application example. In the concrete case, server alerts are captured. In addition, changes on trouble-tickets are traced and reflect as "ticket created", "ticket resolved", "ticket changed" and "ticket reopened" events. The figure also shows how these events are correlated to change history sequences: All ticket events correlate via the unique ticket ID; server alerts are unique via their event handle, server handle and date fields. In the dataset, tickets might be opened due to a server alert, but not necessarily. Many ticket histories also contain solely the various ticket events, in case they have been created manually without a prior alert. In addition, many alerts exist without any ticket events.

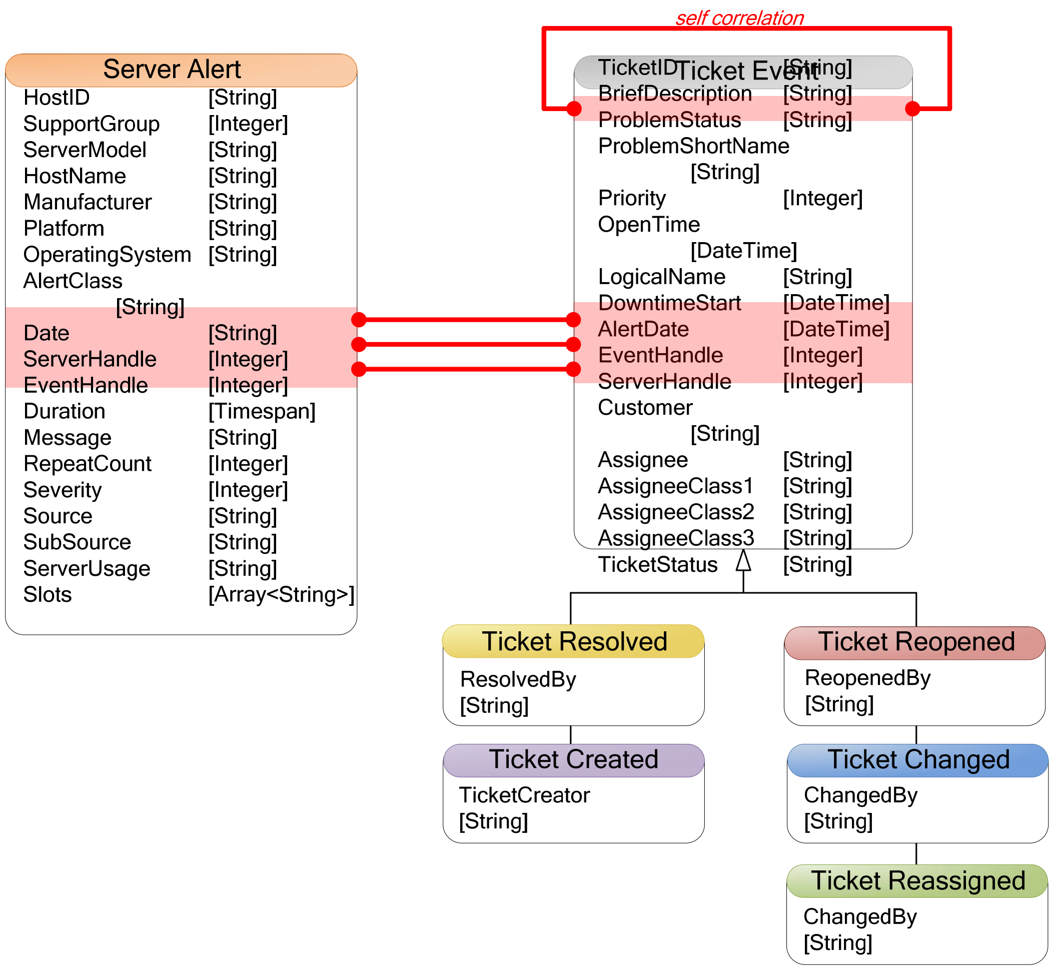


Figure 90: Event types and correlations in evaluation scenario C2 – Trouble tickets

In the given case, the following questions have been of particular interest:

- Which tickets have suspicious or extraordinary histories?
- Which tickets have not been resolved within the foreseen time period? Is it a repeating pattern that always certain ticket classes are not resolved in time?
- With tickets have many reassignments, and is there a general pattern such as *"All tickets of type A are first assigned to support group X which then assigns them to support group Y before they are then again forwarded to support group Z who's members finally resolves these issues"*?

Obviously, for certain queries such as retrieving tickets with many reassignments no similarity search is required. Yet, in order to assess if a certain type of assignment sequence seems to be a general pattern, we propose to apply the presented event sequence similarity model. In many cases people also have a certain suspicion and want to prove if this suspicion holds true based on the historic data.

8.2.2.2 Objectives and evaluation focus

For the evaluation of our similarity search algorithm in the given context, we define the following objectives:

- For a given, interesting sequence of ticket assignments, the similarity search is able to discover further ticket histories, if available, having a similar assignment history.

- It must be possible to assess whether the given assignment sequence can be understood as a general pattern reoccurring several times, or whether it is not reoccurring.

In addition to these measureable objectives, the focus of this evaluation case is on:

- Measuring the performance in case of sequences with strongly varying lengths
- Measuring the performance in case of a large amount of event attributes
- Proving the applicability of the model in a real-world use case

8.2.2.3 C2.a – Searching the complete data set for a known event sequence

8.2.2.3.1 Search pattern and configuration

For the first evaluation scenario, we utilized a known ticket history as the search pattern. This ticket was identified more or less by chance by one of the operators in the incident management department. The plot in Figure 91 shows that this ticket has a significantly long history with several ticket changes and also reassignments (blue, and green events).

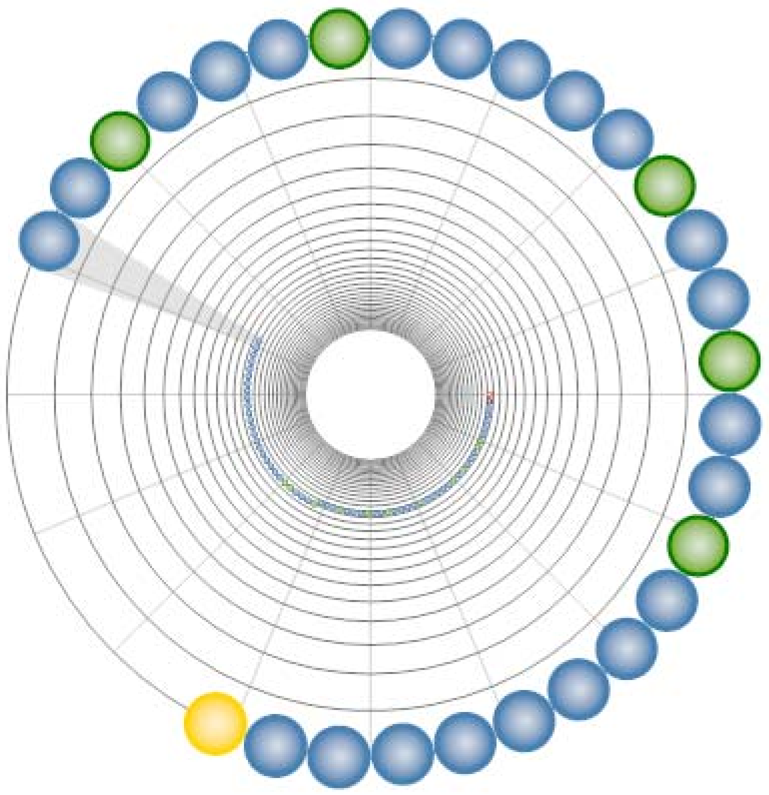


Figure 91: Activity history for a known incident ticket plotted in the event tunnel

For the given use case, the sequence of reassignment is of particular interest. Figure 92 visualizes how the ticket was assigned between different support groups, whereby each sector on the Y-axis represents one support group. Time is on the X-axis.

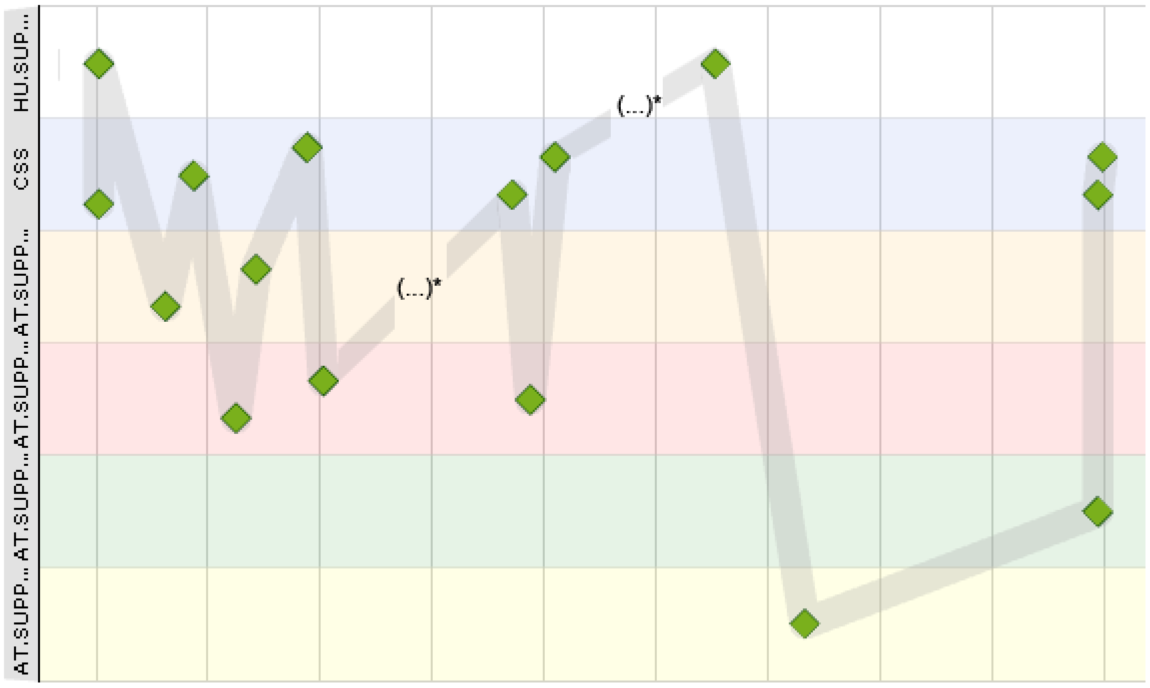


Figure 92: Sequence of ticket reassignment events over time (x-axis) by assigned support groups (y-axis)[10]

For the first scenario, we searched for this complete event sequence in the reference data set of about 165,000 events with the objective to discover similar occurrences.

Search Parameters:

▪ Match must start with first event:	False
▪ Match must end with last event:	False
▪ Time matching mode:	Relative
▪ Attribute similarities:	Ticket Reassigned.Assignee – Levenstein string similarity

8.2.2.3.2 Search results and discussion

The search for the known, very long ticket event sequence returned no matches. This means, with the given configuration for no event sequence a solution could be found with sufficiently low costs to be at least 50% similar.
Our investigations on this result showed that the reason is simply that no other event sequences of such an extreme character, i.e. that many reassignments and ticket changes is contained in the data set. This results in a need of multiple null-mappings in each solution, which drastically decreases the similarity score.

Yet, our objective to figure out whether such a behavior is a reoccurring pattern is fulfilled as we figured out that at least in our reference data set, no significantly similar event sequence is contained.

8.2.2.3.3 Performance summary

Total number of events:	165841
Total number of event sequences:	87241

[10] In the chart, two areas are marked with an asterisk. At these points in time, the data set showed a longer time period between the events which has been cut out in order to fit the figure to the page size.

Average number of events per event sequence:	1,9
Average number of events per event sequence with at least 1 ticket event:	8,47
Initial threshold set for a target similarity of:	0,5

Scenario	Events in pattern	Total time	Algorithm time	Events/sec total	Sequences /sec total	Events/sec algorithm	Seq./sec algorithm
C2.a	91	00:18:43.10	00:08:11.34	147,67	77,68	337,67	177,68

Table 5: Performance results for evaluation scenario C2.a

8.2.2.4 C2.b – Finding reassignment scenarios

8.2.2.4.1 Search pattern and configuration

Scenario C2.a showed that a too specific or extreme pattern is hard to discover in the data. One of the reasons why the previous scenario did not return any results was also that the search was not particularly focused on what we have actually been interested in most: the reassignments. The pattern sequence contained a whole range of ticket changed events, which of course have also been considered during the search and influence the matching process significantly.

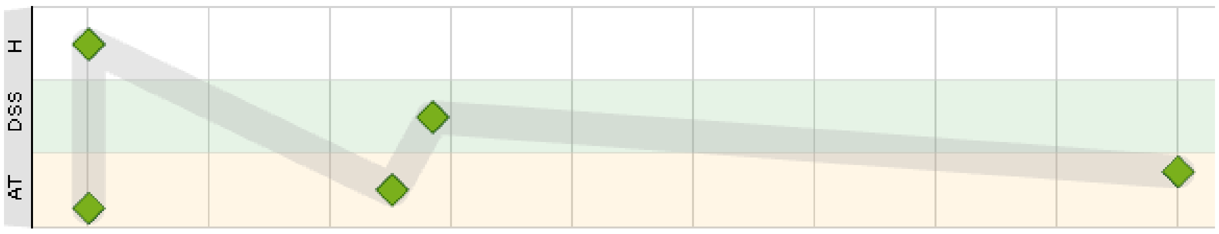
Scenario C2.b attempts to concentrate the similarity search to the reassignment. Therefore, we excluded the ticket changed events totally from the search pattern. In addition, the order remains unconsidered and also the temporal structure is omitted with the objective to simply discover if several support groups always assign the tickets to each other, no matter in which order.

For the scenario, we furthermore chose a shorter reassignment sequence, with reassignments among 3 support departments “AT”, “DSS” and “H”. Within each of these departments support groups exist, such as “AT.SUPPORT.SAP”. We tried to figure out, if there are regular reassignments among these departments, which normally should not occur as each has separate concerns.

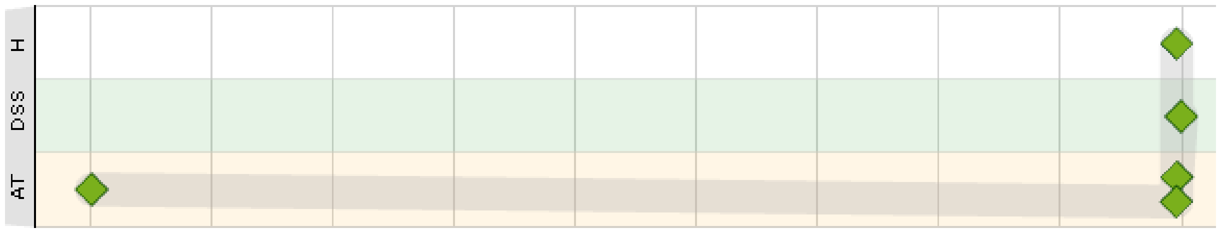
In scenario C2.a we searched the complete data set. In fact, this is not required: event sequences containing only an alert event but no ticket events because for this alert no ticket had been opened, as well as sequences with less than 2 reassignment events can be pre-filtered.

8.2.2.4.2 Search results and discussion

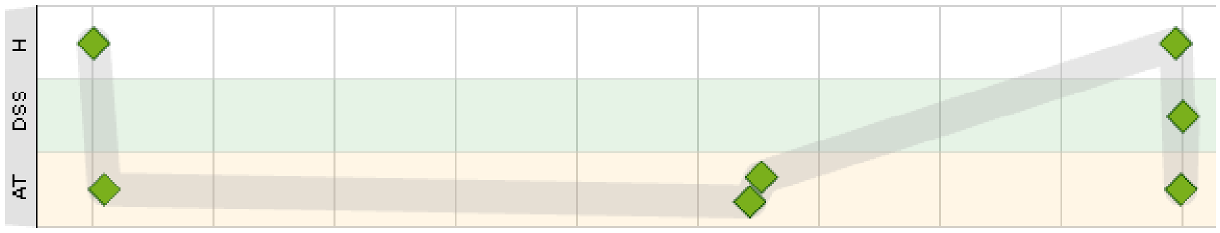
Using above described settings to narrow the search scope, the whole searching process executes more than 10 times faster than in scenario C2.a. The results have been rather surprising: We discovered that more than 8% of all tickets had a match of 75% similarity and higher. Figure 93 depicts the best matches regarding to the reassignments. As can be seen from the figure, each sequence contains reassignments among named departments. Only the order is switched, as we consciously omitted this factor.



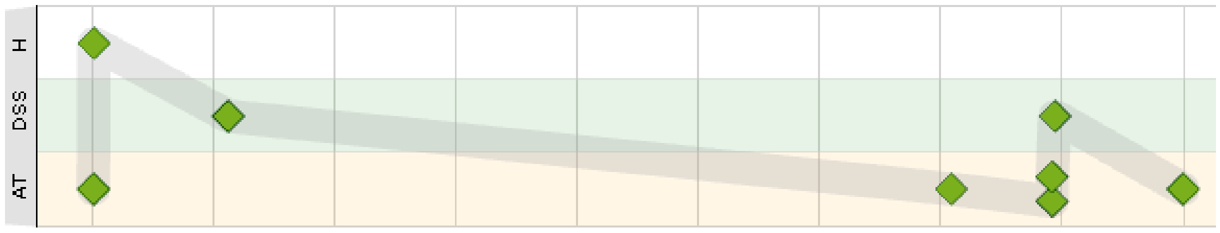
(a) – Ticket reassignments in search pattern



(b) – Match with sim=0.91



(c) – Match with sim=0.89



(d) – Match with sim=0.87

Figure 93: Best matches for evaluation scenario C2.b – Reassignments by support department over time

In addition, searching the limited data set (which can be hold in memory) drastically reduces the data retrieval time to about 1/5th of the time required when reloading sequence by sequence from the database.

The scenario shows that a targeted search, focusing on the current analysis question returns valuable results in short execution times. Yet, this requires a knowledgeable and skilled user, and also some data preprocessing, for instance to be able to load only sequences with more than 3 reassignments or the like. The only problem we encountered with this scenario was how to get accurate results with string similarities. In the given case, the naming convention for a ticket assignee was DEPARTMENT.SUPPORTGROUP.NAME. In the dataset we found entries such as "DSS.SUPPORT.ALL" or "CSS.SUPPORT.ALL". Looking at the string, these values are very similar whereas from there semantics, they are almost not similar as the groups are in different departments. We resolved this issue by splitting up the department substring into a separate event attribute which we considered in the search with a significantly higher weight.

8.2.2.4.3 Performance summary

Total number of events:	10095
Total number of event sequences:	372
Average number of events per event sequence:	27,14
Initial threshold:	∞

Scenario	Events in pattern	Total time	Algorithm time	Events/sec total	Sequences /sec total	Events/sec algorithm	Seq./sec algorithm
C2.b	28	00:00:05.83	00:00:04.71	1725,38	63,81	2143,31	79,98

Table 6: Performance results for evaluation scenario C2.b

8.2.2.5 C2.c – Considering alert events and the order of assignments

8.2.2.5.1 Search pattern and configuration

In scenario C2.b we focused on reassignments but ignored the order of these reassignments. In the next scenario, we considered not only the order in which the assignments happened, but also if the ticket was created for a certain server alert. Thus, the practical question we tried to answer was: For a certain server alert, is the opened ticket (re)assigned in multiple cases in the same way, between the same departments.

8.2.2.5.2 Search results and discussion

In the pattern sequence we chose, a server alert with the message "Disk space warning: only 4,97% free on disk […]" triggered a ticket to be created. This ticket was then first assigned to department "CSS", from "CSS" to "H" and to "AT" where it was reassignment several times within the department, then back to "CSS" and finally resolved. This sequence of reassignments is visualized over time in Figure 94 (sequence highlighted in violet).

The search among 10,000 events finished in less than 10 seconds, and revealed some interesting results: For instance, the best match of the search, depicted also in Figure 94 (grey sequence) showed a very similar sequence of reassignments, from "CSS" to "H", to "AT", only with some more reassignments within the individual departments. Interestingly enough, the ticket was also opened due to an initial alert, and this alert was again a disk space warning. The knowledge about such incidents is a good starting point for investigating in detail the support process in case of disk space warnings.

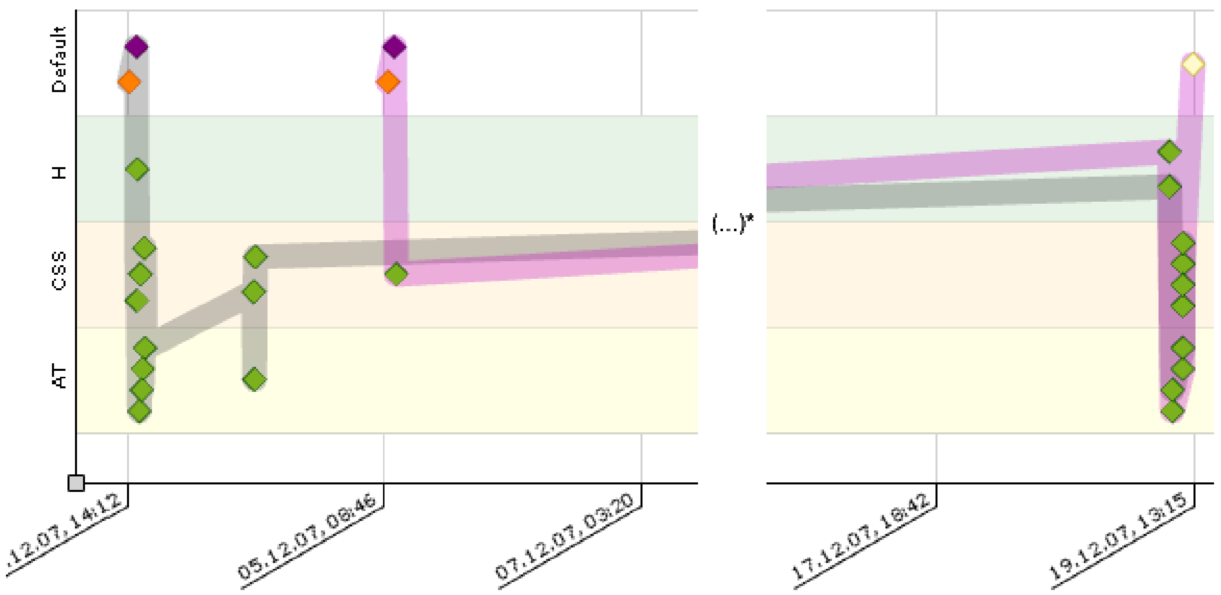


Figure 94: Search pattern and best match for evaluation scenario C2.c[11]

8.2.2.5.3 Performance summary

Total number of events:	10095
Total number of event sequences:	372
Average number of events per event sequence:	27,14
Initial threshold:	∞

[11] In the chart, two areas are marked with an asterisk. At these points in time, the data set showed a longer time period between the events which has been cut out in order to fit the figure to the page size.

Scenario	Events in pattern	Total time	Algorithm time	Events/sec total	Sequences /sec total	Events/sec algorithm	Seq./sec algorithm
C2.b	28	00:00:09.37	00:00:07.63	1077,37	39,70	1323,07	48,75

Table 7: Performance results for evaluation scenario C2.c

8.2.3 C3 Credit card transaction – sequences of purchases

8.2.3.1 Scenario and data structure

In scenario C3 we used a data set containing sequences of purchases from a credit card provider. As these data are highly confidential, all of the following results and considerations are expressed in terms of anonymous names for products, customers and purchase information.

The data set contains sequences of activities for a selected group of 4000 customers. These activities are, besides creating or closing the account, first of all "sales" events. These events reflect that a customer paid for something by credit card. In that case, we have the information on which shop that was (or ATM), the country and the paid amount available for the analysis. Figure 95 shows the occurring types of events and their attributes.

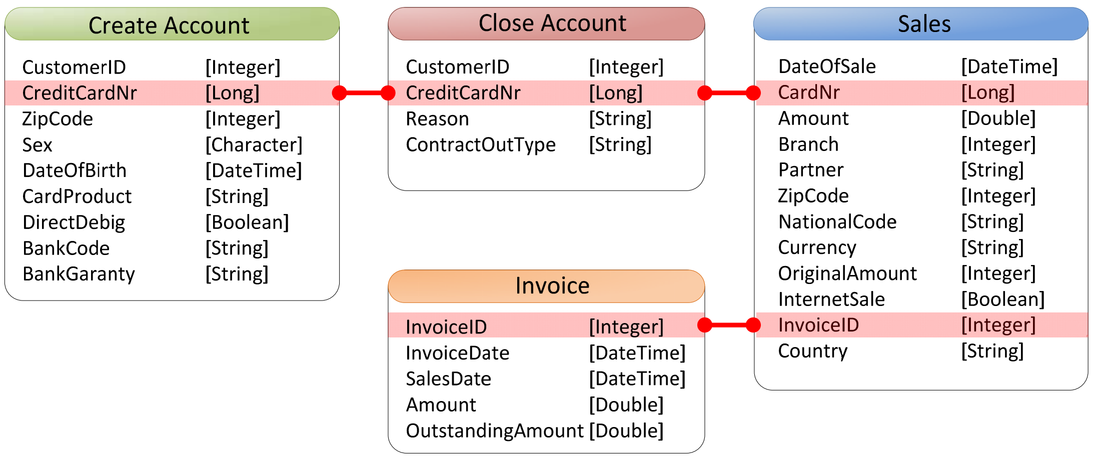


Figure 95: Event types and correlations in evaluation scenario C3 – credit card transactions

In the previous two evaluation scenarios, we focused on the retrieval quality and execution time in case of known pattern sequences for evaluating if other, similar sequences exist and if yes, in which extend these are similar in order to assess whether the given case is a reoccurring pattern.

In this scenario we focus on applying the similarity search in comparison to established and well-known data mining techniques. In the given case, we did an analysis of the raw dataset with RapidMiner[12]. The objective was to figure out if there are certain patterns in the customer behavior for customers whose accounts had to be closed due to illiquidity and thus unpaid invoices.

[12] RapidMinder by Rapid-I is an open-source data mining software, providing access to a whole range of data mining algorithms such as decision trees or lazy learners, association mining techniques and also data pre-processing and feature selection operators.

8.2.3.2 Objectives and evaluation focus

For the evaluation of our similarity search algorithm in the given context, we define the following objectives:

- Figure out if the similarity search is applicable for the given purpose
- Find possible improvements for supporting the analyst's workflow given a similar task

8.2.3.3 C3.a – Data integration and preprocessing

Up to this point, we haven't considered this aspect and started with data already being loaded to the event repository and ready to be searched. Yet, when talking about data mining, it is unavoidable to first talk about data integration and preprocessing.

8.2.3.3.1 Preprocessing for classical data mining

The most important preprocessing step in order to successfully apply existing data mining algorithms was the generation of additional attributes, in order to have an utmost complete attribute space. For instance, the occurrence date attribute had been split up into additional "month of the year", "day of week" and "week of month" attributes in order to make it accessible. The currency of the purchases showed too many distinct values with only a few occurrences each, which caused inappropriate or statistically insignificant results and had to be summarized to "EUR" and "not EUR". Sales amounts had to be categorized into equidistant classes, working with the discrete values was impossible.

8.2.3.3.2 Preprocessing for similarity search

Basically, the similarity search requires less preprocessing, as all attributes, i.e. also discrete values can be used and compared directly, without categorization. In addition, it is not necessary to extract attributes such as "day of week" into separate attributes, as calculated attribute expressions (see section xxx) can be used to extract such values on the fly.

8.2.3.3.3 Summary and discussion

With the use of calculated attribute expressions, the effort for preprocessing is minimal in our approach. Discrete values don't need to be categorized and attribute expressions add artificial event attributes on-the-fly during the comparison which can then be weighted accordingly. Yet, in order to optimize performance of the searching process, we still recommend extracting derived values into separate event attributes during the data integration to save computation time.

8.2.3.4 C3.b – Getting started with the mining process

The next question after preprocessing is how to start the data mining. Below we discuss be situation we faced.

8.2.3.4.1 Getting started with the "classical" data mining

Among the existing data mining approaches, we decided to apply a classification and regression tree (CART) in order to derive simple rules such as "if customers buy more than 4 times in branch *X* and pay in currency *Y*, the probability for illiquidity is 91%". In fact, in order to get started with the mining process, profound knowledge on the existing techniques is required in order to choose the right algorithm for the given purpose, but despite of that only some configuration parameters have to be set.

8.2.3.4.2 Getting started with the similarity search

The goal with similarity search was to find a sequence of certain purchases, which is reoccurring in multiple cases of known customer illiquidity. Obviously, the similarity search engine cannot be directly compared to data mining algorithms such as decision trees or other learners in general. The greatest problem we had in the given case was that we did not have any assumptions or reference cases to be checked for occurrence and validity. Thus, the only thing possible was to pick a sequence more or less by chance and try to search for similar occurrences. We tried picking several sequences, starting with the one customer where most money was lost. Yet, this cannot be called a structured and systematic approach.

8.2.3.4.3 Summary and discussion

The use case shows the necessity to embed the similarity search in a greater context, for instance in the form of a clustering algorithm, which forms groups of similar sequences based on multiple similarity comparisons. As is, only a punctual search is possible. Without initial knowledge on the dataset, it is hard to model a suitable reference pattern.

8.2.3.5 C3.c – Finding sequences of purchases

Finally, taken said limitation into account that we can only pick certain pattern sequences by chance and not automatically investigate the whole data set, we tried to discover sequences of similar purchases for one selected reference pattern.

For the search, we limited the whole dataset of 182.023 events to 14.034 events of those customers, whose accounts have been closed. In total, these are 348 of 98.355 customers. For the search, Levenstein string similarity, which performed quite well in scenario C2 was used for the attributes “Sales.Partner” (i.e. the shop where a purchase took place), “Sales.Currency” and “Sales.Country”. For “Sales.Amount”, normalized absolute difference similarity was uses, as well as Boolean similarity for the attribute “Sales.InternetSale”.

Figure 96 shows how the sales events in the selected pattern sequence are distributed with respect to the product branch (Figure 96a) and the country (Figure 96b).

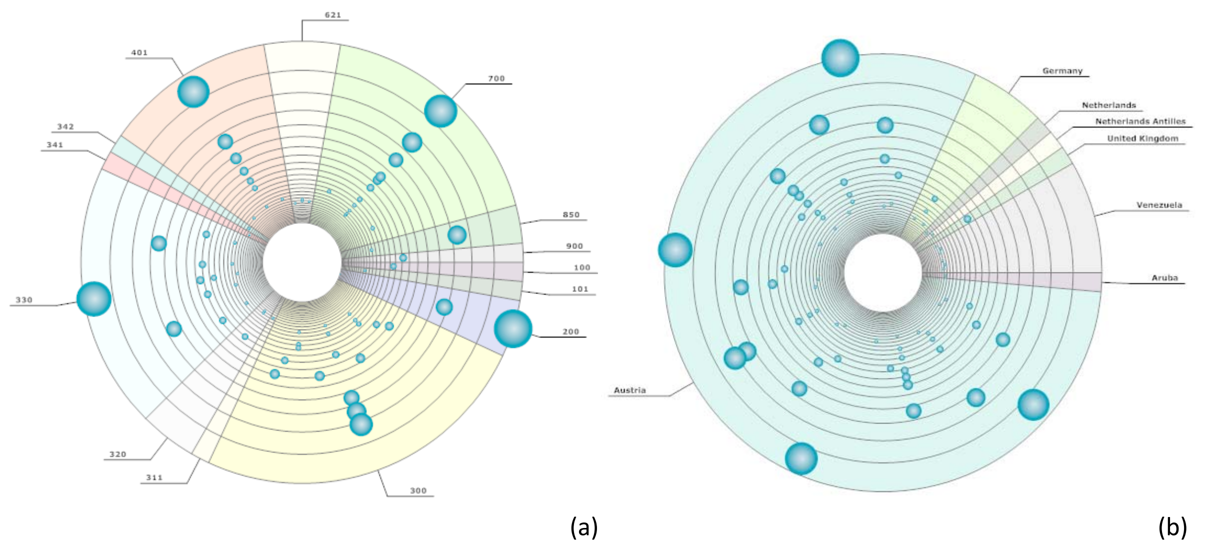


Figure 96: Search pattern events for evaluation scenario C3.c

8.2.3.5.1 Search results and discussion

Given the selected pattern sequence and configuration, the algorithm failed to return valuable results. We tried to adjust the weights of the considered attributes, but the pattern remained too long and too specific to be rediscovered in the data.

The apparent problems are in particular:

- The pattern sequence contains 65 sales events. Sequences with a lower number of events have to be mapped using several null-mappings. Depending on the null-mapping costs, this decreases the similarity score drastically and these sequences soon fall below the threshold. On the other hand, if the null-mapping costs are low, solutions using a log of null-mappings might be preferred over solutions taking the available events.
- The length of the event sequences in the data set varies from 10 up to 530 events. For such a length of an event sequence, a huge amount of solutions exist, and the approach of considering the single events is probably not appropriate any longer. Rather, aggregation would be required.
- When looking at the rules derived from the CART, these patterns could not be discovered with the similarity search, because they are "overruled" in the matching process by the whole range of additional events, which are not statistically cumulating in the pattern. In other words, even if we know that 4 purchases in branch 123 in Germany have always been followed by illiquidity in the past, it might be that we still do not discover such an event sequence as it contains, aside of these 4 events, maybe another 100 purchases, all decreasing the similarity to the reference pattern.
- For very long event sequences, the weight of a single event is minimal. Thus, the matching process continuously has to build up huge solution trees before reaching the similarity threshold. This problem is yet inherent to the chosen approach and could only be omitted by either techniques to detect huge deviations earlier in the matching process or weighting events at earlier stages of the mapping processes stronger compared to the rest in order to reach the threshold faster, if a solution is bad. At the same time, this distorts the correctness of results.

In summary, the evaluation scenario pointed out a set of shortcomings or missing features in the current approach, some of which will be discussed again in the future work section.

8.2.4 C4 Algorithmic trading – trading scenario discovery

8.2.4.1 Scenario and data structure

The term *algorithmic trading* in general refers to using computer programs for submitting trading orders. Hereby computer algorithms are used to determine various parameters of the trade, starting from the traded amount to the completely automatic selection of the instrument to be traded and the time at which it is bought or sold. A more detailed discussion on algorithmic trading and its possibilities and techniques is beyond the scope of this work. Yet, we picked out one aspect from the algorithmic trading domain and tried to apply the proposed event sequence similarity algorithm for this selected application example.

For the fourth evaluation scenario, we took freely available, historic data from the stock market. These include the end-of-day price of about 14,000 stocks over the course of a year, and news, whereby each news item is wrapped into one single news event. News items belonging to a certain stock are correlated with the sequence of stock tick events. Figure 97 shows the event types, their attributes and the defined event correlations.

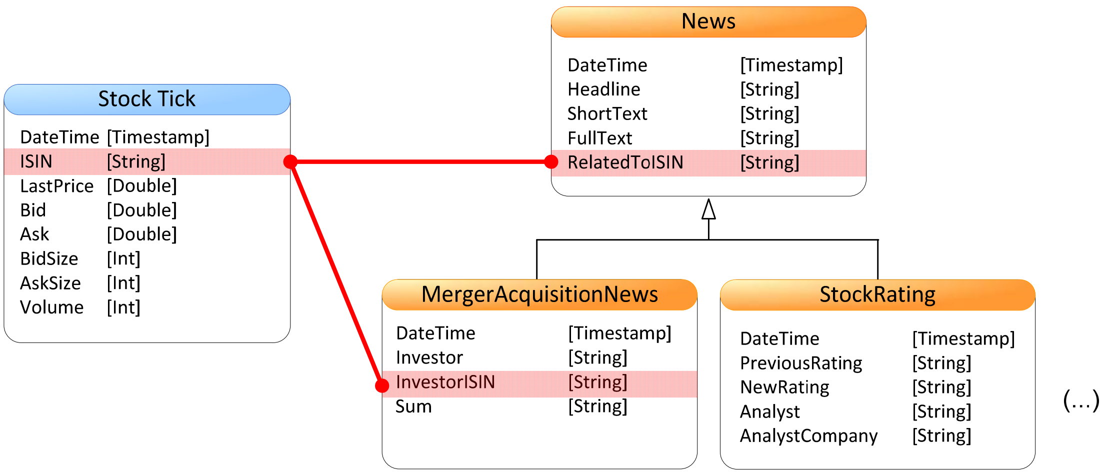


Figure 97: Event types and correlations in evaluation scenario C4 – Trading scenarios

For the scenario, we tried to discover defined trading scenarios. Many traders trade (explicitly or implicitly) in terms of scenarios. For instance, if the price chart forms a certain pattern and the traded volume is increasing at the same time, they buy or sell.

8.2.4.2 Objectives and evaluation focus

For the evaluation of our similarity search algorithm in the given context, we define the following objectives:

- Discovering past occurrences of trading scenario based on price and volume patterns and the combination thereof.
- Discovering trading scenarios based on the combined occurrence of price patterns and occurrence of news events.

8.2.4.3 C4.a – Type matching and normalized absolute difference similarity

During the planning phase, theoretical considerations on this scenario already pointed towards the necessity for enhanced event attribute techniques and a mechanism to consider a series of attribute values in particular. With the evaluation scenario we wanted to judge if made assumptions hold true and the base algorithm with normalized absolute difference similarity is really insufficient for the use case.

8.2.4.3.1 Search pattern and configuration

We searched the complete dataset for the following event sequence, plotted in the event chart with the attribute “price” on the y-axis and the occurrence time of events on the x-axis.

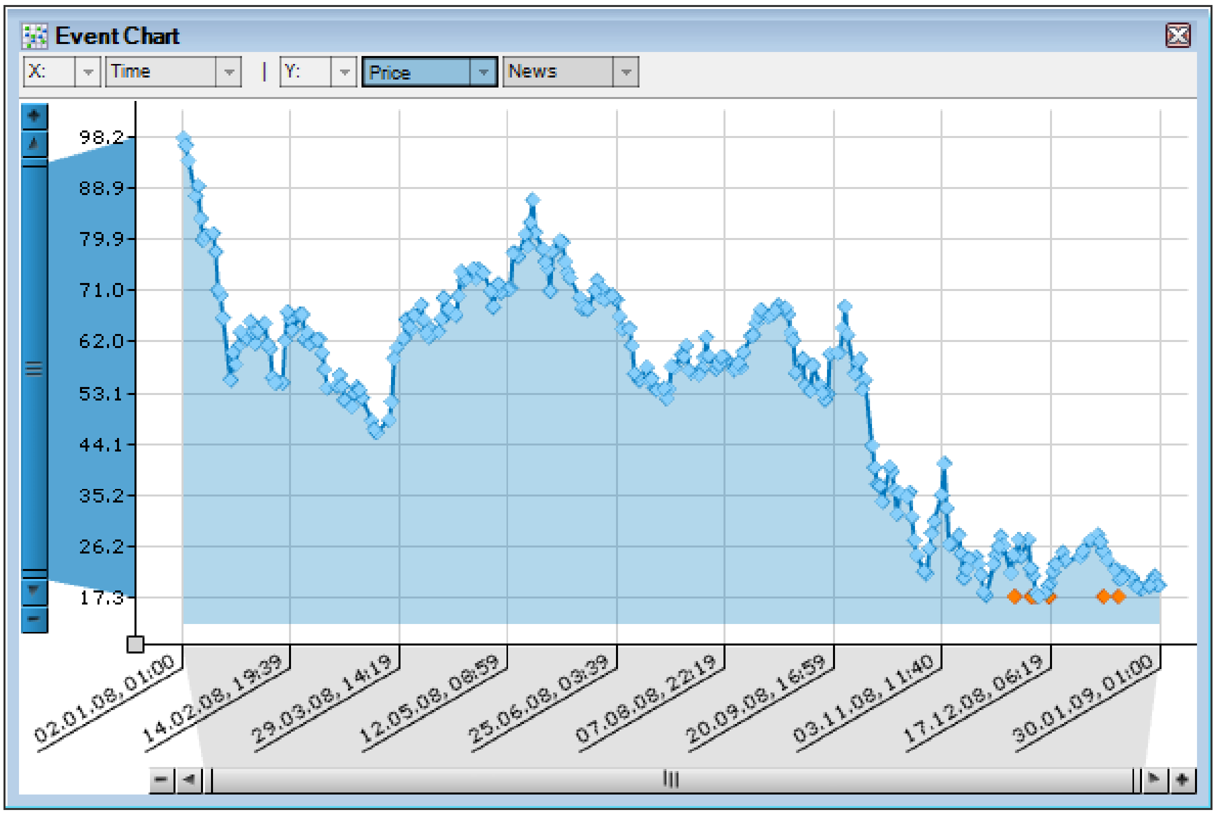


Figure 98: Pattern sequence for evaluation scenario C4.a

We chose the following configuration:

- Match must start with first event: False
- Match must end with last event: False
- Time matching mode: Relative
- Attribute similarities: StockTick.PastPrice – Normalized absolute difference similarity

8.2.4.3.2 Results and discussion

Unsurprisingly, the algorithm does not terminate within several hours or even days. We cancelled the evaluation at a certain stage as the search time already exceeded any practically acceptable limits. Taking a look at the data set, this is explained easily. Figure 99 shows several of the occurring event sequences in the dataset. Each row in the time line view represents one event sequence. Each sequence contains about 290 stock tick events and a few news events. With the current configuration, we consider now the order of event occurrences. As most events are of the same event type, we almost hit the algorithm's worst case, with most events being compatible to each other. In addition, the time deviations between many stock tick events are minimal and thus all can be taken into consideration for a possible mapping.

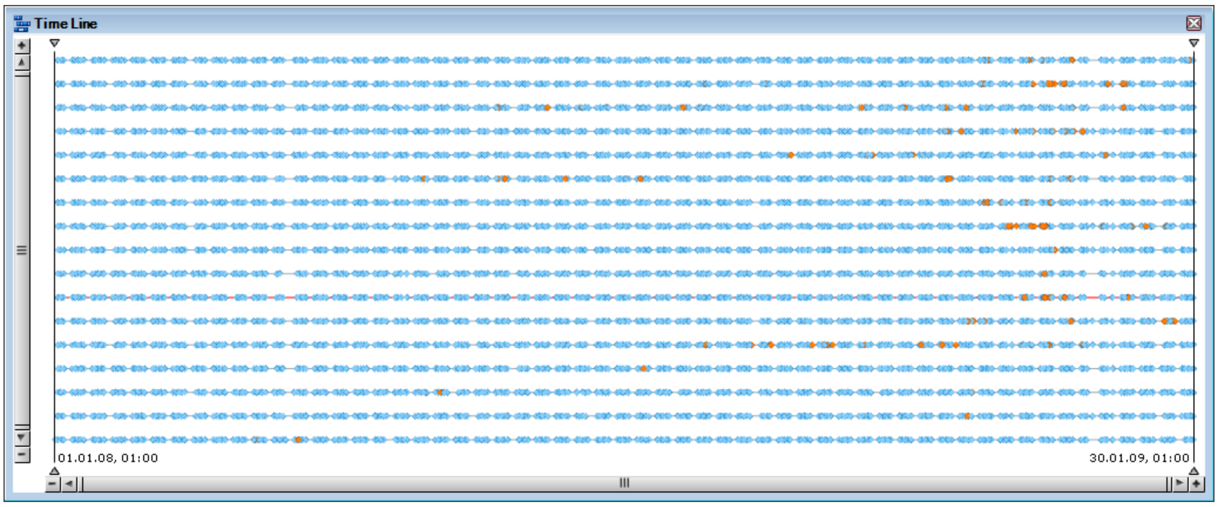


Figure 99: Several event sequences from the data set used for evaluation scenario C4.a

8.2.4.4 C4.b – Using time-series similarity for a single event attribute

Scenario C4.a not only confirmed our assumption that the default base algorithm will fail in the given context, but it also proved already that the integration mode of time-series, performing the type matching first, is not applicable for this dataset. Therefore, for the following scenarios we chose time-series similarity for at least one numeric event attribute of the stock-tick events, and executed it in pre-matching fashion.

8.2.4.4.1 Search pattern and configuration

In C4.b we utilized the same pattern sequence as in the previous scenario and chose the following configuration:

Match must start with first event:	False
Match must end with last event:	True
Time matching mode:	-- (event occurrences times not considered)
Order deviations:	--
Attribute similarities:	StockTick.LastPrice – Normalized sequence similarity
Integration mode of time-series:	Pre-matching execution

8.2.4.4.2 Results and discussion

As this scenario reflects a simple time-series similarity search retrieval results are not discussed separately at this stage. Instead, we refer the reader to Appendix B where detailed results of the separately carried out time-series evaluation are documented. At this stage, the one interesting aspect remaining is the performance of the matching process when integrated into the event similarity framework. As can be seen below, again the data retrieval and the extraction of the attribute values from events before having them at hand cause some overhead. On the other hand, for such a big dataset of over four million records it is simply not possible to hold them in memory (at least not in the form of event objects). In total, the overhead time is around 50%.

8.2.4.5 C4.c – Using time-series similarity for combined price and volume patterns

So far, we did not make use of the ability to combine multiple aspects of the event sequence. In scenario C4.c, now two time-series, i.e. the price series and the volume curve should be considered in combination.

8.2.4.5.1 Search pattern and configuration

In C4.c we utilized the same pattern sequence as in the previous scenario and chose the following configuration:

Match must start with first event:	False
Match must end with last event:	True
Time matching mode:	Relative
Attribute similarities:	StockTick.LastPrice – Normalized sequence similarity
	StockTick.Volume – Normalized sequence similarity
Integration mode of time-series:	Pre-matching execution

The volume curve for the search pattern (depicted in Figure 98) looks as shown below:

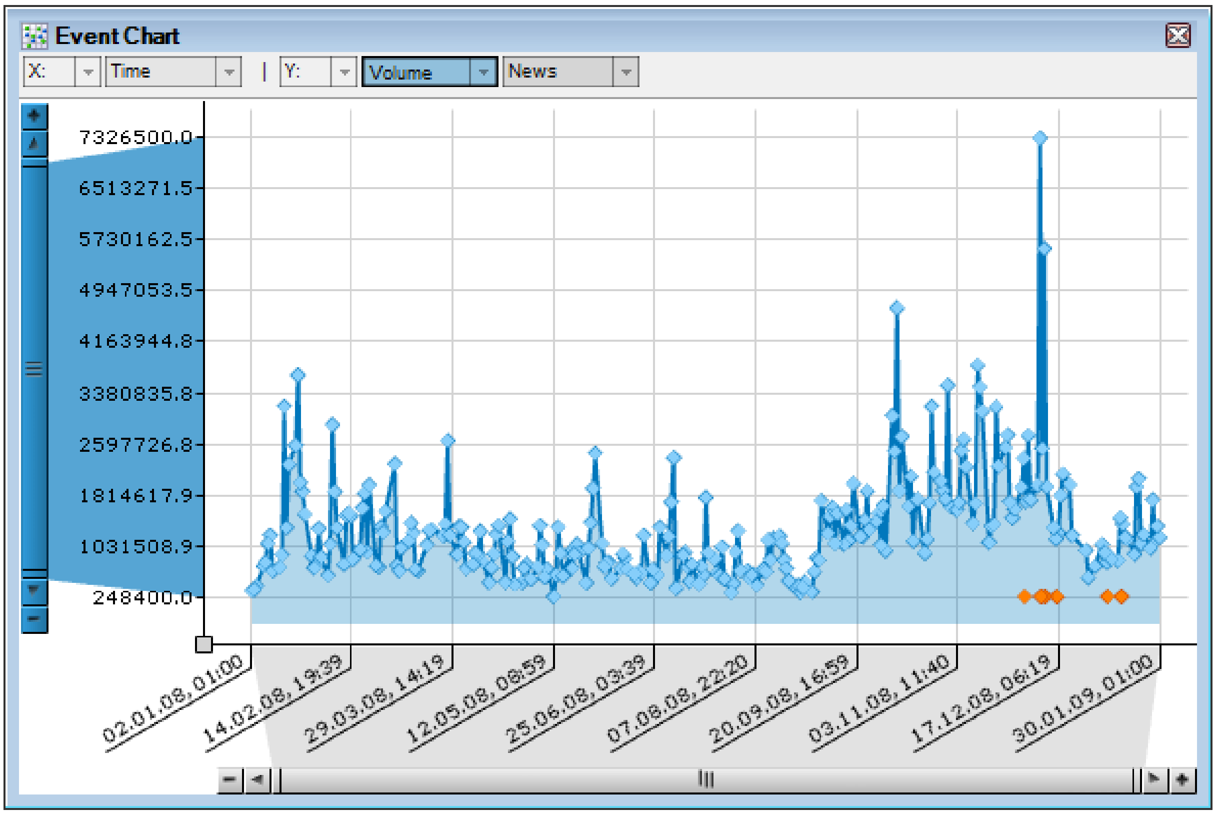


Figure 100: Volume curve for the stock tick events in evaluation scenario C4.c

8.2.4.5.2 Results and discussion

The searching process considering two numeric attribute series is in total only slightly slower than then the searching considering only one sequence. The re-execution of a full-sequence matching process, which is required if the best matches of the attribute series are not overlapping, brings in relation to the normal matching process no remarkable performance overhead. In addition, the time-series matching for the two considered event attributes can be executed in parallel which is the reason for the algorithmic time not to double in comparison to scenario C4.b.

In the dataset, a whole range of considerably good matches could be discovered, showing combined similarities (from price and volume curve) of up to 96%.

8.2.4.6 C4.d – Combining time-series patterns with news events

In the last evaluation scenario, C4.c is extended by considering also the temporal structure and occurrence of news events. In comparison to C4.a, the temporal structure and occurrence of event types has only to be computed based on the best match after executing the time-series similarity, which is fast and straight-forward.

8.2.4.6.1 Search pattern and configuration

C4.d uses the same search pattern as before and the following parameters:

▪ Match must start with first event:	False
▪ Match must end with last event:	True
▪ Time matching mode:	Relative
▪ Order deviations:	Considered
▪ Attribute similarities:	StockTick.LastPrice – Normalized sequence similarity
▪ Integration mode of time-series:	Pre-matching execution

8.2.4.6.2 Results and discussion

As the recomputation of the time deviation and order deviation costs is fast, the search executes only slightly slower compared to scenario C1.c. In the configuration, we weighted order deviations and time deviations almost equal. From the results we found that in practice, the occurrence time of the news should be weighted much higher. The results can further be improved by using occurrence number blocks for the news events, in order to guarantee only at which time news should be present, but to not be influenced too strongly by the actual number of the news items at this time. The rationale is simply that in many cases, the same news content has been captured multiple times from different news channel, which distorts similarity results.

8.2.4.7 Performance summary

Total number of events:	4.010.272
Total number of event sequences:	13.909
Average number of events per event sequence:	288,32

Scenario	Events in pattern	Total time	Algorithm time	Events/sec total	Sequences /sec total	Events/sec algorithm	Seq./sec algorithm
C4.a	282	-- Not terminated --					
C4.b	282	00:35:33.1	00:19:01.4	1880,02	6,52	3418,52	11,85
C4.c	282	00:41:11.4	00:25:42.1	1622,67	5,62	2600,52	9,01
C4.d	282	00:42:01.9	00:27:11.4	1590,17	5,51	2458,17	8,52

Table 8: Performance results for evaluation scenario C4

9 Summary, conclusions and future work

In this thesis, we presented a structured and comprehensive approach for assessing the similarity of event sequences. We showed how to integrate this similarity computation into an event querying and visualization framework, the SENACTIVE *EventAnalyzer*^{TM} in order to provide a mechanism for fuzzy retrieval of event sequences based on a reference pattern. The similarity assessment model is able consider multiple aspects of event sequences, which can be roughly categorized into event type occurrence, temporal structure and single event attribute similarities. In addition, attribute value sequence similarity has been introduced, considering the complete value sequence of attribute values extracted from multiple events, instead of event-by-event attribute comparisons. In terms of single event similarity, we propose a range of default attribute techniques, reaching from normalized absolute difference similarity for numeric attributes over the integration of string similarity measures to calculated attribute expressions. In addition, we proposed a set of building blocks in order to refine the search pattern, for instance to define allowed occurrence numbers of events, weaken the pattern by allowing a selected number of arbitrary events, time constraints and many more. At the end, a linear aggregation of all similarity-relevant features is performed by combining costs of the manifold deviations, and the result is transformed into a final similarity score between 0 (no similiarty) and 1 (equality with respect to all considered features).

The similarity assessment model resulted from analyzing the requirements of different use case domains, all of which showed very different characteristics and difficulties. For instance, in order to discover trading scenarios in the algorithmic trading domain, time-series similarity has been introduced to find similar price series. Here, a pure matching process based on the occurrence of certain event types fails as such sequences only contain regular “stock tick” events. At the same time, in other application areas where business processes execute more or less according to a template (for instance shipment processes, airport turnaround processes, etc.) matching based on event type occurrence and time deviations is appropriate.

It was a defined objective of this research effort to define a framework flexible enough for being adjusted as required for different data sets and application domains, as well as to be open for extensions. This reflects in leaving a lot of control up to the user who can select from different event attribute similarity techniques to be applied for each event type. Furthermore, every single cost factor can be weighted and finally a set of modeling options exist in a graphical search pattern editor. The evaluation shows that the generic character of the similarity framework enables the application for different types of datasets without tailoring the matching process exclusively to one of these cases. We hope that in the future further, not yet considered application areas will be identified and can be covered by adjusting the existing techniques. At the same time, this flexibility comes at a price, which is, that the applied configuration has to be chosen carefully. It is up to this point definitely not a “one-click-search” experience. Instead, in order to retrieve valuable results the whole investigation has to be well-thought, starting from the data integration to a targeted configuration focusing exactly on the analysis questions. Yet, we see the possibility to provide ready-to-use configuration packages in the form of solution templates in the future. Having a suitable configuration set, the proposed integration into the user interface and workflow ease the access to typically complex data mining features, one of which similarity search definitely is.

The presented reference implementation focused in particular on enhanced techniques for considering different semantics of attributes such as named time-series similarity and on modeling the pattern sequence. The evaluation showed that the concepts work well for event sequences up to around 20-50 events. In case of longer event sequences the execution time increases drastically. Though, the evaluation scenario looking at credit card transactions showed that in general it is questionable if such as detailed matching makes sense at all for long event sequences. At some stage such activity sequences contain loads of events surrounding the events which could potentially make up a pattern so that they overrule the high similarity values of well-matching subsequences. One could refer to this effect as kind of noise. The apparent problem is though that it is not known up-front, which events will be similar and which are noise. As already partly discussed in the evaluation section, a user activity of 500 actions (e.g. bet placements, purchases, etc) will hardly reoccur more than once even in a large data set. Instead, multiple small sub-sequences might be common and would be interesting to identify.

The proposed method for comparing time-series is more robust with respect to execution time in case of long event sequences. It is based on the idea of measuring and comparing the slopes between pairs of curve sections. Hereby, the original curve is not sliced at a regular frequency, but at turning points where the major orientation of the curve turns up or down. For the identification of these turning points a technique from the financial market analysis is utilized, detecting trend reversals based on the moving average. By varying the calculation period of the moving average, these turning points can be identified at different granularities. The approach is invariant to the scale of the absolute curve-point values and also supports for subsequence matching.

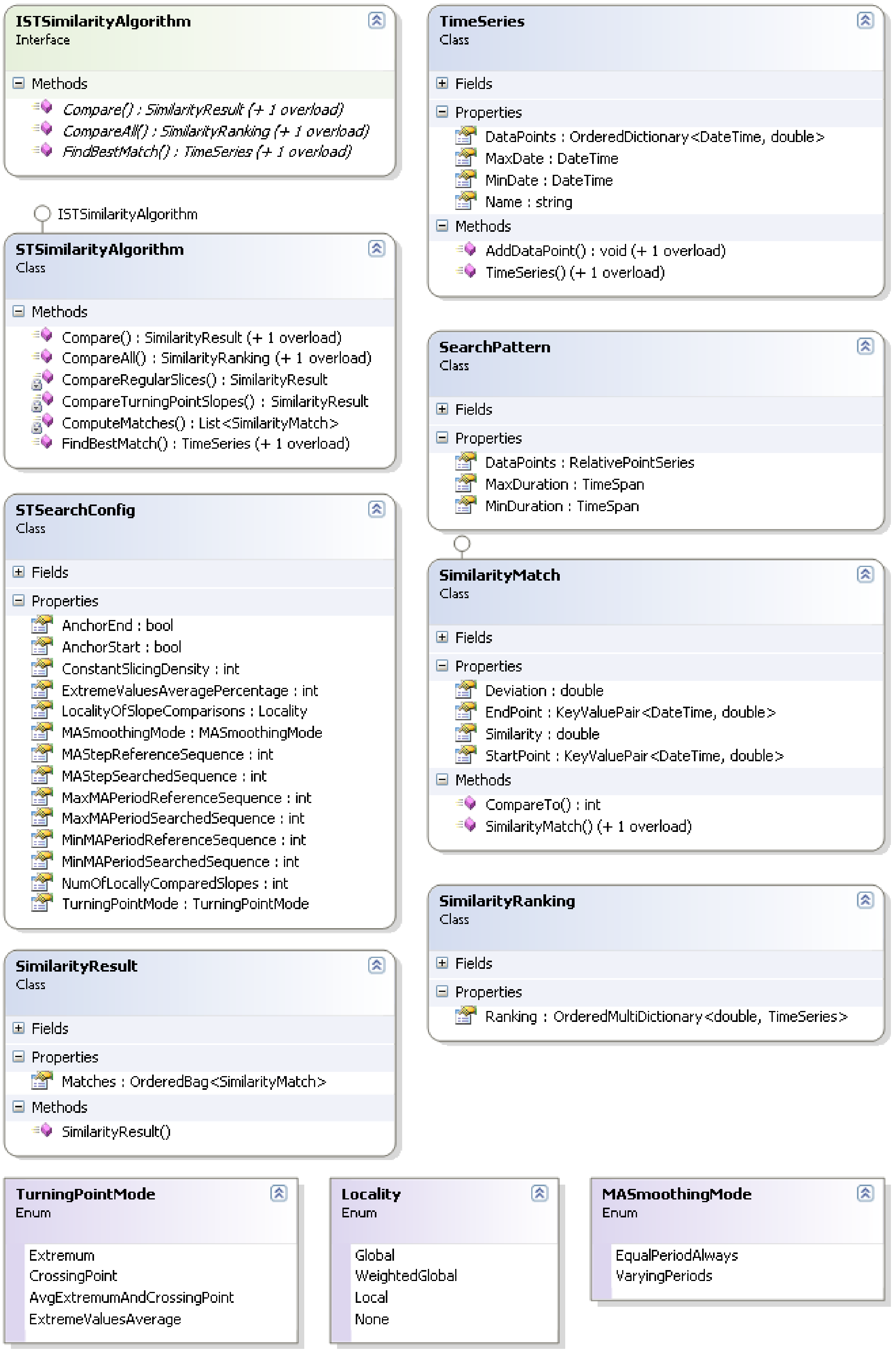
In total, we consider the presented similarity assessment model as a solid basis to apply it for the discovery of event sequences based on a reference pattern. Throughout the works on the basic features, we identified a set of further requirements which would be interesting to cover in various application domains. One potentially highly valuable extension could be to build up a clustering mechanism based on multiple, automatic similarity comparisons. Such a clustering could separate business entities such as customers into different groups based on their behavior, which reflects in event sequences. Having such a mechanism in place would make the whole search more applicable for data mining in general. Currently, as the evaluation underlined, it supports a selective searching process only. Without knowing an interesting reference sequence or pattern, it is hard to get started at all.

Another requirement we figured out was a dissimilarity search: In domains where business processes execute strictly according to a process template, especially those event sequences are of interest, where a strong deviation of any kind is present. Currently, such a query is hardly possible because in the given model only one deviating event sequence could be taken to find similar deviations. Other, unknown deviations could not be found. Still, such a feature could easily be provided by keeping the worst-matching sequences instead of the best-matching ones. Yet, maybe for dissimilarity search other algorithmic approaches are more efficient.

Regarding to applications for the similarity assessment model, forecasting could be a further topic of interest. A forecast of key figures and the possible outcome of a business process could be based upon similar historic event sequences. An example would be forecasting fraud in online gambling based on suspicious behavior.

Finally, further research effort could be spend into a dynamic handling of the similarity threshold. For longer event sequences, the threshold is crucial in order to omit bad matches as early as possible and thus keep the execution time low. At the same time, it is hard to figure out at the beginning of the searching process which threshold is appropriate. We propose to integrate an automated threshold control, which starts at a very restrictive threshold and weakens it step-by-step if insufficient results are retrieved.

Appendix A – The STSimilarity library



```
ISTSimilarityAlgorithm
Interface
Methods
  Compare() : SimilarityResult (+ 1 overload)
  CompareAll() : SimilarityRanking (+ 1 overload)
  FindBestMatch() : TimeSeries (+ 1 overload)

ISTSimilarityAlgorithm

STSimilarityAlgorithm
Class
Methods
  Compare() : SimilarityResult (+ 1 overload)
  CompareAll() : SimilarityRanking (+ 1 overload)
  CompareRegularSlices() : SimilarityResult
  CompareTurningPointSlopes() : SimilarityResult
  ComputeMatches() : List<SimilarityMatch>
  FindBestMatch() : TimeSeries (+ 1 overload)

STSearchConfig
Class
Fields
Properties
  AnchorEnd : bool
  AnchorStart : bool
  ConstantSlicingDensity : int
  ExtremeValuesAveragePercentage : int
  LocalityOfSlopeComparisons : Locality
  MASmoothingMode : MASmoothingMode
  MAStepReferenceSequence : int
  MAStepSearchedSequence : int
  MaxMAPeriodReferenceSequence : int
  MaxMAPeriodSearchedSequence : int
  MinMAPeriodReferenceSequence : int
  MinMAPeriodSearchedSequence : int
  NumOfLocallyComparedSlopes : int
  TurningPointMode : TurningPointMode

SimilarityResult
Class
Fields
Properties
  Matches : OrderedBag<SimilarityMatch>
Methods
  SimilarityResult()

TimeSeries
Class
Fields
Properties
  DataPoints : OrderedDictionary<DateTime, double>
  MaxDate : DateTime
  MinDate : DateTime
  Name : string
Methods
  AddDataPoint() : void (+ 1 overload)
  TimeSeries() (+ 1 overload)

SearchPattern
Class
Fields
Properties
  DataPoints : RelativePointSeries
  MaxDuration : TimeSpan
  MinDuration : TimeSpan

SimilarityMatch
Class
Fields
Properties
  Deviation : double
  EndPoint : KeyValuePair<DateTime, double>
  Similarity : double
  StartPoint : KeyValuePair<DateTime, double>
Methods
  CompareTo() : int
  SimilarityMatch() (+ 1 overload)

SimilarityRanking
Class
Fields
Properties
  Ranking : OrderedMultiDictionary<double, TimeSeries>

TurningPointMode
Enum
  Extremum
  CrossingPoint
  AvgExtremumAndCrossingPoint
  ExtremeValuesAverage

Locality
Enum
  Global
  WeightedGlobal
  Local
  None

MASmoothingMode
Enum
  EqualPeriodAlways
  VaryingPeriods
```

Interface ISTSimilarityAlgorithm

The ISTSimilarityAlgorithm is the interface to the main comparison operations. All provided operations can be accessed via this interface and the below described methods.

Method	Summary
Compare Parameters: ▪ SearchPattern pattern ▪ TimeSeries sourceSeries ▪ STSearchConfig config Return value type: ▪ SimilarityResult	Compares a source series to a search pattern and returns a similarity result with a sorted list of matches.
Compare Parameters: ▪ TimeSeries pattern ▪ TimeSeries sourceSeries ▪ STSearchConfig config Return value type: ▪ SimilarityResult	Compares a source series to an input series and returns a similarity result.
FindBestMatch Parameters: ▪ TimeSeries pattern ▪ ICollection<TimeSeries> sourceSeries ▪ STSearchConfig config Return value type: ▪ TimeSeries	Finds the time series matching best a given reference sequence. Make sure that the pattern sequence is not contained in the collection of input series!
FindBestMatch Parameters: ▪ SearchPattern pattern ▪ ICollection<TimeSeries> sourceSeries ▪ STSearchConfig config Return value type: ▪ TimeSeries	Finds the time series matching best a given reference pattern.
CompareAll Parameters: ▪ TimeSeries pattern ▪ ICollection<TimeSeries> sourceSeries ▪ STSearchConfig config Return value type: ▪ SimilarityRanking	Compares a set of source series to a reference sequence. The result is a ranking of similarity matches.
CompareAll	Compares a set of source series to a reference pattern. The

Parameters: ▪ SearchPattern pattern ICollection<TimeSeries> sourceSeries ▪ STSearchConfig config Return value type: ▪ SimilarityRanking	result is a ranking of similarity matches.

Table 9 Methods of ISTSimilarityAlgorithm

Class STSearchConfig

The STSearchConfig class is the configuration object containing all parameters for time-series similarity search. In order to achieve reasonable matching results for a certain application domain, these parameters must me configured advisedly.

Field	Summary
MinMAPeriodReferenceSequence Type: Int	Gets or sets the minimum moving average (MA) period applied for smoothing the reference sequence and extracting the turning points.
MaxMAPeriodReferenceSequence Type: Int	Gets or sets the maximum moving average (MA) period applied for smoothing the reference sequence and extracting the turning points.
MinMAPeriodSearchedSequence Type: Int	Gets or sets the minimum moving average (MA) period applied for smoothing the searched sequence and extracting the turning points.
MaxMAPeriodSearchedSequence Type: Int	Gets or sets the maximum moving average (MA) period applied for smoothing the searched sequence and extracting the turning points.
MAStepReferenceSequence Type: Int	Step-length for the reference sequence: The algorithm goes stepwise from minimum to maximum MA period. This parameter determines the step-length inbetween. Minimum value is 1. The lower the value, the higher the exactness of the algorithm and the more iterations are necessary.
MAStepSearchedSequence Type: int	MA step-length for the searched sequence.

AnchorStart Type: bool	Determines whether a match must start with the first point of the time-series.
AnchorEnd Type: bool	Determines whether a match must end with the last point of the time-series.
LocalityOfSlopeComparisons Type: Locality (enum)	The locality mode of slope comparisons. Determines which slopes are compared to each other. Global comparison compares each value to each other. Weighted global weights direct neighbours stronger. Local mode compares only a set of local/near values.
NumOfLocallyComparedSlopes Type: Int	This parameter is only relevant in case of Locality set to "Local". Determines how many values before and after the concerned slope are considered.
TurningPointMode Type: TurningPointMode (enum)	Gets or sets the mode how turning points are determined. “Extremum” takes the highest or lowest value of the time-series “CrossingPoint” takes the point where a trend reversal is detected (which is always later than the actual reversal, of course) “AvgExtremumAndCrossingPoint” takes an averaged value between the value at the trend reveral detection and the extreme value since last trend reveral “ExtremeValuesAverage” forms an average of the highest or lowest x% of values between the reversal points.
ExtremeValuesAveragePercentage Type: Int	Relevant only for turning point mode "ExtremeValuesAverage". Determines how many percent of highest or lowest values are averaged for the turning point computation.
MASmoothingMode Type: MASmoothingMode (enum)	Two possible modes for the different passes between minimum MA period and maximum MA period. “EqualPeriodAlways” - it is iterated from minimum of both searched sequence and reference sequence min MA period to maximum of both periods, in the minimum of steps-lengths. The same MA period is used for a comparison of searched and reference sequence. In this mode matches must have a similar overall time scaling. “VaryingPeriods” - Each MA period is compared to each other to find best matches. Used when the pattern can be

	matched also at a very different scaling.
ConstantSlicingDensity Type: Int	This factor is relevant only in case of AnchorStart and AnchorEnd both set to true. In this case the algorithm switches to a mode of constantly slicing the sequence into regular pieces and comparing slopes inbetween. The densitiy determines for how many data points one slice point is generated (kind of a sampling rate).
WeightBySubsequenceLength Type: Boolean	This flag determines whether during the slope comparison, the deviation is weighted relatively to the length of the subsequence between the two curve points. Setting this parameter to true means that a slope deviation of a shorter subsequence is less relevant than the deviation of a longer subsequence. It is relevant in case of irregular turning points extracted: In such a case, short slopes between certain turning points may bias the matching process.

Table 10 STSearchConfig fields with configuration options for the time-series algorithm

Appendix B – Evaluation results time-series similarity model

In the course of this work, a separate evaluation of the proposed time-series similarity search model has been carried out. The rationale of this separate evaluation was to isolate the time-series search performance from the overall event sequence search and assess in details its strengths and weaknesses.

Setup

For the evaluation, price history data of stocks taken from YAHOO Finance[13] have been utilized. We have written a tool to download the free end-of-day data, which is available in CSV format via a download link[14]. In order to automate the downloading of multiple historic price series, our sample data loading tool first parses the YAHOO finance webpage which lists all available symbols (i.e., short names for stocks) and then one-by-one requests the CSV download. In total, 2982 price series have been chosen for which regular data is available. Each series holds the data for at least 1 year.

For the evaluation, the downloaded data are searched for defined reference patterns. Accuracy of the result is judged by plotting the ranked similarity results in comparison to the search pattern. Evaluation speed is measured in terms of execution time.

For the tests, the following hardware has been used:

- DELL Dimension 9200 with Intel Core 2 CPU 6400, 2*2,13 GHz, 2GB RAM

Evaluation results pattern searching

The pattern searching evaluation is defined as a time-series search for a given search pattern with relative data points. For the presented slope-based time-series searching algorithm, this means in particular that the pattern data points are understood directly as the turning points of the search pattern. No moving average smoothing is done therefore on pattern side. Only for the searched sample series, the turning point extraction is carried out.

The sample search patterns defined are 5 typical price series movements in the financial market:

[13] http://finance.yahoo.com

[14] At the time of writing this thesis, data are available via the following URL:
http://ichart.finance.yahoo.com/table.csv?s={0}&a={1}&b={2}&c={3}&d={4}&e={5}&f={6}&g={7}&ignore=.csv
The parameters are: Yahoo symbol name, start month, start data day, start date year, end date month, end date day, end date year, data granularity (e.g. d for daily).

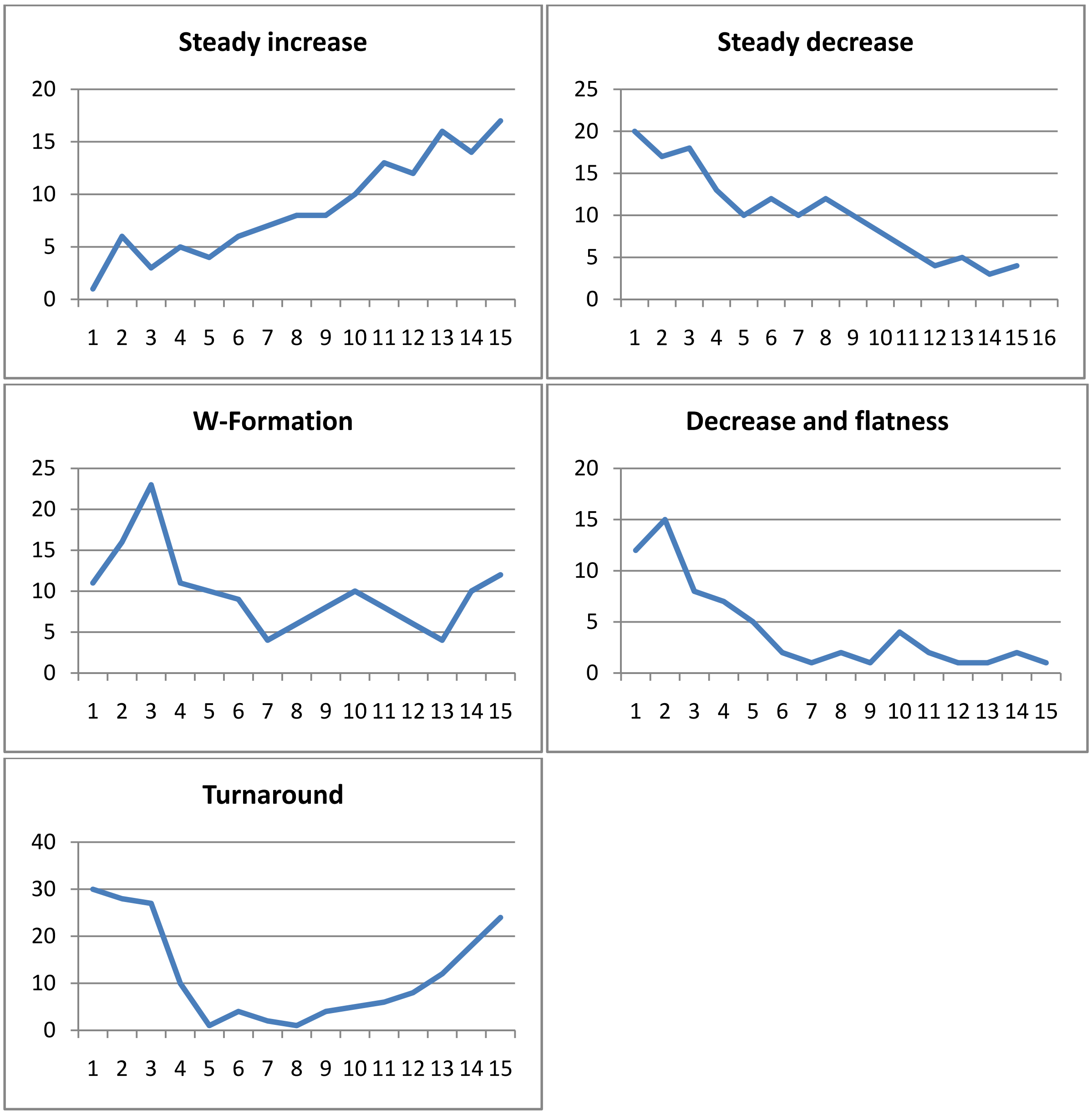


Figure 101: Sample search pattern for time-series evaluation

Scenario 1 – Subsequence pattern searching with varying MA periods

The first evaluation scenario was defined to produce reasonably precise results by varying the MA period smoothing. The algorithm is defined to perform complete subsequence searching, meaning that a match must neither start with the first data point nor end with the last data point. For the evaluation, the following parameters have been chosen for the algorithm:

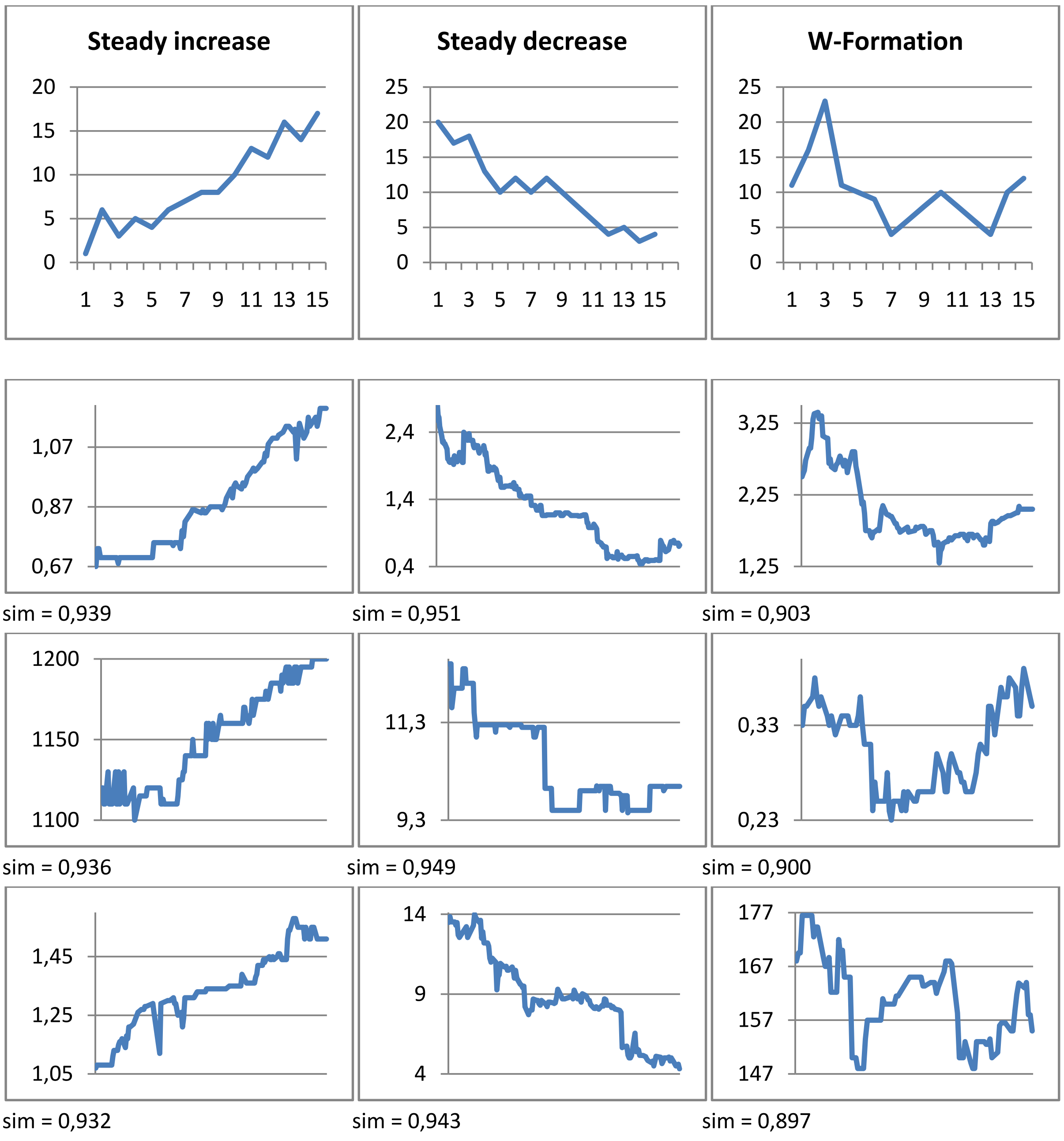
Field	Value
MinMAPeriodSearchedSequence	14
MaxMAPeriodSearchedSequence	30
MAStepSearchedSequence	2
AnchorStart	false
AnchorEnd	false
LocalityOfSlopeComparisons	Global

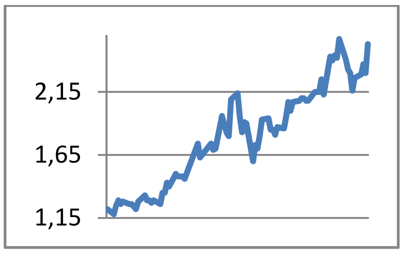
TurningPointMode	Extremum
WeightBySubsequenceLength	true

Table 11: Algorithm parameters for time-series evaluation scenario 1

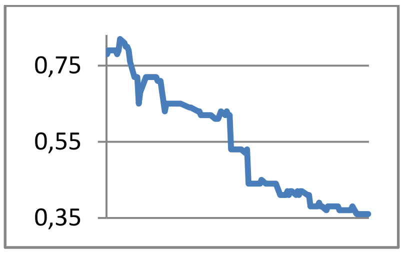
Search results

Figure 102 shows the best matches from the sample data set for each of the defined patterns. Below every match, the computed similarity score (sim) is listed. The plot shows that matches above a similarity score of 0,9 appear very similar and accurate. Below, deviations are already quite distinctive.

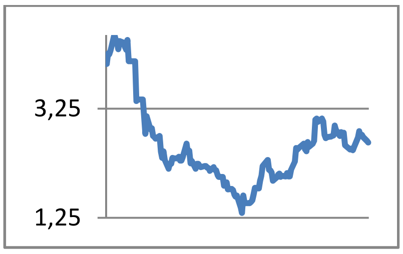




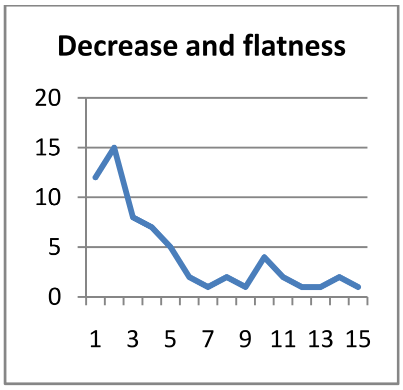
sim = 0,926

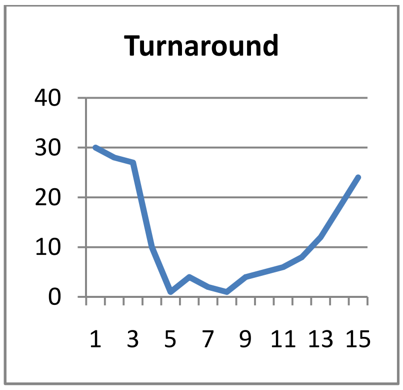


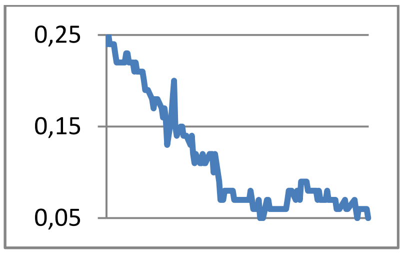
sim = 0,942



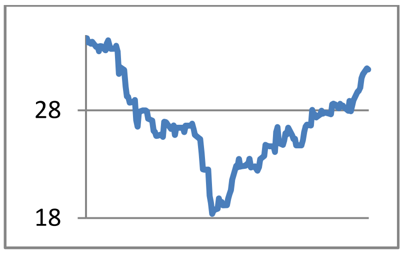
sim = 0,896



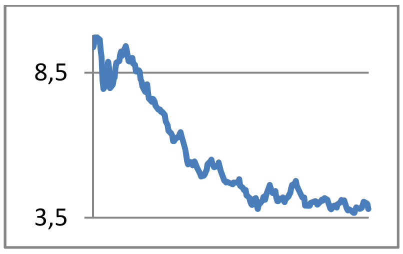




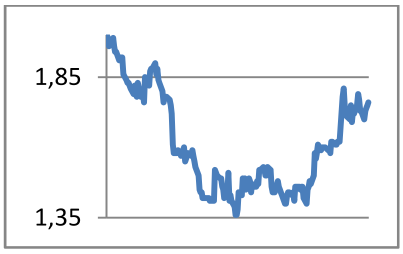
sim = 0,941



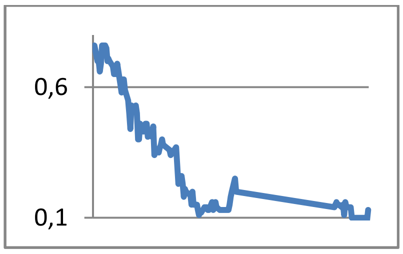
sim = 0,917



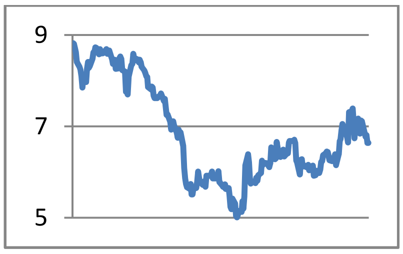
sim = 0,938



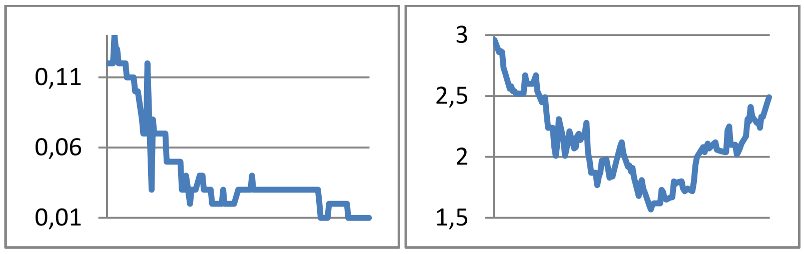
sim = 0,904



sim = 0,935



sim = 0,901



sim = 0,935 sim = 0,899

Figure 102: Search results for defined time series patterns

Performance

The execution speed of the search strongly depends on the following parameters:

- The number of data points in the searched sequence – The more data points the longer takes the MA smoothing and potentially more turning points emerge.
- The characteristics of the searched sequence – The more fluctuations and direction changes it has, the more turning points are extracted and the more slopes have to be compared.
- The minimum MA smoothing period – Especially for small periods the computation effort is large, as many turning points emerge from short-term movements.
- The MA step and the maximum MA smoothing period – The step length directly determines the number of iterations until the set maximum MA period is reached.

Thus, the performance is directly proportional to the number of turning points extracted, which depends on the MA period (the shorter the period, the more turning points), and the number of iterations with varying periods.

For the above presented search results and configuration, the following performance was measured:

Run	# Time series	# Parallel threads[15]	Avg DP per series	DP in pattern	Total time	Series/sec
1	2982	1	185	15	00:01:58	25,13
2	2982	5	185	15	00:01:28	33,85
3	2982	8	185	15	00:01:21	36,41
4	2982	15	185	15	00:01:30	32,81

Table 12: Performance results of time series pattern searching scenario 1

The result shows that the use of parallel threads speeds up the execution by up to 40%. With too many parallel threads, performance decreases again.

Scenario 2 – Subsequence pattern searching with anchored matches

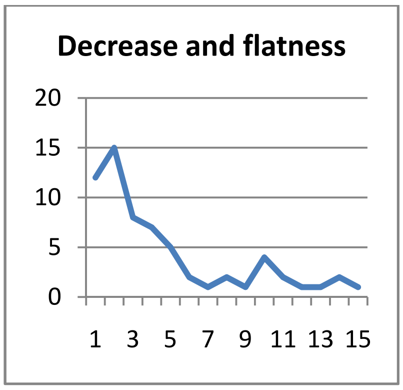
The second evaluation scenario is defined for higher evaluation speed as the MA period is not varied but only one pass is executed at a fixed MA period. In addition, the matches are anchored at the end, which also limits the algorithmic effort.

[15] For the implementation of muli-threaded tests, the SmartThreadPool was utilized. http://www.codeplex.com/smartthreadpool

For scenario 2, different parameters have been tried out and the performance and results have been compared for one selected search pattern.

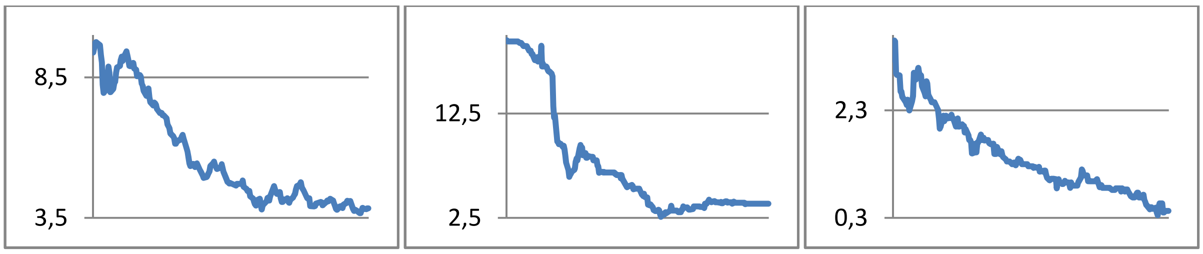
Search results

Figure 103 shows the three best matches for different configurations C1 to C3. The parameters of each configuration are given below.



C1 – Search Results

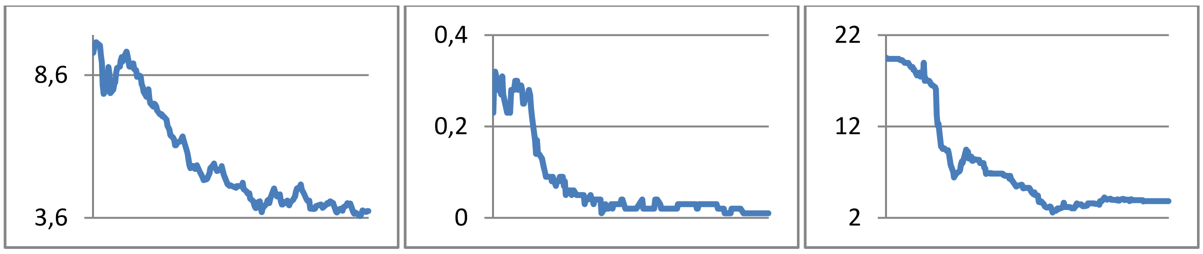
MA Period = 21, LocalityOfSlopeComparisons: Global, TurningPointMode: Extremum



sim = 0,939 sim = 0,926 sim = 0,919

C2 – Search Results

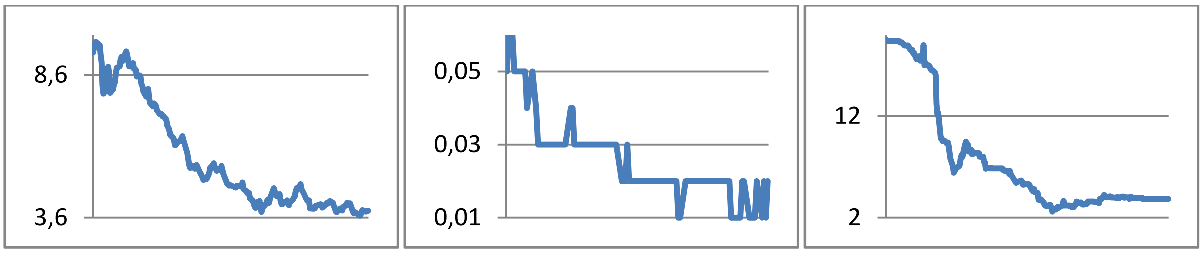
MA Period = 21, LocalityOfSlopeComparisons: Local, TurningPointMode: ExtremeValuesAverage



sim = 0,882 sim = 0,882 sim = 0,867

C3 – Search Results

MA Period = 21, LocalityOfSlopeComparisons: WeightedGlobal, TurningPointMode: AvgExtremumAndCrossingPoint



sim = 0,930 sim = 0,922 sim = 0,921

Figure 103: Search results for decrease and flatness pattern with different configurations

Subjectively, results for this pattern appear to be most accurate with configuration C2. In case of local slope comparison mode, the overall similarity score is lower for the same sequence. In all three cases, the best match was the same time-series. Another series was ranked second in C1 and third in both C2 and C3.

Performance

For the above presented search results and configuration, the following performance was measured:

Configuration	# Time series	# Parallel Threads	Avg DP per series	DP in pattern	Series/sec[16]
C1	2982	8	206	15	926,7
C2	2982	8	206	15	876,1
C3	2982	8	206	15	852,0

Table 13: Performance resulsts of time series pattern searching scenario 2

Evaluation results reference sequence searching

Scenario 3 – Reference sequence searching varying MA periods

In this evaluation scenario, the input for the search algorithm is not a search pattern with a couple of turning points, but a time-series taken from the original dataset. This means in particular that both the pattern sequence and the target sequence undergo the MA smoothing and turning point extraction process.

In scenario 3, a reference sequence is searched with the MA smoothing mode “VaryingPeriods”, thus each comparison runs through multiple passes of MA variations for the reference and target sequences. For the scenario, the following configuration parameters have been chosen:

Field	Value
MinMAPeriodSearchedSequence	20
MaxMAPeriodSearchedSequence	40
MAStepSearchedSequence	2
MinMAPeriodReferenceSequence	20
MaxMAPeriodReferenceSequence	40
MAStepSearchedSequence	2
AnchorStart	True
AnchorEnd	False
LocalityOfSlopeComparisons	Global
TurningPointMode	ExtremeValuesAverage
ExtremeValuesAveragePercentage	5
WeightBySubsequenceLength	true

Table 14: Algorithm parameters for time-series evaluation scenario 3

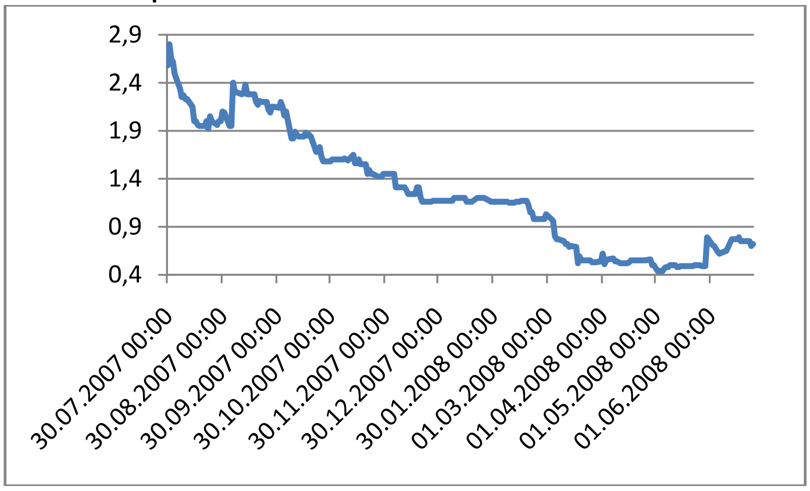
As can be seen from the table, the MA period is varied between 20 and 40 at a step length of 2 for both series. This means $\left(\frac{40-20}{2}\right)^2 = 100$ passes of slope comparisons for each sequence.

[16] As the execution speed strongly varied up to 150 series/sec, the average speed of 3 successive runs has been taken.

Search results

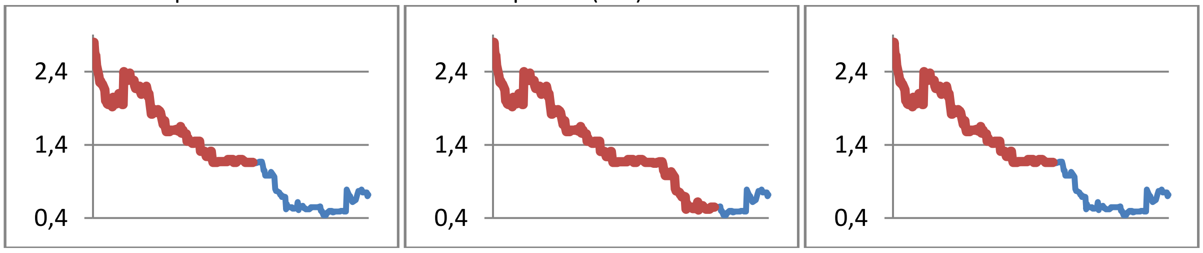
Figure 104 shows the best matches for a given search pattern with the presented configuration.

Reference sequence

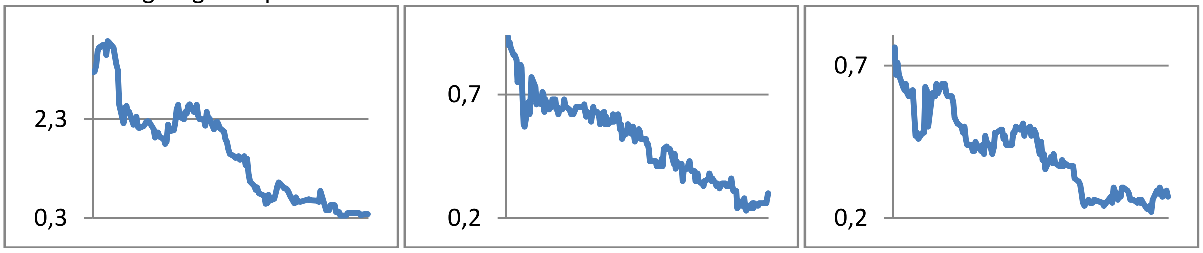


Matches

Matched subsequences from the reference sequence (red)



Best matching target sequences



sim = 0,979 sim = 0,978 sim = 0,975

Figure 104: Search results for reference sequence time-series evaluation scenario 3

Performance

For the above presented search results and configuration, the following performance was measured:

Run	# Time series	# Parallel Threads	Avg DP per series	# DP in ref.seq.	Series/ses
1	2981	8	206	236	11,3
2	2981	15	206	236	9,68

Table 15: Performance results of time series reference sequence searching scenario 3

Index of figures

Index of tables

Index of algorithms

Bibliography

[1] Agrawal R., Faloutsos C., Swami A.R.: Efficient Similarity Search in Sequence Databases. In FODO, pp.69-84, 1993.

[2] Agrawal R., Lin K., Sawhney H.S., Shim K.: Fast similarity search in the presence of noise, scaling, and translation in time-series databases. In Proc. 21st Int. Conf. on Very Large Data Bases (VLDB'95), Zurich, Switzerland, pp. 490–501, 1995.

[3] Altschul S.F., Gish W., Miller W., Myers E.W., Lipman D.J.: Basic local alignment search tool. J.Mol.Biol. 215:403-410, 1990.

[4] Berchtold S., Kriegel H.: S3: similarity search in CAD database systems. *SIGMOD Rec.* 26, 2 (Jun. 1997), 564-567, 1997.

[5] Brettlecker G., Milano D., Ranaldi P., Schenk H.J., Schuldt H., Springmann M.: ISIS and OSIRIS: A Process-Based Digital Library Application on Top of a Distributed Process Support Middleware. In Digital Libraries: Research and Development. LNCS, vol. 4877, pp.46-55, Springer, Heidelberg, 2007.

[6] Bueno R., Traina A.J., and Traina, C.: Genetic algorithms for approximate similarity queries. Data Knowl. Eng. 62, 3, 459-482, 2007.

[7] Bush R.R., Mosteller F.: A model for simulus generalization and discrimination. Psychological Review, 58, pp. 413-423, 1951.

[8] Ceglar A., Roddick, J.F.: Association Mining. *ACM Comput. Surv.* 38, 2 (Jul. 2006), 5, 2006.

[9] Chan K., Fu A.W.: Efficient Time Series Matching by Wavelets. In Proceedings of the 15th IEEE International Conference on Data Engineering, pp. 126-133, 1999.

[10] Chapman S.: SimMetrics. Open-source library of similarity metrics. Available from http://www.dcs.shef.ac.uk/~sam/simmetrics.html , last access 21/09/2008.

[11] Chen Z., Concepcion A., Metcalf A., Arokiya J., Bohannan L.: Smart Sequence Similarity Search (S4) System. In Applications of Fuzzy Sets Theory. LNCS, vol. 4578, pp.643-650, Springer, Heidelberg, 2007.

[12] Chu K, Wong M.: Fast Time-Series Searching with Scaling and Shifting. In Proceedings of the 18th ACM Symposium on Principles of Database Systems, pp. 237-248, 1999.

[13] Das G., Mannila H., Ronkainen P.: Similarity of attributes by external probes. Knowledge Discovery and Data Mining (KDD 1998), pp.23-29, 1998.

[14] Eisler H., Ekman G.: A mechanism of subjective similarity, Acta Psychologica, 16, pp. 1-10, 1959.

[15] Faloutsos C., Jagadish H., Mendelzon A., Milo, T.: A Signature Technique for Similarity-Based Queries. In *Proceedings of the Compression and Complexity of Sequences* (SEQUENCES 1997). Washington, DC, 1997.

[16] Faloutsos C., Ranganathan M., Mano Lopoulos Y.: Fast subsequence matching in time-series databases. In Proceedings of ACM SIGMOD International Conference on Management of Data, pp. 419-429, 1999.

[17] Floratos A., Rigoutsos I., Parida L., Gao Y.: DELPHI : A pattern-based method for detecting sequence similarity. IBM J. Research & Development, 45(3/4):455-474, May/July, 2001.

[18] Gati I., Tversky A.: Weighting common and distinctive features in perceptual and conceptual judgments. Cognitive Psychology, 16, pp. 341-370, 1984.

[19] Goldstone R.L., Son J.Y.: Similarity. In R.A. Wilson & F. C. Keil (eds.) MIT Encyclopedia of the Cognitive Sciences. Cambridge, MA: MIT Press.

[20] Gowser J. C.: A General Coefficient of Similarity and Some of Its Properties. In: *Biometrics*, 27(4). 857 - 871, 1971.

[21] Guttman A.: R-Trees: A dynamic index structure for spatial searching. In Proceedings of the ACM SIGMOD Conference, pp. 47-57, 1984.

[22] Holyoak K.J., Morrison R.G.: The Cambridge Handbook of Thinking and Reasoning. Cambridge University Press, 2005.

[23] Kahveei T., Singh A.: Variable length queries for time series data. In Proceedings of 17th International Conference on Data Engineering, pp. 273-282, 2001.

[24] Kanth K.V., Agrawal D., Singh A.: Dimensionality Reduction for Similarity Searching in Dynamic Databases. In Proceedings of the ACM SIGMOD Conference, pp. 166-167, 1998.

[25] Keogh E.J., Pazzani M.J.: A simple dimensionality Reduction Technique for Fast Similarity Search in Large Time Series Databases. In Proceedings o fthe 4th Pacific-Asia Conference on Knowledge Discovery and Data Mining PAKDD, pp.122-133, Kyoto, Japan, 2000.

[26] Korn F., Jagadish H., Faloutsos C.: Efficiently supporting ad hoc queries in large datasets of time sequences. In Proceedings of the ACM SIGMOD International Conference on Management of Data, pp. 289-300, 1997.

[27] Lin D.: An Information-Theoretic Definition of Similarity. In Proceedings of the Fifteenth international Conference on Machine Learning. J. W. Shavlik, Ed. Morgan Kaufmann Publishers, San Francisco, CA, pp.296-304, 1998.

[28] Liu J., Wong K.C., Hui K.K.: Discovering User Behavior Patterns in Personalized Interface Agents. In Proceedings of the Second international Conference on intelligent Data Engineering and Automated Learning, Data Mining, Financial Engineering, and intelligent Agents. K. Leung, L. Chan, L. Chan, H. Meng, and H. Meng, Eds. Lecture Notes In Computer Science, vol. 1983. Springer-Verlag, London, 398-403, 2000.

[29] Luckham D.: The Power of Events. Addison-Wesley, 2002.

[30] Lv Q.: Similarity Search for Large-Scale Image Datasets. Doctoral Thesis, Princeton University, 2006.

[31] Mannila H., Moen P.: Similarity between event types in sequences. In Proceedings of the First International Conference on Data Warehousing and Knowledge Discovery (DaWaK'99), pp. 271-280, Florence, Italy, 1999.

[32] Mannila H., Seppänen J.K.: Finding similar situations in sequences of events via random projections. In Proceedings of the First SIAM International Conference on Data Mining (SDM'01), Chicago, 2001.

[33] Markman A.B., Gentner D.: Structural alignment during similarity comparisons. Cognitive Psychology, 25, pp. 431-467, 1993.

[34] Moen P.: Attribute, Event Sequence, and Event Type Similarity Notions for Data Mining. PhD Thesis, Report A-2000-1, Department of Computer Science, University of Helsinki, 2000.

[35] Motro A.: VAGUE: A user interface to relational databases that permits vague queries. *ACM Trans. Inf. Syst.* 6, 3 (Jul. 1988), pp. 187-214. 1988.

[36] Negi T., Bansal V.: Time Series: Similarity Search and its Applications. In Proceedings of the International Conference on Systemics, Cybernetics and Informatics: ICSCI-04, Hyderabad, India, pp 528-533, 2005.

[37] Obweger H.: Similarity search in large-scale event spaces. Master thesis, Vienna University of Technology, 2008.

[38] Pratt K.: Locating Patterns in Discrete Time Series. Master thesis, Computer Science and Engineering, University of South Florida, 2001.

[39] Pu J., Lou K., Ramani K.: A 2D Sketch-Based User Interface for 3D CAD Model Retrieval. Computer-Aided Design & Applications, Vol. 2, No. 6, pp 717-725, 2005.

[40] Rafiei D.: On Similarity-Based Queries for Time Series Data. In *Proceedings of the 15th international Conference on Data Engineering* ICDE, pp.410-423, Washington, DC, 1999.

[41] Rozsnyai S., Schiefer J., Schatten A.: Concepts and Models for Typing Events for Event-Based Systems. In: *Proceedings of the 2007 inaugural international Conference on Distributed Event-Based Systems*. 62 – 70, 2007.

[42] Rozsnyai S., Vecera R., Schiefer J., Schatten A.: Event Cloud: Searching for Correlated Business events. In Proceedings of the 4th IEEE International Conference on Enterprise Computing, E-Commerce and E-Services (EEE 2007), pp.409-420, Tokyo, Japan, 2007.

[43] Schiefer J., Seufert A.: Management and Controlling of Time-Sensitive Business Processes with Sense & Respond. In: *Proceedings of the international Conference on Computational intelligence For Modelling, Control and Automation and international Conference on intelligent Agents, Web Technologies and internet Commerce 2005.* 77 – 82, 2005.

[44] Shepard R.N.: The analysis of proximities: Multidimensional scaling with an unknown distance function. Part I. Psychometrika, 27, pp. 125-140. 1962.

[45] Shepard R. N.: Toward a Universal Law of Generalization for Psychological Science. In: *Science*, 237. 1317 – 1323, 1987.

[46] Sjoberg L.: A cognitive theory of similarity. Goteborg Psychological Reports, 2:10, 1972.

[47] Struzik Z., Siebes A.: The haar wavelet transform in the time series similarity paradigm. In Proceedings of Conference on Principles of Data Mining and Knowledge Discovery, pp. 12-22, 1999.

[48] Suntinger M., Obweger H., Schiefer J., Groeller M.E.: The Event Tunnel: Interactive Visualization of Complex Event Streams for Business Process Pattern Analysis. In Proceedings of the IEEE Pacific Visualization Symposium (PacificVIS ’08), pp. 111-118, Kyoto, Japan, 2008.

[49] Suntinger M., Obweger H., Schiefer J., Groeller M.E.: Event Tunnel: Exploring Event-Driven Business Processes. In IEEE Computer Graphics and Applications 28, 5 (Sep. 2008), 46-55.

[50] Toshniwal D., Joshi R.C.: A new approach for similarity search in time series databases based on slopes. In Proceedings of the 4th IEEE International Conference on Intelligent Systems, Design and Applications, Budapest, Hungary, pp. 719- 724, 2004.

[51] Toshniwal D., Joshi R.C.: Similarity Search in Time Series Data Using Time Weighted Slopes, In Informatica International Journal of Computing and Informatics 29:1, pp. 79-88, 2005.

[52] Tsai T.-H., Lee S.-Y.: SimSearcher: A Local Similarity Search Engine for Biological Sequence Databases. In Proceedings of the IEEE Fifth International Symposium on Multimedia Software Engineering (ISMSE'03), pp.305-313, 2003.

[53] Tversky A.: Features of Similarity. Psychological Review, 84, pp. 327-352. 1977.

[54] Vlachos M., Hadjieleftheriou M., Gunopulos D., Keogh, E.: Indexing Multidimensional Time-Series. The VLDB Journal 15:1, pp. 1-20. 2006.

[55] Wattenberg M.: Sketching a Graph to Query a Time-Series Database. CHI 2001 Short Talk. Available from http://www.bewitched.com/projects/querysketch/wattenberg-chi2001.pdf, last access 19/09/2008.

[56] Weiss G.M., Hirsh H.: Learning to Predict Rare Events in Event Sequences. In Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining (KDD-98), pp.359-363, Menlo Park, CA, 1998.

[57] Wu Y., Agrawal D., El Abbadi A.: A comparison of DFT and DWT based similarity search in time series databases. In Proceedings of 9th ACM International Conference on Information and Knowledge Management, pp. 488-495, 2000.

[58] Yi B.K., Faloutsos C.: Fast time sequence indexing for arbitrary Lp norms, The VLDB Journal, pp. 385-394, 2000.

[59] Zhang T., Chen W., Hu M., Peng Q.: A Similarity Computing Algorithm for Proteins. In Proceedings of the Ninth international Conference on Computer Aided Design and Computer Graphics CAD-CG. Pp.168-172, Washington, DC, 2005.