FAKULTÄT FÜR !NFORMATIK

Similarity Searching in Complex Business Events and Sequences thereof

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Software Engineering & Internet Computing

eingereicht von

Hannes Obweger
Matrikelnummer 0425962

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)

Eidesstattliche Erklärung

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 23.03.2009 Hannes Obweger, Bsc.

Abstract

This thesis contributes to the field of complex event-data analysis novel and formally well-founded methods for similarity searching, both on the level of single events and on the level of sequences of events. As event-based systems may produce highly diverse data sets, the main focus of our considerations is on highest possible flexibility. Also, the approaches shall be intelligible to business analysts and, of course, generate meaningful and intuitive results. Finally, the approaches shall be conceptually independent from concrete Complex Event Processing solutions and instead build upon abstract and generally accepted definitions of events, event types, etc.

Our approach on single-event similarity builds upon geometric ideas of similarity, with event attribute values defining the relative positioning of two events in an *n*-dimensional space. Thereby, the similarity between two events is calculated from weighted attribute-level similarities.

The proposed approach on event-sequence similarity outperforms existing approaches by allowing analysts to consider event-level similarities, order, and relative and absolute temporal structures in a highly flexible manner. It builds upon an assignment-based understanding of sequence similarity, where each unit of the pattern sequence is considered either represented by a certain event of the target sequence or missing therein. Our algorithm finds the best-possible assignment of the target sequence using a Branch & Bound strategy. This assignment is then used for calculating the similarity between the given sequences.

We conclude this work with a practical evaluation, where we apply the approach on event-sequence similarity in real-world scenarios from three application domains. We figured out that the algorithm performs excellent for short and sharp-edged sequences where a majority of events constitute clear and significant characteristics of the event sequence.

General remarks

The on-hand thesis results from a long-term research project on event-based similarity searching and serves as a basis for further extensions and improvements as presented by Martin Suntinger in his thesis, “Event-Based Similarity Search and its Application in Business Analytics”.

Content as presented in the following section was elaborated in collaboration with him and can be found in a more or less equivalent form in his thesis [49]:

- Section 1: Introduction
- Section 7: Application and results

Content

1 Introduction

1.1 Complex Event Processing

Event-based systems and particularly the concept of *Complex Event Processing* (CEP) [27] 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.

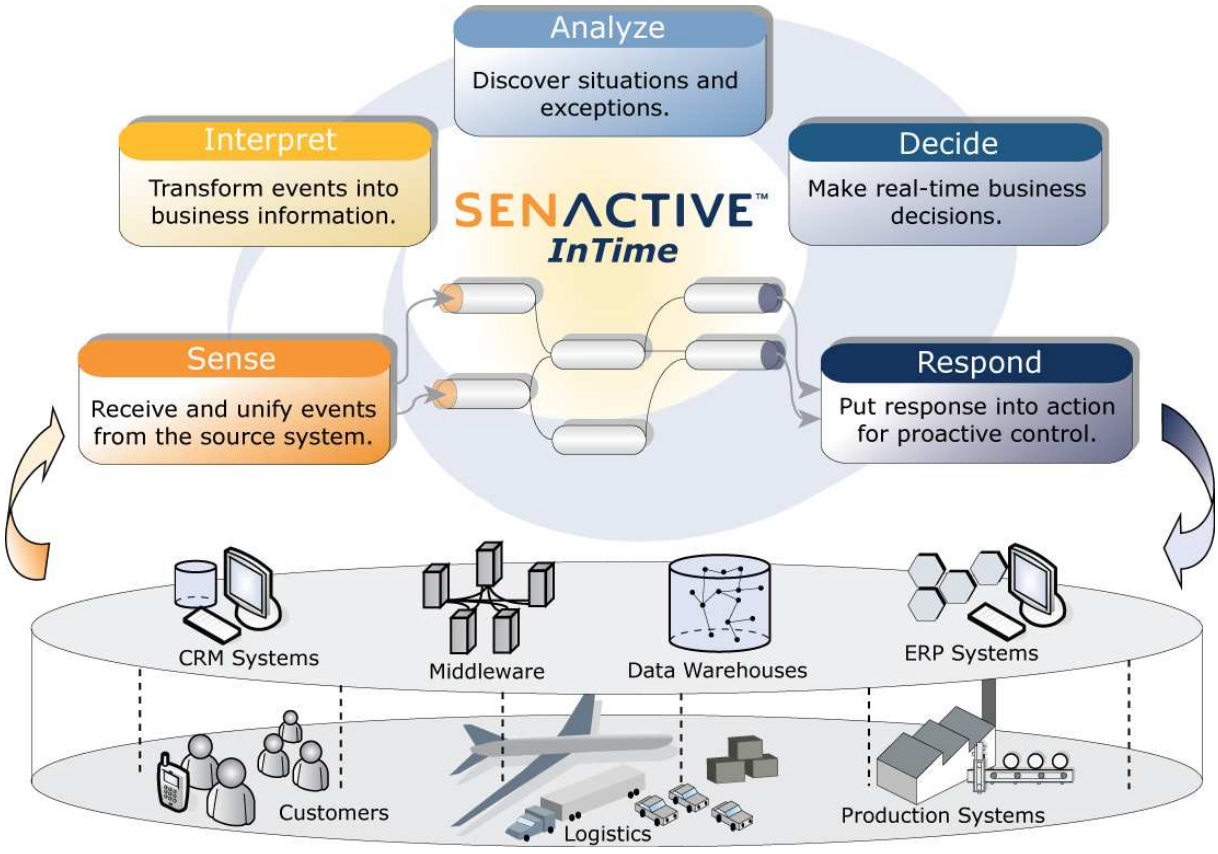


Figure 1: The sense and response model[1]

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 five 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. [46]

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.

1.2 Similarity searching in historic event data

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 or a whole sequence of events on hand, which other entities are similar to this? 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.

1.3 Objectives

This thesis contributes to the field of Event Data Warehousing [42] (EDWH)/Complex Event Data Analysis comprehensive and formally well-founded approaches on similarity searching in historic event data, both on the level of

- **single events,** and
- **sequences of events.**

For the purpose of this thesis, there is no restriction on a certain technique or a certain understanding of similarity that should underlie our solutions. Yet, the approaches shall be intelligible to business analysts, and, of course, generate meaningful and somewhat natural results. Also, the approaches shall be conceptually independent from a concrete CEP solution, and instead build upon abstract and generally accepted definitions of complex events, complex-event sequences, etc. The concrete prototype implementation, however, shall build upon *SENACTIVE's InTime* [46], one of the most promising CEP frameworks at the time of writing. Finally, as always for similarity searching and search algorithms in general, computational complexity and performance are critical issues. Yet, as there is little research done on this topic and the character of the on-hand thesis is a largely explorative one, we do not claim optimal performance but instead seek to define a solid groundwork for further optimizations.

In the following, let us define three basic, direction-giving requirements for similarity searching in the context of CEP. Applied to the on-hand thesis, these requirements are relevant both for single-event- and event-sequence similarity. Further, topic-specific requirements will be listed in the corresponding sections below.

1.3.1 A generic approach on event similarity

Over the past few years, complex event processing has gained more and more importance in several business domains. Two such domains do not necessarily have much in common: Complex event processing was proven to be useful in such different business domains as fraud-detection [50][51] and medical care [39]. It is easy to see that event-based systems may produce highly diverse data sets. Hence, an approach towards event-similarity that is intended to extend a generic complex event processing framework must not only fulfill the requirements for one specific domain (and likely fail in all others); it must instead be generic and flexible enough for any possible set of business events.

It is essential to understand that in the on-hand thesis, we do not aim for a single, all-purpose event-similarity measure. Yet, we aim to develop a consistent and coherent strategy that allows defining event-similarity measures among all kinds of complex business events and domain-specific problems.

1.3.2 An interest-driven approach

Today's real-world business processes are complex entities that execute in socio-economic systems of even higher complexity. When analyzing such business processes, the analyst always does so with a certain (possibly vague and imprecise) question in mind, with a certain focus. Some factors and characteristics are emphasized, while others are considered to be less important or even completely ignored. In complex event processing, an event represents a state change or action that occurs in a business environment. Consequently, when analyzing such events and sequences thereof, it always depends on the analyst certain interest whether two events are assessed to be similar and or not. It is easy to see that there is no single similarity measure for all purposes. Instead, similarity measures must be designed *per interest,* per *"focus of analysis"*.

What we call an "interest-driven approach" on event-similarity supports the detailed and target-oriented analysis of business events. Especially for complex scenarios that comprise several relevant aspects, it may come along with quite a large number of similarity measures, though. In order to achieve optimal efficiency, a framework is required that allows managing event-similarity measures in a quick and straightforward fashion.

1.3.3 Leaving control up to the expert

In the previous section, we have pointed out that a similarity measure for complex events is intrinsically tied to a certain interest, to a certain point of view. At this point, one might think of a magic black box that takes the analyst's interest as input, makes some chattering noises and creates a perfect-fitting similarity measure. Life is not a walk in the park, though.

We claim that in order to achieve efficient and powerful event-similarity searching, defining a similarity measure must actively involve the domain expert. Hereby, the domain expert must be granted extensive control over all relevant aspects of the similarity measure. Such degree of involvement may, of course, be costly. Yet, we think that in real-world scenarios, only the domain expert himself has sufficient knowledge about the data's possibly complex semantics and its unique characteristics. Furthermore, we think that only if the business analyst knows how a similarity measure works, e.g., by configuring it "by hand", he or she can adequately interpret the measure's results in their certain context, and adapt the similarity measure if necessary.

We have stated that leaving control up to the domain expert allows target-oriented similarity searching even in complex event data. This may, indeed, be costly. Therefore, a toolkit is required that assists the domain expert where possible. First of all, it must allow the quick and simple definition of a similarity measure.

1.4 Overview

The on-hand thesis is structured as follows: In section 2, we will discuss related work and give an overview about the current state of research. In section 3, we will clarify basic terms and define those concepts that form the base for later considerations. The technological background that underlies our ideas will be presented in section 4. In section 5, we will present the first key part of this thesis, a distance-based approach on single-event similarity. The second key part of this thesis, a new, assignment-based approach on event-sequence similarity, will be presented in section 6. Eventually, we will evaluate our approaches by applying them in real world-scenarios in section 7. The thesis is concluded by section 8, where we will summarize the most relevant parts of our work and also present future research topics.

2 Related Work

This section discusses related work and gives an overview about the current state of research. The subsections correspond to single aspects of the on-hand thesis and discuss the according contributions in detail.

2.1 Similarity between single events

Similarity between single, multivariate data objects has attracted much attention in the database domain. The approaches proposed are of certain interest for our work, since records in a database can be considered being somehow equivalent to complex events. As a record is of contained in a table of a certain schema and composed from a number of primitive fields, a complex event is of a certain type and provides a number of attributes.

Nearest neighbor (NN) searching, or, more generally, *top-k searching* is a heavily discussed feature. A top-*k* search returns the *k* objects that are nearest, i.e., in a certain sense most similar, to a given query object. Numerous contributions discuss the efficient handling of such queries in relational database management systems (RDBMS) [8][13][10]. Since our solution builds upon an existing framework that does not support top-k searches, we must compare runtime-objects efficiently.

Several database maintenance tasks require *record linkage* [34][43][20], or *record matching*. Record linkage is the task of detecting records that refer to one and the same real-world entity but differ in one or more fields. One usually assumes two records to be duplicates in the sense of semantic equality if the measured similarity between them is beyond a certain, domain-dependent threshold. *Data cleansing* [24] targets the elimination of such duplicates. The *merge/purge problem* [18] aims at merging the possibly incomplete and inconsistent duplicate records yielding one single representation that is as complete and correct as possible.

2.2 Similarity between sequential data

The ability to capture, process and maintain sequential data has been crucial for several business domains for decades. Recently, the focus has shifted towards gaining knowledge from accumulated sets of sequential data. Applications come, for instance, from stock market analysis, the medical domain, or voice recognition. Most of the work that contributes to searching sequential databases deals with exact searches, i.e., searches that return sequences that are certainly equal to a given query. Yet, several contributions emphasized the demand for vague queries that focus on finding sequences similar to a given query rather than being certainly equal [4] [37].

Note that the majority of contributions on sequence similarity treat time-series, i.e., sequences of numeric values, either single-dimensional [7][52] or multi-dimensional [55]. Our contribution deals with similarity search in sequences of complex events. Nevertheless, we adopt certain ideas and parts of the domain-specific terminology.

2.3 Similarity between event sequences

Recently, notable attention has been paid on analyzing sequences of events. Though, one must differentiate between the various understandings of event sequences that underlie the proposed solutions.

In the following sections, we discuss the possible models of event sequences, ranging from rather simple to complex. The solution proposed in the on-hand thesis is designed to operate on the most complex model. Though, we decided to discuss existing methods for event-sequence *similarity-searching* that are suitable for simpler models. These may not be directly adaptable to work on complex event-sequences, but may reveal commonalities or alternative - and possibly opposed - viewpoints.

In addition to traditional similarity measures, we pay some attention on the various kinds of *pattern matching*. Usually, pattern matching is strict: A sequence of entities matches a certain pattern, or it does not – there's nothing in between. Yet, pattern matching can be applied for a certain, admittedly restricted kind of similarity-search. From a given event sequence, one could evolve a slightly generalized pattern. The resulting set of matches is then similar to the original sequence – in a certain, previously defined sense. The basic approach proposed in the on-hand thesis does not apply pattern matching or underlying concepts.

2.3.1 Simple events in a certain order

In simple models, an event is of a certain event type but does not contain any additional information concerning the corresponding action or state change and its context. Further on, event sequences have a defined order, while the exact occurrence time of an event and the elapsed time span between two consecutive events are omitted. Therefore, a so-defined event sequence corresponds to an ordered collection of discrete values. Applications that build upon the described model come, for instance, from sociology [1][36].

The described event model is somehow simplistic. Though, the analysis of so-defined event sequences allows utilizing well-discussed algorithms and data structures from the sequence-analysis domain. *String comparison* deals with measuring the similarity between sequences of characters [16]. In the bio-informatics domain, several tools exist that calculate the similarity between DNA or RNA sequences [2]. *Regular expressions* can be used to detect event sequences that match a certain pattern [18][36]. *Weighted regular expressions* [6] assign costs to each pattern element and may guide the way to flexible and configurable similarity searching in event sequences at the described level of abstraction. Real-world applications, however, have not yet been described.

2.3.2 Complex events in a certain order

Some domains require complex events that adequately describe the underlying action or state change, while the exact temporal relationships between events can be omitted. In such cases, an event sequence corresponds to an ordered collection of complex data objects. Applications that typically build upon this model come from the sales domain [5].

Lots of attention has been paid on detecting frequent patterns in so-defined event sequences [5][56]. In the data mining discipline, one could be interested in analyzing basket data to detect certain chains of product purchases that appeared frequently. The evolving knowledge could then be used to implement automatic recommendation systems that do not only provide related products but also take the temporal order of other customers' purchases into account.

Note that certain equality between so-defined event sequences can be extremely rare. Nevertheless, an adequate similarity model which is applicable in practice has not yet been published. Complex events often represent a set of certain entities. Consequently, a relation between event sequences can be defined upon subset relations between the constituting events.

2.3.3 Simple events at certain time stamps

In some domains, analysts rather focus on the temporal relationships between certain types of events than on the detailed characteristics of the underlying actions or state changes. Therefore, it is sufficient to use simple events, i.e., events that are of a certain event type but do not contain additional information. In the described model, event sequences correspond to time-stamped sequences of discrete values. Applications that build upon this model come, for instance, from network traffic analysis [34].

Mannila and Ronkainen [31] show an exact approach that calculates the similarity between two so-defined event sequences by computing the minimal edit distance. The edit distance is a similarity measure well known in string comparison and reflects the work that is required to transfer one sequence into another. The proposed measure adapts the *move*-operations' cost-function in order to reflect the time span an event was shifted by in time. Continuing work [34] addresses the rather tricky topic of assigning costs to the various edit operations and applies the proposed similarity model on telecommunication alarms and WWW page requests. We adopt some of the basic ideas, but address subsequence searching as well as similarity at the level of single events.

Mannila and Seppänen [33] follow an approximate approach by generating *k*-dimensional vectors from event sequences that serve as "fingerprints". These fingerprints are calculated from *k*-dimensional random vectors that are associated with the occurring event types. Event sequences with similar fingerprints are likely to have a quite low edit distance. The proposed approach does not attain similarity on the level of events and is again restricted in comparing event sequences of different length. Hence, it allows the high-performing pre-selection of potential similarity matches under certain conditions.

A rudimental kind of pattern matching for so-defined event sequences was proposed by Mannila et al. [31]. The approach is restricted to process temporally short patterns, so called episodes. Asarin et al. [5] introduced *timed regular expressions* that extend the regular expressions' original expressiveness and allow dealing with time spans between consecutive entities. Presently, applications in analyzing real-world event sequences have not yet been described.

2.3.4 Complex events at certain time stamps

Tightening competition in various business domains has lead to complex event models that allow gaining knowledge from both, complex and expressive events and the temporal relations between them. In such models, an event sequence corresponds to a time-stamped sequence of complex data objects.

Little research has been done on analyzing so-defined event sequences. Mannila and Toivonen [30] extended the rudimental pattern matching approach proposed by Mannila et al. [31] to allow restrictions on event attributes. Again, patterns are restricted to a small set of events and relatively short time spans. Similarity measures, by the way, have not been proposed at all.

In the on-hand thesis, we introduce a similarity measure for complex event sequences as described above. The measure considers both, similarity at the level of single events as well as temporal relationships between them. The proposed algorithm is highly configurable and can easily be adapted to various business domains. Furthermore, it allows processing event sequences at a lower level of complexity, such as described in 2.3.1, 2.3.2, and 2.3.3. Specialized algorithms are assumed to perform much better at these levels, though.

3 Terms, definitions and notations

In section 2, we have introduced the concepts of *similarity* and *events* and *event sequences* in a pragmatic and fairly informal fashion. The goal of this section is to fully clarify those concepts, to define according notations that will be used in the remainder of this thesis, and also to present our idea of illustrating event types, events, and sequences thereof.

3.1 Similarity

The concept of similarity, describing the perceived degree of *resemblance* of an entity a to an entity b, is somewhat inherent to the human mind and should be familiar to readers: More or less implicitly, one assessed the similarity between faces, ideas, pop songs, etc., in everyday's life. Musing on the meaning of similarity in psychology and philosophy, however, is far outside the scope of the on-hand thesis (and, by the way, even further outside the "scope" of the author); let us end this odyssey with the words of James' *Principles of Psychology* [22]: "This sense of Sameness is the very keel and backbone of our thinking".

What exactly makes an entity similar to another one solely depends on the certain context, and can, of course, not be defined in a general manner. In the following, let us therefore introduce those terms and concepts that allow defining and characterizing the (arithmetic and highly context-specific) measurements of similarity in mathematics and computer science.

3.1.1 Similarity measures

In computer science, much effort has been spent on calculating meaningful and somewhat "natural" similarities between however-defined entities. Functions and algorithms that perform such calculation are usually referred to as *similarity-measures*, often regardless of the function's co-domain, and other properties. In the on-hand thesis, we consider a similarity measure "in the broad sense"[2] a function or algorithm that returns values between 0 and 1, with a higher result indicating greater similarity. Formally, it can thus be defined as follows:

Definition: Given a set of entities A, we refer to a function $sim: A \times A \to \mathbb{R}$ as a *similarity-measure (in the broad sense)* for A if for all $a, b \in A$

1. $0 \leq sim(a, b) \leq 1$
2. $sim(a, b) = 1$ if $a = b$

[3]

Note that for a similarity measure in the broad sense, we explicitly do not require the property of *symmetry*: Given a set of entities A, two entities $a, b \in A$ and a similarity measure $sim: A \times A \to [0,1]$, $sim(a, b)$ (the similarity "of b to a") is not necessarily equal to $sim(b, a)$ (the similarity "of a to b"). Therefore, to avoid ambiguities, we will speak of similarities *of target entities to pattern entities* in the following.

[2] We will define the term similarity-measure in a strictly mathematical sense in section 3.1.3.

[3] Due to its boundedness, we will consider the interval $[0,1]$ the co-domain for so-defined similarity measures, i.e., given a set of entities A, we will write $sim: A \times A \to [0,1]$ instead of $sim: A \times A \to \mathbb{R}$. We will behave accordingly for all later definitions of similarity-measures and distance-functions.

3.1.2 Distance and similarity

A concept that is strongly related to the idea of similarity (and sometimes used more or less synonymously) is that of *distance*, or, equivalently, of *costs*.[4] So-called geometric similarity-models base upon the idea that a (possibly conceptual) distance between two entities is inversely related to the perceived similarity between them: The larger the distance between two entities, the smaller the similarity, and vice versa. Thus, in accordance with similarity-measures "in the broad sense", let us define distance-functions "in the broad sense" as follows:

Definition: Given a set of entities A, we refer to a function $d: A \times A \to \mathbb{R}$ as a *distance-function (in the broad sense)* for A if for all $a, b \in A$

1. $d(a,b) \geq 0$
2. $d(a,b) = 0$ if $a = b$

Working with distances instead of similarities has notable advantages: Often, distance-function arise more or less directly from the data's certain characteristics; consider, for instance, the geometric distance between cities, or the absolute difference between two numbers. Also, calculations may be easier to perform on distances rather than on similarities. Similarities, on the other hand, are bound to a $[0,1]$-interval and may thus be easier to comprehend. A statement such as "a resembles b to a degree of 64%", for instance, might be more useful than "The distance between a and b is 1492" in many cases.

It is easy to see that for using the advantages of both concepts, transformations between distances and similarities are indispensable. Generally, this can be done in a variety of ways. In the on-hand thesis, we will follow Shepard [46], who proposed an exponential relation:

Given a set of entities A and a similarity-measure $sim: A \times A \to [0,1]$, a corresponding distance function $d: A \times A \to \mathbb{R}_0^+$ is defined as follows:

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

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

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

3.1.3 Metrics, pseudo-metrics and similarity-measures in the strict sense

The above definitions of similarity-measures and distance-functions are natural and should be generally acceptable among all understandings of similarity.[5] In a strictly mathematical sense, however, more rigorous

[4] When using the term *costs* instead of *distance*, one might understand them the costs of reaching b from a, or, alternatively, the costs of transforming b into a.

[5] Depending on the certain similarity model, very different requirements exist on similarity-measures. A brief introduction on the most relevant similarity models is given by Suntinger [49].

criteria exist on distance-functions, and, following from that, on similarity-measures. These additional criteria are valuable when calculating similarities between larger sets of entities.[6]

The most rigorous conditions exist on *metrics* (i.e., on distance-functions in the strict sense):[7]

Definition: Given a set of entities A, we refer to a function $d: A \times A \rightarrow \mathbb{R}$ as a *metric* for A if for all $a, b, c \in A$

1. $d(a,b) \geq 0$
2. $d(a,b) = 0$ iff $a = b$
3. $d(a,b) = d(b,a)$
4. $d(a,c) \leq d(a,b) + d(b,a)$

In many cases, condition 2 (i.e., returning a distance of 0 for identical entities only) is too restrictive. We refer to a distance-function that fulfils conditions 1, 3, and 4 and a less restrictive version of condition 2, 2', as *pseudometric*:

Definition: Given a set of entities A, we refer to a function $d: A \times A \rightarrow \mathbb{R}$ as a *pseudometric* for A if for all $a, b, c \in A$

1. $d(a,b) \geq 0$
2'. $d(a,b) = 0$ if $a = b$
3. $d(a,b) = d(b,a)$
4. $d(a,c) \leq d(a,b) + d(b,a)$

Following from the above definitions, let us now define *(strong) similarity-measures in the strict sense* and *weak similarity-measures in the strict sense*:

Definition: Given a set of entities A, we refer to a function $sim: A \times A \rightarrow \mathbb{R}$ as a *(strong) similarity measure in the strict sense* for A if for all $a, b, c \in A$

1. $0 \leq sim(a,b) \leq 1$
2. $sim(a,b) = 1$ iff $a = b$
3. $sim(a,b) = sim(b,a)$
4. if a and b and b and c are similar then a and c are also similar

Definition: Given a set of entities A, we refer to a function $sim: A \times A \rightarrow \mathbb{R}$ as a *weak similarity measure in the strict sense* for A if for all $a, b, c \in A$

1. $0 \leq sim(a,b) \leq 1$
2'. $sim(a,b) = 1$ if $a = b$
3. $sim(a,b) = sim(b,a)$
4. if a and b and b and c are similar then a and c are also similar

[6] Consider, for instance, a distance function $d: A \times A \rightarrow \mathbb{R}_0^+$ fulfilling the triangle-inequality. Here, when searching for all pairs of entities that are close to each other, calculating a distance between two entities a and c, $a, c \in A$, can be omitted if both $d(a,b)$, $b \in A$, and $d(b,c)$ are great.

[7] *Metric* and *pseudometric* are fundamental terms in mathematics. In their notation, however, the according definitions follow Moen [35].

3.1.4 Certain equality

We have stated that with both *similarity-measures in the broad sense* and *weak similarity-measures in the strict sense*, pairs of *non-equal* entities may result in a similarity of 1. Yet, it is essential to differentiate between "traditional" equality and non-equality ("$a = b$", "$a \neq b$"), and *absolute similarity* (i.e., a similarity of 1) and *absolute dissimilarity* (i.e., a similarity of 0). We therefore opted for a separate terminology, defined following Monge and Elkan [36]:

Definition: Given a similarity-measure $sim: A \times A \rightarrow [0,1]$, we refer to two entities a and b, $a, b \in A$, as *certainly equal* with respect to sim if $sim(a, b) = 1$. We refer to a and b as *certainly unequal* with respect to sim if $sim(a, b) = 0$.

3.2 Events

The term "event" is heavily used in computer science and has more or less different meanings across the various domains and contexts it is appears in. In this section, we seek to give a brief introduction on events in the context of Complex Event Processing (CEP); in the literature, such events are often referred to as *complex events*.

A commonly used definition of events in the context of CEP comes from Luckham's standard work on CEP, *The Power of Events* [27]:

> *An event is an object that is a record of an activity in a system. The event signified the activity. An event may be related to other events. [27]*

From that rather general starting point, let us go into more detail on the certain characteristics of complex events: Rozsnyai [40], listing a good many definitions from various domains, derives the following definitions:

> *Definition 1: Events are defined as observable actions or relevant state changes that can be absorbed by IT systems.*
>
> *Definition 2: Events can be decomposed to several causally related events. Several events can be aggregated to high level events.*
>
> *Definition 3: Events mark a specific point in time or in an aggregated form the timespan of an activity. [40]*

As aggregated events (composed events, high-level events) are out of the scope of the on-hand thesis, we skip Definition 2 (and narrow Definition 3) of Rozsnyai's congregation, and instead assume atomicity as suggested by Zimmer and Unland [57], among others.

Definition: Events…

- are defined as observable actions or relevant state changes that can be absorbed by IT systems,
- are atomic, i.e., they cannot be further dismantled and happens completely or not at all, and
- mark a specific point in time.

Thus, from a computation point of view, we consider an event an immutable entity that describes a state-change or action, together with the context that state-change or action occurs in, through publicly available, named and strongly-typed properties - the so-called *event attributes*. The certain set of event-attributes that are available for a certain event, of course, fully depends on the context, on the certain *meaning* of that event; hence, the only event-attribute that must be available across *all* events is the time of occurrence.

It is easy to see that the described understanding of events is in general accordance with the concept of objects in modern object-oriented programming languages such as C# or Java: An event can be considered an object meeting according constraints. From a computation point of view, we therefore build upon the following understanding of events:

> **Definition:** An event is an immutable object, providing its time of occurrence and other, context-dependent information about the represented state-change/action and the context that occurs in through - and only through - publicly available fields. We refer to the fields of an event as *event-attributes.*

As the time of occurrence must be available for each so-defined event, we can define the *time span* between two events regardless of the events' actual form.

> **Definition:** Let $\mathbb{E}$ denote the set of events, and let a function $v_t: \mathbb{E} \to \mathbb{R}_+$ address the time of occurrence of an event. Given two events *e* and f, we refer to the result of a function $t: \mathbb{E} \times \mathbb{E} \to \mathbb{R},\ t(e,f) = v_t(f) - v_t(e)$, as the *(absolute) time span* between *e* and f.

We will use the above notations, i.e., $v_t: \mathbb{E} \to \mathbb{R}_+$ for addressing the time of occurrence of an event and $\mathbb{E}$ for addressing the set of all so-defined events, throughout the on-hand thesis. More generally, we will address the value of an attribute a of type $\mathbb{A}$ in an event *e* through a function $v_a: \mathbb{E} \to \mathbb{A}$, i.e., through $v_a(e)$.

3.2.1 Event types

Event types define the structure, i.e., the set of event attributes, of a certain *class* of events, both in its syntax and (more or less explicitly) in its semantics. As each event must be an *instance* of a certain event type, event types relate to the concept of *classes* in OOP as *events* relate to *objects*.

> **Definition:** An event type defines the structure of a class of events. The structure of an event is represented by a collection of event attributes. Each event is an instance of exactly one event type.

Formally, we understand an event type T a set of tuples $T = \{(a_1, \mathbb{A}_1), (a_2, \mathbb{A}_2), \dots, (a_k, \mathbb{A}_k)\}$, with a_i being the name and $\mathbb{A}_i$ being the domain of the i^{th} event attribute defined in T. With $\mathbb{T}$ denoting the set of all so-defined event types, we address the event type of an event *e* through a function $typeOf: \mathbb{E} \to \mathbb{T}$, i.e., through $typeOf(e)$.

Event models may also implement the concept of *subtyping* for event types. As in this respect, event types can be considered equivalent to classes, we will not go into further detail here. Yet, if event types T and S are in a (however-specified) subtype-relationship, with T being the subtype of S and S being the supertype of T, we write $T <: S$.

3.2.2 Illustration

Throughout the following chapters, we will depict both events and event types as rectangular nodes, horizontally separated into a header and a body area.

For events, the header shows a (locally unique) label that allows addressing the represented event throughout the certain context, as well as its type's name in squared brackets. The body lists the according event attributes' labels, together with their values. For event types, the header shows the name of the represented event type. Here, the body lists the defined event attributes' labels, together with their data types in squared brackets. Figure 2 below shows exemplary illustrations an event e and its event type E.

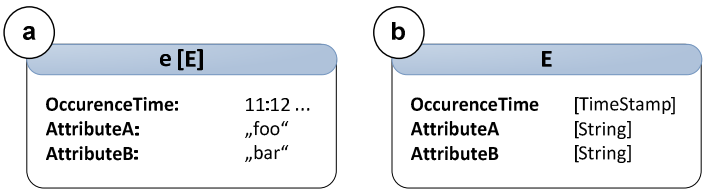


Figure 2: Illustrating events and event types

In both cases, equally colored headers indicate equal event types. Also, event attributes that are irrelevant in the given context may be omitted.

3.3 Event sequences

We have stated that an event represents a single action or state-change that occurs in a business environment. Complex happenings in such business environment, such as business-process instances, etc., thus result in amounts of related, successive events. We refer to such collections of events as *event sequences:*

Definition: An event sequence S is a list of events ordered by their times of occurrence, i.e., $S = (e_1, e_2, \dots, e_n)$, where $\mathrm{v}_t(e_i) < \mathrm{v}_t(e_{i+1})\ \forall i = 1, \dots, n-1$.[8]

Definition: Given an event sequence S and an event $e \in S$, we refer to the result of a function $pos(e, S) = |\{f | f \in S, v_t(f) \leq v_t(e)\}|$, i.e., the number of events in S with a time of occurrence smaller than or equal to e, as the *position* of e in S. Hence, we refer to an event $e \in S$ with $pos(e, S) = i$ as the i^{th} event in S.

Definition: Given an event sequence $S = (e_1, e_2, \dots, e_n)$, we refer to a sequence $S' = (e_k, e_{k+1}, \dots, e_l)$ with $k \geq 1$, $l \leq n$ and $k < l$ as a *sub-sequence* of S, $S' \subseteq S$.

In various contexts, one might be interested in the "extent" of a certain event-sequence. We distinguish between the *size* and the *length* of a solution:

[8] We assume unique time stamps for the sake of simplicity, in order to ensure a defined temporal order of events. In real-world scenarios, however, several events may occur in one and the same time stamp: Here, one can establish a unique order by taking another unique attribute, e.g., a unique ID, into account.

Definition: Given an event sequence S, we refer to the number of events in S as the *size* of S. We address the size of an event sequence S through $|S|$.

Definition: Given an event sequence S with e_i addressing the i^{th} event in S, we refer to the result of a function $l(S) = v_t(e_{|S|}) - v_t(e_1)$ as the *length* of S.

Finally, let us specify some relations between events of an event-sequence:

Definition: Given an event sequence S and two events $e, f \epsilon S$, we refer to the result of a function $d(e, f, S) = pos(f, S) - pos(e, S)$ as the *distance* between e and f *in* S. In the case that $d(e, f, S) = 1$, we refer to e and f as successive in S.

Definition: Given an event sequence S and two events $e, f \epsilon S$, we refer to the result of a function $t_r(e, f, S) = \frac{v_t(f) - v_t(e)}{l(S)}$ as the *relative time span* between e and f *in* S.

3.3.1 Event-sequence signature

Listing the according event types instead of concrete events allows viewing event sequences from a much higher level of abstraction. Particularly relevant for event-sequence similarity-measures as proposed in section 6, we refer to such construct as *event-sequence signature*:

Definition: Given an event-sequence S, with e_i addressing the i^{th} event in S, we refer to a sequence of event types $\mathfrak{S}$, $\mathfrak{S} = (T_1, T_2, \dots, T_3), T_j = typeOf(e_j)\ \forall$ j $= 1 \dots |S|$, as the *signature* of S.

In the following, we will address the signature of an event-sequence S through a function $signatureOf(S)$. As usual, we will address the *size* of a signature $\mathfrak{S}$, i.e., the number of event types in $\mathfrak{S}$, through $|\mathfrak{S}|$.

As solely build upon event types, an "$is\ a$" relationship between event-sequence signatures can be defined from subtype-relationships between comprised event types:

Definition: Given two event-sequence signatures $\mathfrak{S}$ and $\mathfrak{T}$, with T_i addressing the i^{th} event type in $\mathfrak{S}$ and U_i addressing the i^{th} event type in $\mathfrak{T}$, $\mathfrak{S}\ is\ a\ \mathfrak{T}$ if $|\mathfrak{S}| = |\mathfrak{T}|$ and $T_j <: U_j\ \forall\ j = 1 \dots |\mathfrak{S}|$.

3.3.2 Illustration

Throughout the following chapters, we will depict an event sequence S as follows: Along a horizontal time axis, we draw nodes representing the single events in S. Unless otherwise stated, the vertical position (from the left to the right) of a certain node corresponds to the according event's position in S, but says nothing about the event's exact occurrence time.

In scenarios where event-attributes are irrelevant, we illustrate single events as circles. If available, the label inside a node displays a (locally unique) name that allows addressing the corresponding event throughout the certain context.[9] Figure 3 below shows an exemplary illustration of an event sequence $S = (e, f, g, h)$:

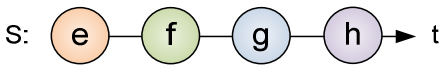


Figure 3: Illustrating event sequence, hiding event attributes

Otherwise, in scenarios where event-attributes are relevant, we illustrate single events as presented in section 3.2.2. Figure 4 below shows an exemplary illustration of S:

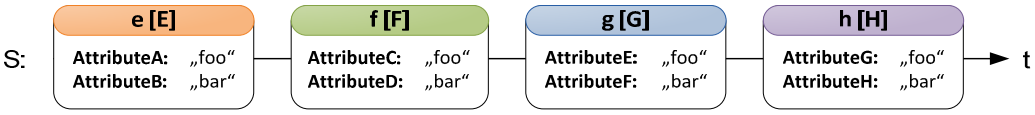


Figure 4: Illustrating event sequence, showing event attributes

In both cases, equally colored nodes indicate equal event types.

[9] For the sake of comprehensibility, we often use names that refer to the certain event's type. In such case, we refer to the earliest event of a type A as a_1 (or, alternatively, a_a), to the second earliest as a_2 (a_b), etc.

4 Technological background

Before reaching the key parts of our thesis - approaches on single-event- and event-sequence similarity - let us take a look at the technological background our considerations build upon: Originally, the proposed algorithms were designed to work as a complement of *SENACTIVE InTime* [46]. The InTime product suite is one of the most promising CEP solutions and was awarded "innovative, impactful and intriguing" in Gartner's Cool Vendor Report 2008 [16]. In section 4.1, we will describe the SARI event model that underlies InTime and, following from that, our implementations of the proposed algorithms. InTime's basic architecture will be presented in section 4.2. In section 4.3, we will present InTime's most prominent analytical application and also the framework for similarity searching, the *SENACTIVE EventAnalyzer*. *Event Access (EA) expressions*, used for accessing events throughout SENACTIVE's product line on event processing, will finally be presented in section 4.4.

Always keep in mind, however, that from a conceptual point of view, the proposed approaches on event similarity are independent from the concrete CEP application, but instead build upon events, event types and event-sequences as defined in section 3 above.

4.1 The SARI event model

Various event models have been proposed in the literature, with complexity ranging from trivial to sophisticate. *SENACTIVE InTime,* according applications and also our implementations of the proposed approaches on similarity searching build upon the SARI ("sense and response infrastructure") event model which originally proposed by Schiefer and Scheufert [44] and described in more detail by Rozsnyai et al. [40].

4.1.1 Event types

Figure 5 shows the essential parts of SARI's event-type model: An event type can inherit from a more general "super event-type". Each event type contains an arbitrary number of event attributes. Hereby, each event attribute corresponds to an event attribute type. Possible attribute types are single-value types, collection types, and dictionary types. A single-value type can either be a base runtime type, such as a string, an integer, etc., or another event type. Attributes of another event type we refer to as *nested events*. Runtime types, however, are the lowest-level attribute types.

Figure 6 shows an exemplary implementation of the SARI event model. Here, a base *TransportEvent* has the runtime-type attributes *TransportAmount* and *Destination*. From the base *TransportEvent*, two event types inherit: *TransportStarted* and *TransportEnded*. *TransportStarted* extends the base event type by three runtime-type attributes. *TransportEnded* adds a runtime-type attribute *RecipientID* and a nested event *StartEvent* of type *TransportStarted*.

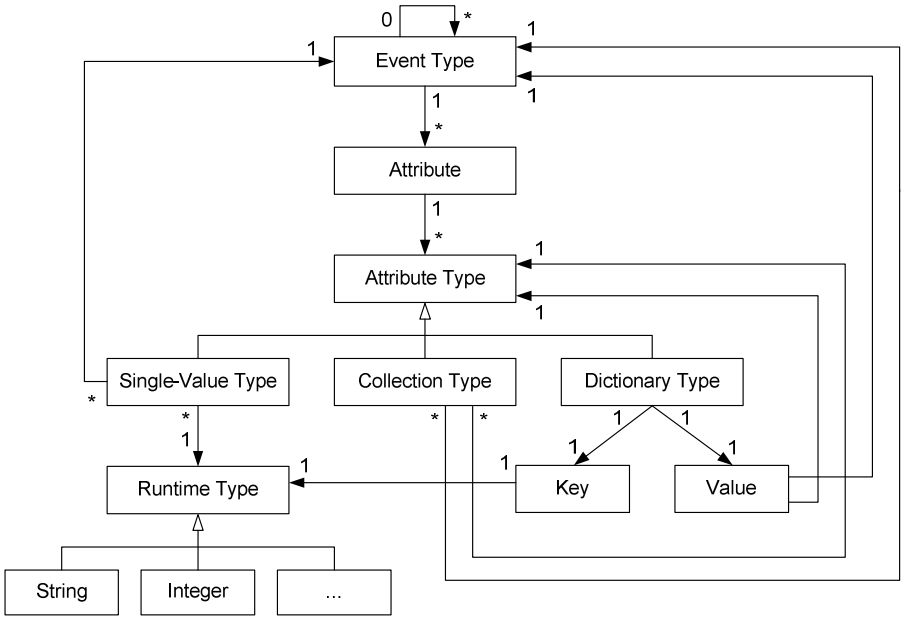


Figure 5: The SARI event type model [41]

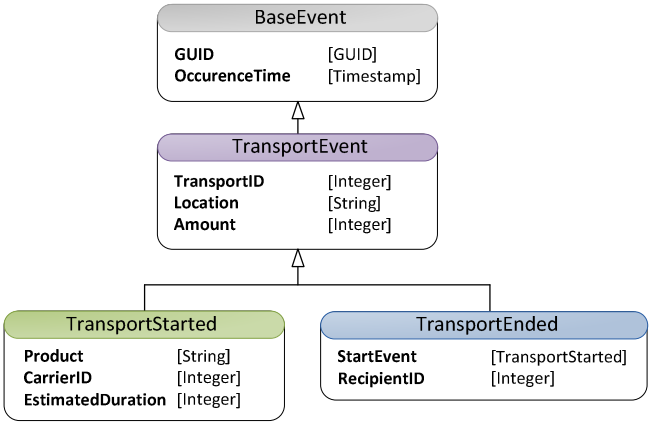


Figure 6: An exemplary implementation of the SARI event type model

4.1.2 Correlations

In many cases, single events have a certain context and are semantically related to other events: A *TaskStarted*-event, for instance, is probably semantically related to a *TaskCompleted*-event with the same task identifier. In SARI, *correlations* [45] are sequences of semantically related events. C*orrelation sets* are user-defined template definitions for how "relatedness" is identified. The correlation set defines tuples of attributes whose values must match in order for events to correlate.

Figure 7 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, ten *BetPlaced*- and two *CashOut*-events.

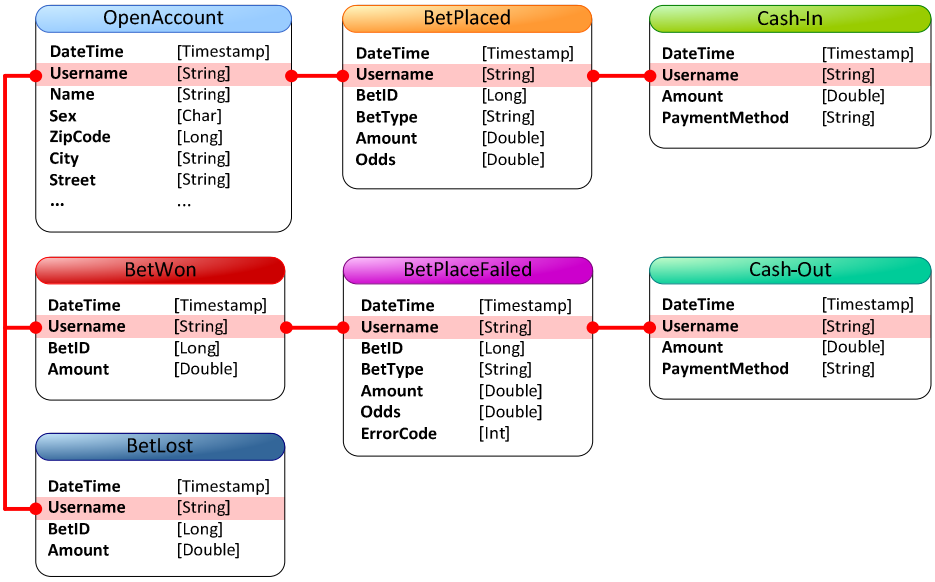


Figure 7: An exemplary correlation set definition

As detected in the underlying event-processing logic and stored in the data repository, correlations form a set of "pre-defined" event sequences in analytical applications. For our implementation of event-sequence similarity, they therefore serve as a starting point for the definition of pattern sequences and also as a basic "universe" for event-sequence similarity-searching. Note, however, that generally, our implementation is fully independent from the concept of correlations and instead could be applied to any other kind of event-sequence.

4.2 Architectural overview

In an upcoming paper [42], we outline SARI's overall architecture as shown in Figure 8 and describe it as follows:

"The bottom of the figure shows source systems (i.e., the event producing components) continuously generate event notifications. The Sense Layer represents the adapters of SARI that can be docked to the event producing systems or the communication infrastructure. The adapters can gather events in either a push or pull process and propagate them into the event processing realms.

The internal communication infrastructure uses an event bus for publishing the received events to the event processing models. SARI uses sockets as a generic interface for sending and receiving events to and from event processing models. The processing of event streams is performed in event processing maps where the event processing flow is modelled with various components according to the business requirements. For the event data storage, SARI uses sockets for "forwarding" event data to the database. In other words, users can define for any type of socket, whether the received events of the socket should be stored in the database.

The EventBase extends SARI's event processing model with an efficient up-to-date operational storage together with retrieval mechanisms for business events for analytical as well as operational purposes. [...] The core access component of the EventBase is a query engine supporting SARI-SQL. It is set on top of the EventBase data repository and exposes its services through programming interfaces and a graphical user interface. "

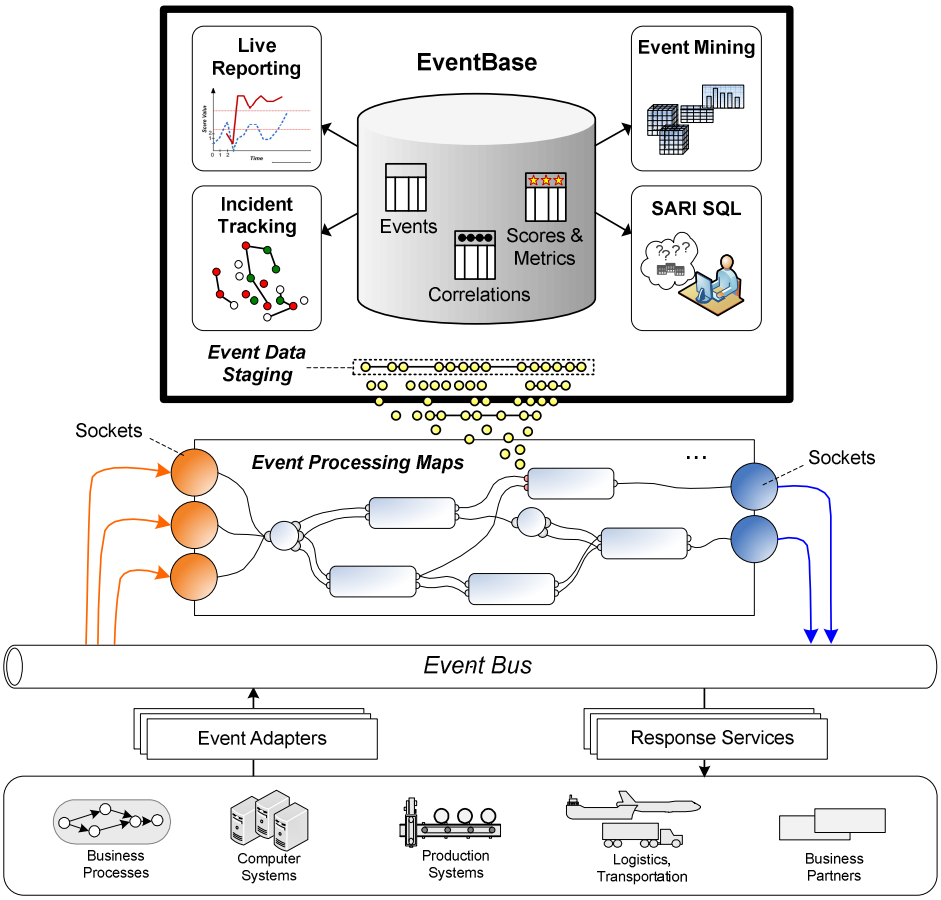


Figure 8: Architectural overview [42]

Event-similarity searching as proposed in the on-hand thesis was intended to set up on SARI's Event Data Warehouse implementation, the EventBase, and to retrieve event data via the afore-said programming interface. As event-similarity unleashes its full value in combination with tailored query mechanisms, event data visualizations etc., we decided to integrate our implementations into the first grown-to-maturity, EventBase-based analysis framework, the EventAnalyzer.

4.3 The EventAnalyzer

The *SENACTIVE EventAnalyzer*^{TM} is a business intelligence tool built on top of the EventBase and the most successful analytical application in SENACTIVE's product line. 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 9 below shows a view of the EventAnalyzer with some of the named modules. For further information on the visualizations provided by the EventAnalyzer, the interested reader may refer to Suntinger et al. [50].

Figure 9: The SENACTIVE EventAnalyzer

Together with diverse extensions, we implemented the proposed approaches on event-similarity as extensions of the Event Analyzer: From a given (single) event or a given correlation, the business analyst may either start a similarity search directly or derive a pattern differing from the original entity. For more details on the integration of the proposed algorithms into the EventAnalyzer, the pattern editor and the visual presentation of calculated similarities, the reader may refer to Suntinger [49].

4.4 Event Access (EA) expressions

Throughout SENACTIVE's product line-up on complex event processing, *accessing* events and sets thereof plays a central role in various features and tasks; consider, for instance, the EventAnalyzer's color-coding feature, where historic events are colored based upon certain event-attribute values. *Event access (EA) expressions* have been developed in order to gain such access through an easy to understand syntax. Following a simple and intuitive notation, they allow performing arbitrary calculations on single events, possibly comprising one or more event attributes, and also complex operations on sets of events.

Digging deeply into EA expressions' theoretic background and their exact syntax, however, is far out of the scope of the on-hand thesis, and so is much of their functionality and power. We will therefore restrict our discussion on EA expression to those (in fact, very basic) aspects that are relevant for calculating similarity between single events. For a more detailed description of EA expressions, the interested reader is referred to Rozsnyai [40].

The simplest EA expression is of the form *EventType.SingleValueAttribute* and addresses a single-value event attribute.[10] Applied to an event of type *EventType*, it returns the event's certain *SingleValueAttribute*-value. The dot-notation also allows addressing event-type attributes (a.k.a. nested events) recursively: An expression *EventType.EventTypeAttribute.SingleValueAttribute* addresses the *SingleValueAttribute* of a nested event *EventTypeAttribute*. Furthermore, EA expressions gain access to collection- and dictionary-typed event attributes: A certain element of a collection can be accessed through *EventType.CollectionAttribute[i]*, where i is the index of the requested element. Similarly, elements of a dictionary can be accessed through *EventType.DictionaryAttribute[key]*.

Finally, note that EA expressions are non-destructive; access to an event through an EA expression is always *read-only* and does not allow altering an event's attribute values. Furthermore, EA expressions are strongly typed, i.e., an EA expression's return type is known a-priori. These two characteristics play a central role in single-event similarity-searching and will be referred to in section 5.4.1.

Example: Consider an event a of event type *TransportEnded* as shown below.

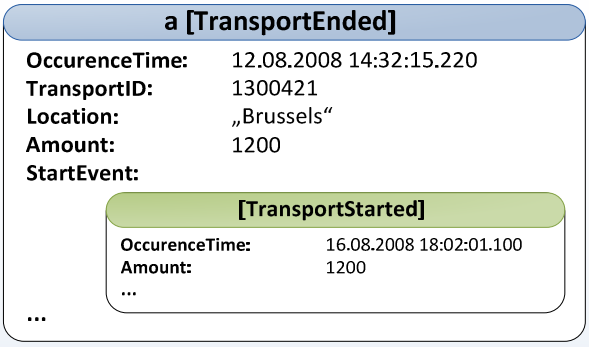


Table 1 lists a few exemplary EA expressions defined for event type *TransportEnded* and their results if applied on a.

EA expression	Result for a
Location	"Brussels"
Location = "Brussels"	true
Amount / 1000	1.2
TransportID + Amount	1301621
StartEvent.Location	"Vienna"
OccurrenceTime – StartEvent.OccurenceTime	99h, 29m, 45s, 880millis

Table 1: Exemplary EA expressions and results thereof

[10] If explicitly declared for a certain event type, referring to this type can be omitted. For sake of simplicity, we will presume such explicit declaration in the following sections.

5 Finding similar events

After successfully clarifying the basic terminology and the technical background that underlies our considerations, let us now continue to the first key part of the on-hand thesis: Finding similar (single) events. Both functional and valuable on its own, the proposed approach will play a central role in the subsequent, second key part of the on-hand thesis, similarity searching for sequences of events.

In section 5.1, we will present basic considerations on event similarity and summarize evolving requirements. In section 5.2, we will demonstrate underlying ideas with a simple example. Concepts derived therein will be formalized in section 5.3, where a basic approach on event-similarity will be presented. The basic approach will be further extended in section 5.4. In section 5.5, we will discuss the most relevant attribute-level similarities. For a more practical understanding, we will present a real-world example in section 5.6. Finally, in section 5.7, we will summarize the pros and cons of the proposed algorithm and also take a look on its properties and its computational complexity.

5.1 Basic considerations and evolving requirements

In section 1.3, we have listed three general demands on similarity searching in event spaces. In this section, let us discuss one further aspect specific to single-event similarity-searching, and, finally, summarize evolving requirements.

5.1.1 An attribute-driven approach

In order to outperform purely subjective (and, in fact, arbitrary) assessments, a comprehensible similarity measure must be based upon objectively given, measurable properties and features. This simple and intuitive rule holds for any kind of complex entity, be it geometric shapes, apples, pears, Bob Dylan songs - or complex events.

When talking about “objectively given properties” of an event, one will immediately think of event attributes as discussed in section 3.2. There are, however, other objectively given characteristics that are usually not accessible through event attributes; consider, for instance, an event’s size in memory. Yet, one major aspect of the complex event processing paradigm is a shift from a more technical to a more business-oriented perspective on business intelligence: For a business analyst, an event represents a certain state change or action in a business process, with event attributes describing it in detail. Technical issues are out of his or her scope. As a consequence, we decided to restrict the set of possibly relevant characteristics to event attributes.

Following the motto “Whatever makes up an event should be accessible through its event attributes”, this approach abstracts from the concrete implementation and allows focusing on the content of the data. Event similarity becomes a toolkit for business analysts. Also, in the rather uncommon case that “purely technical” characteristics shall be part of a similarity measure, an additional attribute can be defined and set accordingly in the event processing logic.

5.1.2 Requirements to an event-similarity framework

From the above discussion and the preliminary notes on similarity searching, we derive the following requirements:

1. An approach on event similarity should be generic and flexible enough to apply for all possible kinds of business events, and all possible business scenarios these events are generated in.
2. The similarity between two events solely depends on event attributes.
3. A similarity measure is always defined based upon a certain interest.
4. A similarity measure is always defined under the active involvement of the domain-expert.
5. A framework for event-similarity allows both defining and managing similarity-measures in a quick and straightforward fashion.

5.2 A geometric approach on event similarity

Due to its specific structure, an event can easily be considered a point in an n-dimensional space, with each dimension corresponding to a certain event attribute. For such data, geometric understandings of similarity where proven useful. Yet, as an event may contain non-numeric attributes, the distance between two such points is not "naturally" given. [11]

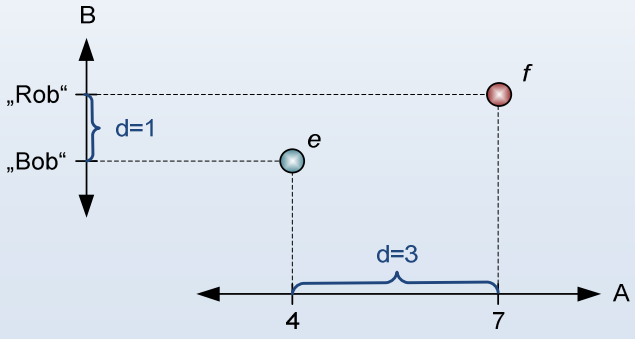
In order to calculate the relative positioning of two events, we therefore consider an additional "layer of abstraction" and let an expert (i.e., someone that knows about the events' semantics) define a distance-function for each event attribute considered relevant. These functions are then applied on the two events' attribute-values:

Example: Consider two events e and f as shown below.

e [SimpleEvent]	
A	4
B	„Bob“

f [SimpleEvent]	
A	7
B	„Rob“

For positioning e and f in a 2-dimensional space, let us use the (algebraic) difference and the *Levenstein distance* [26] for A and B, respectively.



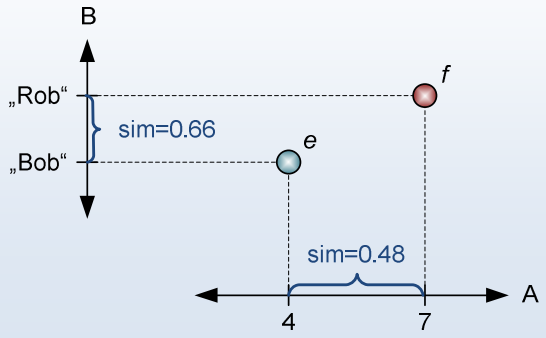
By introducing distance functions, the relative positioning of two events in an n-dimensional space is uniquely defined. Calculating an overall distance between two such points is trivial; distance metrics such as the *city block distance* or the *Euclidean distance* are well-known and easy to understand.

[11] In fact, even for numeric attributes, a variety of distance measures are possible. A few approaches are discussed section 5.5.1.3.

Yet, we should keep in mind that distance functions as defined in section 3.1.2 may return *infinite* for certainly unequal values; consider, for instance, a distance function for Boolean values. This implies, however, that in such case, the distance between two events is infinite regardless of all the other attributes' distances. Clearly, this conflicts with common expectations. Attribute-level distance measures must therefore be bound to a certain range, as, for instance, 0 and 1. As discussed in section 3.1.1, such binding is implied when using similarity measures instead of distance functions.

Again, consider two points in an n-dimensional space, each representing an event. The two points' relative positioning is now defined with respect to a set of similarity measures, each returning a value in the $[0,1]$-interval.

Example: Consider two events *e* and f as shown in the previous example. *e* and f shall now be positioned according to similarities calculated with a *Levenstein*-based similarity and the *relative absolute difference similarity* (see section 5.5.1.3):



Here, the overall set of attribute-level similarities corresponds to the "geometric closeness" of two points in an n-dimension space. An overall closeness, i.e., an overall event-similarity, can now be calculated in a straightforward fashion; one could, for instance, compute the average attribute similarity.

5.2.1 Similarity measures and event types

Event similarity as shown in the above section comprises some or all of the compared events' attributes. This leads to a fairly intuitive conclusion: A similarity measure is always bound to a certain set of event attributes. These event attributes must be available in order to successfully apply the similarity measure. Consider, for instance, a similarity measure that comprises an event attribute that holds the location where the event was originally created: The similarity measure will hardly yield meaningful results if applied to a pair of events that simply do not contain such an attribute.

In the event model that underlies our solution, such sets of event attributes can be defined through event types. Each event implementing a certain event type provides the event attributes declared therein, each having the defined name and being of the defined type. Therefore, in order to "take a walk on the (type-)safe side", one could bind each similarity measure to a certain event type. A similarity measure for event type T, for instance, is then restricted to the event attributes declared in T, and can only be applied to events of T or of subtypes of T.

In our approach, we opted for such bindings: Event similarity becomes type-safe.

5.3 Measuring event similarity

Let us now formalize the above-presented, distance-based approach on single-event similarity:

5.3.1 A basic similarity measure for events

Consider an event type T defining n event attributes, and let a_i and $\mathbb{A}_i$ address the name and the type of the i^{th} event attribute in T. We now let the user define a collection of attribute-level similarity-measures $sim_1, sim_2, \dots, sim_n$ with $sim_i: \mathbb{A}_i \times \mathbb{A}_i \to [0,1]\ \forall\ i = 1 \dots n$, i.e., one similarity measure per event attribute defined in T. Also, we let the user define an aggregation function $f: [0,1]^n \to [0,1]$ that allows calculating an overall similarity from n attribute-level similarities.

A basic similarity measure $sim: T \times T \to [0,1]$ on events of type T is then defined as follows:

$$sim(e,f) = f\left(sim_{a_1}\left(v_{a_1}(e), v_{a_1}(f)\right), \dots, sim_{a_n}\left(v_{a_n}(e), v_{a_n}(f)\right)\right)$$

The similarity between two events is thus calculated from the attribute-level similarities that exist between according attribute-values of the compared events.

It is easy to see that selecting appropriate attribute-level similarity-measures is the decisive step in defining an event-level similarity-measure: By declaring which values are assessed to be similar and which are not, one "formalizes" the semantics that underlie an event attribute. We have already stated that defining a similarity-measure for complex events must involve the domain expert; at this point, this should become apparent: Only the domain expert can decide how similar "4" is to "7" with respect to a certain event attribute, defined in a certain event type, used in a certain business domain.

5.3.2 Weights

Configured adequately, the present definition of a similarity measure yields meaningful results for any kind of business events. Note, however, that each event attribute has an equal impact on the overall event-similarity. This clearly conflicts with the idea of a flexible and generic approach on event similarity. With different questions in mind, business analysts will assess one and the same event attribute to be essential on one day and of little importance, or even irrelevant, on another.

In section 5.2, we have stated that our approach is based upon geometric ideas on similarity. When considering pairs of events as points in an n-dimension space in a relative positioning to each other, it becomes obvious that each dimension - event attribute, respectively - could be scaled by a distinct scale factor.

Again, consider an event type T defining n event attributes, and let a_i and $\mathbb{A}_i$ address the name and the type of the i^{th} event attribute in T. In addition to attribute-level similarity-measures, we now let the user define a collection of weights $\omega_1, \omega_2, \dots, \omega_n$ with $\omega_i \in [0,1]\ \forall\ i = 1 \dots n$ and $\sum_{j=1}^{n} \omega_i = 1$ (i.e., summing to unity), with ω_j defining the impact of a_j on the overall similarity between two events.

With an extended aggregation function $f: [0,1]^n \times [0,1]^n \rightarrow [0,1]$ that allows taking weights into account,[12] a weighted similarity-measure $sim_w: T \times T \rightarrow [0,1]$ on events of type T is then defined as follows:

$$sim_w(e,f) = f\left(sim_1\left(v_{a_1}(e), v_{a_1}(f)\right), \ldots, sim_n\left(v_{a_n}(e), v_{a_n}(f)\right), \omega_1, \ldots, \omega_n\right)$$

5.3.3 Summary

Let us recapitulate our approach on single-event similarity presented thus far: Given an event type T defining n event attributes, with a_i and $\mathbb{A}_i$ addressing the name and the type of the i^{th} event attribute in T, configuring a similarity-measure on single-events can be considered defining a 3-tuple

$$\langle S, W, f \rangle,$$

where

- S is a collection of attribute-level similarity-measures $sim_1, sim_2, \ldots, sim_n$ with $sim_i: \mathbb{A}_i \times \mathbb{A}_i \rightarrow [0,1]\ \forall\ i = 1 \ldots n$; each in accordance with the certain semantics of the corresponding event attribute,
- W is a collection of weights $\omega_1, \omega_2, \ldots, \omega_n$ with $\omega_i \in [0,1] \forall\ i = 1 \ldots n$ and $\sum_{j=1}^{n} \omega_j = 1$; each defining the impact of the corresponding event attribute on the similarity between two events, and
- f is an aggregation function $f: [0,1]^n \times [0,1]^n \rightarrow [0,1]$.

In order to overcome some abstractness, allow us to anticipate some attribute similarity measures in the following example. The attribute similarity measures which will be discussed in detail in section 5.5.

Example: Consider an event type *TransportEvent* as shown in Figure 6. The business analyst wants to define a similarity measure that takes the event attributes *TransportID*, *Location* and *Amount* into account. Based upon both the business analyst's interest and the data's semantics, the business analyst chooses similarity measures and weights as shown in Table 2.

Event attribute	Similarity measure	Weight
TransportID	Lookup table similarity for numeric values	0.4
Location	Lookup table similarity for strings	0.3
Amount	Normalized absolute difference	0.3

Table 2: Exemplary attribute-level similarity-measures and weights

The two remaining attributes, *GUID* and *OccurrenceTime*, are not taken into account and weighted by zero.

[12] In our implementation, we use the weighted average originally proposed by Gowser [19]. As the name implies, it calculates the weighted average of the attribute-level similarities:

$$f_{wa}\left((s_1, s_2, \ldots, s_n), (\omega_1, \omega_2, \ldots, \omega_n)\right) = \sum_{i=1}^{n} (s_i * \omega_1)$$

The weighted average is a highly intuitive and should be appropriate in most cases. Dey et al. [12], however, point out that the distance between complex entities can be calculated in a variety of ways. In real-world scenarios, one usually chooses a metric from the Minkowski family of metrics, from which the *weighted average* is the simplest case.

5.4 Possible extensions

An approach on single-event similarity as presented thus far should be flexible enough for covering a wide range of business scenarios and problems. Yet, we found out that in practice, business analysts may require (apparently specific) features and options that exceed the original expressiveness. Unlike the above-presented "building blocks" of a similarity-measure, however, according configurations are not *required* for a similarity-measure to work; we thus understand them as possible extensions to a functional base approach. In the on-hand section, we will present two extensions - *attribute functions* and *required attributes* – and, finally, summarize the resulting "full-featured" approach on single-event similarity.

Unless otherwise stated, the full-featured approach as summarized in section 5.4.3 is assumed throughout the following sections.

5.4.1 Attribute functions

By handling each event attribute separately and completely independently from all others, the present approach results in surprisingly simple similarity-measures even for complex events. It is, however, not possible to comprise "event-internal" relations between attributes, such as, for instance, the difference between two numeric attributes *A* and *B*. Hence, a business analyst cannot assess two events to be similar if in both events, two or more attributes are in a "similar" relation to each other. Consider the following example:

> **Example:** *TransportEnded*-events contain information about a transport's start- and end-time, among others. To avoid redundancies, the application designer decided not to implement a distinct attribute holding a transport's duration.
>
> A business analyst may now be interested in transports that are of a similar duration, certainly independent from a transport's certain start time. Intuitively, one would suggest considering both the event's start-time attribute and the event's end-time attribute, e.g., by weighting them equally. This approach fails, however, as it clearly considers the transport's actual time of execution instead of its duration only.

In order to address such issues, we decided to extend the original approach by providing a mechanism that allows performing *arbitrary calculations* on the event attributes of events. Before going into details, however, let us define the concept of *attribute functions* on a given event type T*:*

> **Definition:** Given an event type $T \subseteq \mathbb{E}$ and a set of supported return types $\mathfrak{A}$, we refer to a function $f: T \to \mathbb{A}, \mathbb{A} \in \mathfrak{A},$ as *attribute function* if
>
> 1. f is strongly typed, i.e., the type of its output must be known a-priori, and
> 2. f is non-destructive, i.e., it does not alter the event's state.

[13, 14]

[13] The stated conditions are implicitly given for functions in a strictly mathematical sense. In practice, however, attribute functions may be implemented in languages that allow both the definition of non-typed functions and modifying event data.

With a so-defined attribute-function, a user can access any given relation between the attributes of an event. He or she may, for instance, calculate the difference between two numeric attributes, or concatenate two strings. Thus, in telling us "something" about the underlying state-change or action, attribute functions can be considered somewhat equivalent to "ordinary" event attributes.

An intuitive consequence is that of treating attribute functions in the same way as event attributes, i.e., to allow taking them into account for event-similarity. Let us go one step further: Elementary attribute-functions simply return a certain event-attribute's value. Taking such functions into account for event-level similarity is thus equivalent to comprising the according event-attributes themselves.

Hence, for the sake of consistency, let us reject the original idea of measuring similarities between "plain" attribute-values. Instead, let us add a further layer of abstraction and measure attribute-level similarities between the results of user-defined attribute-functions:[15]

Thus, given an event type T, we let the user define a collection of attribute-functions $af_1, af_2, \dots, af_k$ with $af_i: T \to \mathbb{A}_i,\ \mathbb{A}_i \in \mathfrak{A}\ \forall\ i = 1 \dots k$. For each attribute function, we now let the user choose an adequate attribute-level similarity-measure, i.e., we let the user define a collection of attribute-level similarity-measure $sim_1, sim_2, \dots, sim_k$ with $sim_i: \mathbb{A}_i \times \mathbb{A}_i \to [0,1]\ \forall\ i = 1 \dots k$. Also, we let the user define a collection of weights $\omega_1, \omega_2, \dots, \omega_k$ with $\omega_i \in [0,1]\ \forall\ i = 1 \dots k$ and $\sum_{j=1}^{k} \omega_j = 1$, with ω_j defining the impact of af_j on the overall similarity between two events.

With an extended aggregation function $f: [0,1]^k \times [0,1]^k \to [0,1]$, a similarity measure $sim_{af}: T \times T \to [0,1]$ featuring attribute functions is then defined as follows:

$$sim_{af}(e,f) = f\big(sim_1\big(af_1(e), af_1(f)\big), \dots, sim_k\big(af_k(e), af_k(f)\big), \omega_1, \dots, \omega_k\big)$$

Example, continued: As part of his or her similarity measure for *TransportEnded*-events, the business analyst defines an attribute function that calculates the difference between the event-attributes *EndTime* and *StartTime*; with EA expressions, a so-defined attribute function is written

TransportEnded.StartTime – TransportEnded.EndTime.

The business analyst chooses the *normalized absolute difference similarity* (see section 5.5.1.3) and weights the attribute function by 1.0.

Note that in the following, we will speak of *attributes, attribute-level similarity-measures* and *attribute-level similarities* also when, in fact, *attribute-functions* are used.

[14] In our solution, we realize attribute functions with *EA expressions*. Besides being both strongly typed and non-destructive, EA expressions are used among numerous features of the SENACTIVE *EventAnalyzer* and hence should be familiar to the user. Some restrictions must be made, though, to ensure that an EA expression returns a supported type.

[15] Keep in mind that most event-level similarity-measures are still based upon "plain" attribute-values, i.e., on elementary attribute-functions. In the *EventAnalyzer*, we therefore "hide" that additional layer of abstraction to a large extent: Primarily, we let the user choose *event attributes* and generate the according, elementary attribute-functions implicitly. Only when complex calculations are required, attribute functions must be formulated explicitly.

5.4.2 Required attributes/attribute-functions

In certain cases, business analysts may consider two events *certainly unequal* if one or more attributes are *not certainly equal* with respect to the selected attribute-level similarity-measures. In other words, when calculating the similarity between two events, a similarity-value of 0 is expected in any case where one or more *required attribute-level similarities*, i.e., similarity-values resulting from *required attributes*, are below 1. Otherwise, if all required attribute-level similarities are 1, an event-level similarity – one may call it "base similarity" – shall be calculated as usual. Consider the following example:

> **Example:** After exposing a previously undetected fraud case, the betting broker is interested in finding similar betting scenarios in its legacy data. The analyst plans to perform a similarity search on a characteristic *BetPlaced*-event, but wants to exclude events that have already triggered a fraud alarm from the result set. Intuitively, one would argue to simply include the *AlarmTriggered*-attribute into the similarity measure and weight it adequately. This would, in fact, reduce the similarity score of events that have already triggered a fraud alarm. Excluding them, however, would require messing around with a threshold, which is imprecise, time-killing and likely to eliminate too large parts of the result set.

In order to address such issues, we decided to extend our approach as follows:

Given an event type T, a set of supported return types $\mathfrak{A}$ and a (however-defined) base similarity-measure $sim: T \times T \to [0,1]$ on events of type T, we let the user define an additional collection of *required* attribute-functions $af'_1, af'_2, \dots, af'_l$ with $af'_i: T \to \mathbb{A}'_i$, $\mathbb{A}'_i \in \mathfrak{A}\ \forall\ i = 1 \dots l$. Also, we let the user choose adequate attribute-level similarity-measures $sim'_1, sim'_2, \dots, sim'_l$ with $sim'_i: \mathbb{A}'_i \times \mathbb{A}'_i \to [0,1]\ \forall\ i = 1 \dots l$.

A similarity measure $sim_r: T \times T \to [0,1]$ featuring *required attributes* is then defined as follows:

$$sim_r(e,f) = \begin{cases} sim(e,f), & sim'_i\big(af'_i(e), af'_i(f)\big) = 1\ \forall\ i = 1 \dots l \\ 0, & otherwise \end{cases}$$

> **Example, continued:** The business analyst chooses the default similarity measure for Boolean values (section 5.5.1.2) for the *AlarmTriggered*-flag and marks the attribute as *required*. Based upon the remaining attributes, the analyst defines a similarity measure just as usual. When executing the similarity search, a similarity value of zero is calculated for all events that have triggered a fraud alarm.

5.4.3 Summary

At this point, let us recapitulate the extended, "full-featured" approach on single-event similarity: Given an event type T and a set of supported attribute-types $\mathfrak{A}$, configuring a similarity-measure on single-events can be considered defining a 5-tuple

$$\langle F, S, W, f, F', S' \rangle,$$

where

- F is a collection of attribute-functions $af_1, af_2, \dots, af_k$ with $af_i: T \to \mathbb{A}_i$, $\mathbb{A}_i \in \mathfrak{A}\ \forall\ i = 1 \dots k$,
- S is a collection of attribute-level similarity-measures $sim_1, sim_2, \dots, sim_k$ with $sim_i: \mathbb{A}_i \times \mathbb{A}_i \to [0,1]\ \forall\ i = 1 \dots k$; each in accordance with the certain semantics of the corresponding attribute function,
- W is a collection of weights $\omega_1, \omega_2, \dots, \omega_k$ with $\omega_i \in [0,1]\ \forall\ i = 1 \dots k$ and $\sum_{j=1}^{k} \omega_i = 1$; each defining the impact of the corresponding attribute function on the similarity between two events,
- f is an aggregation function $f: [0,1]^k \times [0,1]^k \to [0,1]$ that allows calculating an overall event-level similarity - not taking required attributes into account - from attribute-level similarities and weights,
- F' is a collection of required attribute-functions $af'_1, af'_2, \dots, af'_l$ with $af'_i: T \to \mathbb{A}'_i$, $\mathbb{A}'_i \in \mathfrak{A}\ \forall\ i = 1 \dots l$, and
- S' is a collection of attribute-level similarity-measures for required attribute functions $sim'_1, sim'_2, \dots, sim'_l$ with $sim'_i: \mathbb{A}'_i \times \mathbb{A}'_i \to [0,1]\ \forall\ i = 1 \dots l$; each, again, in accordance with the certain semantics of the corresponding attribute function.

Due to the increased level of complexity, let us demonstrate the "full-featured" calculation of event-level similarities in pseudo code:

Name:	getSimilarity	
Description:	Calculates the similarity between two events.	
Input:	**e:**	The first event.
	f:	The second event.
	functions:	A field of attribute functions.
	simMeasures:	A field of similarity measures for **functions**.
	weights:	A field of weights for **functions**.
	aggr:	An aggregation function.
	requFunctions:	A field of required attribute functions.
	requSimMeasures:	A field of similarity measures for **requFunctions**.
Output:	A similarity between 0 and 1.	
Variables:	**i:**	An index.
	eResult:	The result of an attribute-function on **e**.
	fResult:	The result of an attribute-function on **f**.
	similarity:	An attribute-level similarity.
	similarities:	A field of attribute-level similarities.

```
01:     // Evaluate required similarities
02:     for i = 0 to requFunctions.length step 1
03:             double eResult = requFunctions[i](e);
04:             double fResult = requFunctions[i](f);
05:             double similarity = requSimMeasures[i](eResult, fResult);
06:             if (similarity < 1) then
07:                     return 0;
08:             end
09:     end
10:
11:     // Return "1.0" if no "regular" attribute functions were chosen
12:     if (functions.length == 0) then
13:             return 1;
14:     end
15:
16:     // Calculate attribute-level similarities
17:     double[] similarities = new double[functions.length];
18:     for i = 0 to functions.length step 1
19:             double eResult = functions[i](e);
20:             double fResult = functions[i](f);
21:             double similarity  = simMeasures[i](eResult, fResult);
22:             // Add to attribute-level similarities
23:             similarities[i] = similarity;
24:     end
25:
26:     // Calculate event-level similarity
27:     return aggr(similarities, weights);
```

Algorithm 1: Calculating event-level similarities

5.5 Measuring attribute-level similarities

In our framework for measuring event similarity, we allow the analyst choosing from a large set of type-specific attribute-level similarity-measures. In the following, let us discuss the most prominent attribute-level similarity-measures for all attribute types defined in the SARI event model. As those types are considered *supported* for our implementation of attribute functions, the discussed similarity-measures can be used consistently among both the basic approach as discussed in section 5.3 and the extended approach as discussed in section 5.4.

5.5.1 Runtime types

5.5.1.1 Lookup tables

Before discussing the various, type-specific similarity-measures provided "out of the box", let us pay some attention to a simple but particularly useful extension to regular similarity measures.

Anderberg [3] suggests *lookup tables*, where the user "manually" assigns similarity values to arbitrary pairs of values. Thus, by letting the user define a function explicitly, lookup tables allow creating similarity measures from scratch. This has two notable advantages: Besides working for all runtime types declared in the SARI event model, it allows expressing highly purpose-specific, "semantic" relationships. Consider for, instance, the following lookup-table similarity-measure from the sports domain:[16]

Term 1	Term 2	Similarity
Rugby	American Football	1.0[17]
Free throw	Penalty	0.7
Penalty	Direct free kick	0.2

Table 3: An exemplary lookup-table similarity-measure from the sports domain

It is easy to see that defining a lookup table requires at least partly knowledge about the set of possible attribute values. Such knowledge, however, is available in many cases. A business analyst that analyses a betting broker's legacy data, for instance, knows about possible values for attributes that hold the sport type that is assigned to some of the captured events.

5.5.1.2 Booleans

Measuring the similarity between two Boolean values is somehow insipid: It should be easy to follow that zero is certainly unequal to non-zero, and vice versa. A similarity function for Boolean values $sim_b: \{0,1\} \times \{0,1\} \rightarrow \{0,1\}$ must therefore be defined as follows.

$$sim_b(a,b) = \begin{cases} 1, & a = b \\ 0, & a \neq b \end{cases}$$

We omit an example for obvious reasons. Keep in mind, however, that comprising Boolean values into an event-similarity measure may be highly valuable in combination with a well-considered weighting model.

[16] A similarity function defined via a lookup table is, of course, symmetric. At this point, one might argue that a transitive relation is implied as well; one could, for example, expect a certain similarity relation between the terms "Free throw" and "Direct free kick", as $sim(\text{"Free throw"}, \text{"Penalty"}) = 0{,}7$ and $sim(\text{"Penalty"}, \text{"Direct free kick"}) = 0{,}2$. In our implementation, such relation is not given per se: If required in the certain domain, the user must define appropriate value-pairs explicitly.

[17] Experts in physical ball sports may forgive my ignorance.

5.5.1.3 Numeric types

The SARI event model that underlies our solution defines several numeric attribute types, such as *Integer*, *Float* or *Double*. Similarity measures that are based upon arithmetic operations usually apply to all of these; so do the measures presented below.

As one can perform however-defined transformations on the absolute distance between two values, most distance measures for numeric types can be calculated in a relatively straightforward fashion. A similarity measure that has been excessively used in cluster analysis is the absolute difference between two values, normalized on a $[0,1]$-scale. With $A \subset \mathbb{R}$ addressing the overall set of values existing in the given context, the *normalized absolute difference similarity* $sim_{nad}: A \times A \rightarrow [0,1]$ is defined as follows:[18]

$$sim_{nad}(x,y) = 1 - \frac{|\,x-y\,|}{\max\limits_{a\in A} a - \min\limits_{a\in A} a}$$

In certain cases, a relative similarity measure may be more appropriate. We therefore provide the *relative absolute difference similarity,* $sim_{nad}: \mathbb{R} \times \mathbb{R} \rightarrow [0,1]$, which is calculated as follows:

$$sim_{rad}(x,y) = 1 - \frac{|\,x-y\,|}{\max\limits_{v\in\{x,y\}} v}$$

The above similarity measures work well for numeric attributes that have continuous or ordinal values, such as amounts, prices, grades, etc. In many scenarios, though, one will use numeric attributes for holding categorical information: Consider an event that shall contain a reference to a complex (database) entity, as, for instance, a certain customer entity in a customer relationship management system (CRMS). With adaptability in mind, the application designer will probably define a numeric attribute *CustomerID* that holds the primary database key of the referenced entity. In such a case, the above similarity measures are obviously useless: A business analyst will be interested in similar customers rather than in similar customer IDs.

Thus, when using numeric attributes for holding categorical data, one usually needs a similarity measure that takes underlying semantics into account. Besides the ever-present possibility to implement a special-purpose similarity-measure by oneself (that could, for instance, query the corresponding database entities and perform a domain-specific comparison on the customers' purchases during the last few months), the business analyst may apply a *lookup table similarity* for numeric values. A detailed description of the generic lookup-table similarity-measure was given in section 5.5.1.1.

5.5.1.4 Time stamps and time spans

As both kinds of data can be transformed into (de-facto continuous) amounts of sufficiently small units, time stamps as well as time spans can be considered numeric types in the proposed approach on event-similarity: For time spans, the transformation is trivial. For time stamps, a constant reference time stamp is required;[19] all possible time stamps can then be transformed into temporal differences - i.e., time spans - to the given reference point. As based upon the UNIX (or POSIX) time, such a reference point is implicitly given with the April 1st, 1970 in most modern programming languages.

---

[18] Note that normalizing the difference requires a-priori knowledge about the overall range of the attribute's values. In our architecture for attribute similarities, we provide a mechanism that allows read-only access to the current set of events, and can be used to analyze the current set of events, and to configure the similarity measure accordingly.

[19] Note that when applying the *relative absolute difference similarity*, the reference time stamp may have seemingly paradox effects on the similarity between two time stamps.

5.5.1.5 Strings

For decades, string similarity has been a major topic in information technology. Discussing prominent approaches, however, is outside the scope of this thesis; interested readers may refer to the rich body of according literature (see, for instance, [16] and [58]).[20]

5.5.2 Collections and Dictionaries

For collection-types such as *sets* and *lists*, we allow choosing a set-based similarity-measure in accordance with Sjoberg's feature-based understanding of similarity [48]: With $\mathbb{A}$ denoting an arbitrary single-value type, a similarity measures on collections of $\mathbb{A}$s, $sim_{col}: \mathbb{A}^m \times \mathbb{A}^n \rightarrow [0,1], m,n \in {\mathbb{R}_0}^+$, is defined as follows:

$$sim_{col}(V,W) = \frac{|V \cap W|}{|V \cup W|}$$

For dictionaries, a similarity measure sim_{dict} is defined equivalently.

The above similarity measures calculate "overall" similarities between collections of either values or key/value pairs, i.e., they take *all* the collections' elements into account. Also, eventual orders (as, for instance, given in a *list*) are not considered at all. sim_{col} and sim_{dict} are thus well suitable if there is no specific, "meaningful" structure in the values of a collection-typed attribute, or if such structure is not considered relevant by an analyst.

Consider, however, an event type *WebshopVisit* with a list-typed attribute *VisitedProducts* holding product IDs in the order of visits. Here, a business analyst may be interested in all customers that started their shopping trip with visiting a product somewhat similar to the latest Bob Dylan single. It is easy to see that such scenarios are far too specific for providing according similarity-measures out of the box. Yet, depending on the expressiveness of the certain implementation, attribute-functions may be used to access multi-value attributes in a directed manner. With EA expression, for instance, the above product ID was accessed through *WebshopVisit.VisitedProducts[0]*.

5.5.3 Event types/nested events

Event-typed attributes, a.k.a. nested events, may provide extensive information to the business analyst and thus require highly purpose-specific attribute similarity-measures. As the on-hand approach on event similarity was designed to address such issues, it seems obvious to also use it on the level of event attributes.

The described, recursive approach allows highest precision, but to the expense of a fairly longsome definition process. Alternatively, when addressing only a few of the nested event's attributes, it may be sufficient to address these by defining appropriate attribute functions. Attribute functions are required in any case where calculations on both top-level attributes and sub-level attributes shall be performed. Again, a sufficiently expressive implementation of attribute function is required.

[20] In our implementation, we integrated Sam Chapman's excellent *SimMetrics* library [9] and allow the user choosing from the variety of string-similarity measures available therein.

5.5.4 Dealing with *null*, *NaN* and *infinity*

In the CEP solution that underlies our implementation, all types of event attributes are implemented as *reference types*. Consequently, both complexly and primitively typed attributes may hold *null*-values. Also, attributes of numeric types may hold exceptional values, i.e., *NaN (not a number), positive infinity* and *negative infinity*. In C#, *NaN* results from dividing zero by zero. Similarly, positive or negative *infinity* is the result of dividing a positive or negative value by zero.

When calculating the similarity between events, both *null*-values and exceptional numeric values may lead to problems.

5.5.4.1 Calculating similarities

It is easy to see that without an exceptional handling of the above values, most of the presented similarity-measures fail on null-values. In our solution, the implementation of such handling is left to the similarity-technique. In the similarity-techniques that are provided out of the box, however, we follow a simple approach that follows the default comparison implementation in C#:

Given a however-defined attribute-level similarity-measure $sim: \mathbb{A} \times \mathbb{A} \rightarrow [0,1]$, we define an extended similarity-measure $sim': \mathbb{A}' \times \mathbb{A}' \rightarrow [0,1]$ with $\mathbb{A}' = \mathbb{A} \cup \{null, NaN, +\infty, -\infty\}$, adding the handling of exceptional values to sim, as follows:

	$\boldsymbol{null}$	$\boldsymbol{NaN}$	$+\infty$	$-\infty$	$\boldsymbol{x} \in \mathbb{A}$
$\boldsymbol{null}$	1	0	0	0	0
$\boldsymbol{NaN}$	0	0	0	0	0
$+\infty$	0	0	1	0	0
$-\infty$	0	0	0	1	0
$\boldsymbol{y} \in \mathbb{A}$	0	0	0	0	$sim(x, y)$

Table 4: Dealing with exceptional values

Note that in C#, the expression $(double.NaN == double.NaN)$ yields $false$, wherefore $sim'(NaN, NaN) = 0$.

Unfortunately, the presented approach is certainly insufficient in many cases. Alternatively, one can use attribute function to transform exceptional values (e.g., to zero or an empty string), or apply a *lookup-table similarity measure*, which allows defining domain-specific similarities for $(null, null)$, (NaN, NaN), $(null, NaN)$, etc.

5.5.4.2 Normalizations

In 5.5.1.3, we have discussed the *Normalized absolute difference similarity,* which is calculated by normalizing the absolute difference between two numeric values. When including (positive and/or negative) infinity into the total range of values, such normalization must yield unsatisfying results, as

$$\frac{x}{\pm\infty} = 0 \; \forall \, x \in \mathbb{N}.$$

In our implementations of similarity-measures, infinity is already handled exceptionally. We therefore decided to ignore infinity when calculating the overall range or values. Other implementations, however, may follow a different approach.

5.6 Example

In the above sections, have defined a generic approach on event-level similarity and discussed a variety of attribute-level similarity-measures to build upon. Let us now demonstrate the proposed approach with a real-world example.

5.6.1 Defining a similarity measure for single events

Consider two event types from the logistics-domain, *TransportEnded* and *TransportStarted*, as shown in Figure 10. Note that besides a number of runtime-typed attributes, *TransportEnded*-events contain a nested *TransportStarted*-event:

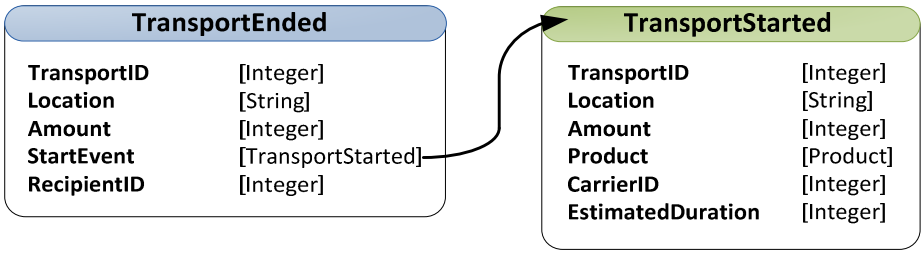


Figure 10: The structure of *TransportEnded*- and *TransportStarted*-events

Based upon his or her certain interest, the business analyst defines an event-level similarity measure as shown in Table 5. Here, attribute functions are depicted as EA expressions:

Attributes/Attribute functions	Return type	Similarity measure	Weight/Required
StartEvent.EstimatedDuration - (CreationTime - StartEvent.CreationTime)	Timestamp	Normalized absolute difference	0.6
Amount	Integer	Relative absolute difference	0.2
Location	String	Lookup table similarity Brussels – Amsterdam ➔ 0.9 Brussels – Barcelona ➔ 0.3	0.2
RecipientID	Integer	Normalized absolute difference	*Required*

Table 5: Attribute-level similarity measures

5.6.2 Calculating event similarities

Let us now apply the above-defined similarity measures to a pair of *TransportEnded*-events, e and f, as shown in Figure 11:

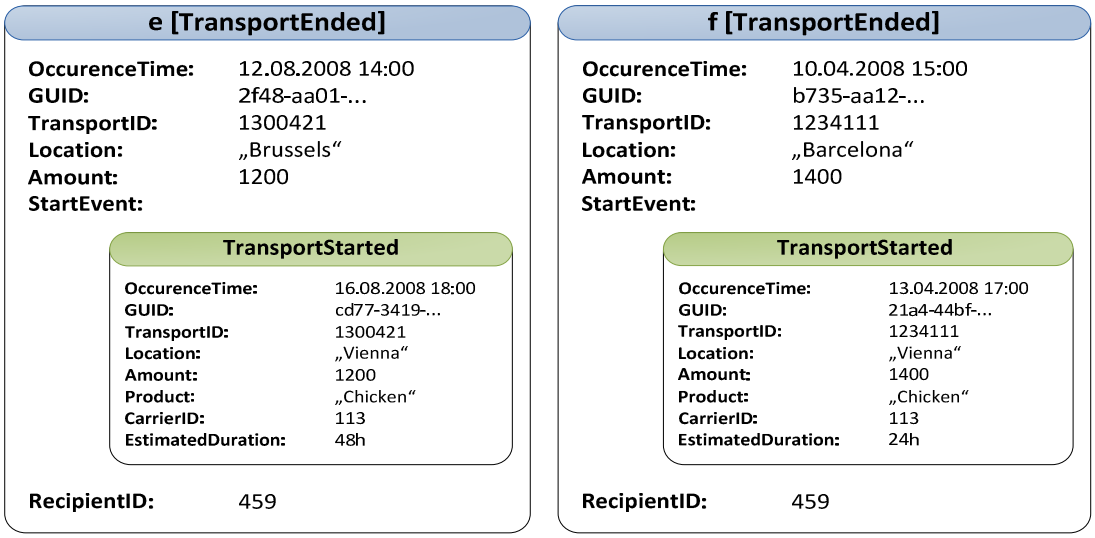


Figure 11: Exemplary *TransportEnded*-events

Table 6 below shows the attribute-level similarities calculated for *e* and f. As using the *normalized absolute difference* for the delay (i.e., for "StartEvent.EstimatedDuration - (CreationTime - StartEvent.CreationTime)"), let us assume that delays between *-10h* and *70h* are calculated for the overall set of *TransportEnded*-events.

i	af_i	$af_i(e)$	$af_i(f)$	$sim_i(faf_i(e), af_i(f))$
1	**StartEvent.EstimatedDuration - (CreationTime - StartEvent.CreationTime)**	52h	50h	0.975
2	**Amount**	1200	1400	0.858
3	**Location**	"Brussels"	"Barcelona"	0.300
4	**RecipientID**	459	459	1.000

Table 6: Attribute-level similarities

As the only required attribute function, "RecipientID", results in an attribute-level similarity of 1.0, the "basic" similarity is calculated. Table 7 below thus shows the calculation of the overall, event-level similarity between *e* and f. As an aggregation function, we use the *weighted average*:

i	$sim_i(faf_i(e), af_i(f))$	ω_i	$sim_i(faf_i(e), af_i(f)) * \omega_i$
1	**0.975**	0.6	0.585
2	**0.858**	0.2	0.172
3	**0.300**	0.2	0.060
			$\sum_{i=1}^{3} sim_i(faf_i(e), af_i(f)) * \omega_i = \mathbf{0.817}$

Table 7: Calculating the event-level similarity from attribute-level similarities

With the above defined similarity-measure, a similarity of 0.817 is calculated between *e* and f.

5.7 Discussion

Thus far, we have discussed a simple yet powerful approach on event-sequence similarity and demonstrated it in a concrete example. Finally, in the on-hand section, let us summarize the "pros and cons" of the proposed algorithm and also take a look on its properties and its complexity.

5.7.1 Pros and cons

As building upon the somewhat "straightforward" geometric understanding of similarity, the presented approach on event-sequence similarity is highly intuitive and should be easy to understand for business analysts. Yet, by extending the very basic distance-based approach on similarity between vectors by two user-configurable „layers of abstraction", *attribute-functions* and corresponding *attribute-level similarity-measures*, it gains a broad expressiveness and should be applicable in most scenarios. Also, note that the proposed approach handles times of occurrence just like any other event-attribute. Therefore, it is generally conceivable to use the on-hand approach for "time-independent" complex entities, such as, for instance, *entity beans*.

Keep in mind, however, that the great flexibility of the proposed approach requires lots of configuration and the intensive involvement of domain-experts. All the more clearly, a sophisticated management of existing filter management is required; one might, for instance, think of a "library" of valuable event-similarity measures. The SENACTIVE EventAnalyzer serving as a framework for our implementation allows the user choosing from all stored similarity-measures that are generally compatible with a given pattern event. Also, the user can add various meta-information to a similarity measure, such as, for instance, an informal description of its certain semantics.

5.7.2 Properties

In section 3.1.3, we have described similarity measures in the strict sense as generally favourable when calculating similarities between large amounts of entities. So, which "kind" of similarity measure is the proposed one? Obviously, as the similarity between two events is derived from attribute-level similarities, the nature of a similarity measure depends on the attribute-level similarity-measures comprised therein:

A similarity-measure as proposed in the on-hand thesis is a *strong similarity measure in the strict* sense iff

a. it does not comprise required attributes/attribute-functions,
b. the aggregation function is a linear combination of attribute-level similarities and weights, and
c. all comprised attribute-level similarity measures are (strong) similarity measures in the strict sense.

Consequently, a similarity measure is a *weak similarity measure in the strict sense* iff

a. it does not comprise required attributes/attribute-functions,
b. the aggregation function is a linear combination of attribute-level similarities and weights, and
c. all comprised attribute-level similarity measures are strong or weak similarity measure in the strict sense.

In all other cases, a similarity measure is a *similarity measure in the common sense*.

5.7.3 Complexity

Finally, given a similarity-measure sim comprising n attribute-functions/attribute-level similarity-measures, let us examine the algorithm's runtime both without and with required attributes:

5.7.3.1 Without required attributes

Let us come back to Algorithm 1 as shown in section 5.4.3, assuming that sim does not comprise required attributes: Here, as each single attribute-level similarity affects the overall similarity between two events, all attribute-functions and according attribute-level similarities must be calculated independently from eventual intermediate results. Therefore, the runtime of the proposed algorithm solely depends on the runtimes of the comprised attribute-functions and attribute-level similarity-measures. Assuming attribute-functions evaluating in constant time and letting a_{best}, a_{avg} and a_{worst} address the best-case, average-case and worst-case runtime throughout all available event-level similarity measures, a so-defined similarity measure without required attributes has the following, asymptotic runtimes:

$$T_{best}(n) = n * a_{best}$$

$$T_{avg}(n) = n * a_{avg}$$

$$T_{worst}(n) = n * a_{worst}$$

5.7.3.2 With required attributes

Per definition, the similarity between two events is 0.0 when one or more required attributes have a similarity smaller than 1.0. As we evaluate required similarities at the beginning of Algorithm 1, the "best case" is that where the first required similarity is smaller than 1.0; here, the calculation can be cancelled right after evaluating one attribute-level similarity measure:

$$T_{best}(n) = 1 * a_{best} = a_{best}$$

The average-case runtime depends on the probability p that the results of an attribute-function are *certainly equal* across pairs of events: The higher p, the more calculations can be cancelled "early" within required attributes. Given a probability p, a function $p: \mathbb{Z}^+ \rightarrow [0,1]$ calculating the probability that the proposed algorithm can be cancelled with the i^{th} required attribute in sim is defined as follows:

$$p(i) = \begin{cases} p, & i = 1 \\ \big(1 - p(i-1)\big) * p, & i > 1 \end{cases}$$

The average-case run-time of the proposed algorithm *with required attributes* is then calculated as follows:

$$T_{avg}(n) = \sum_{i=1}^{m} (p(i) * i) + \left(1 - \sum_{i=1}^{m} p(i)\right) * n$$

The worst-case runtime remains unchanged.

6 Finding similar sequences of events

Similarity searching for single events as proposed in the previous section may serve a useful tool for business analysts. In certain scenarios, however, the analyst may focus on whole sequences of events, possibly representing complex, composite actions and processes in the business environment. In the on-hand section, let us therefore present the second key part of the on-hand thesis, an approach on similarity-searching for sequences of events.

In section 6.1, we will present basic considerations on event-sequence similarity and summarize evolving requirements. In section 6.2, we will demonstrate underlying ideas with concrete example. Following from that, we will derive the basic concepts of an approach on event-sequence similarity in 6.3. In section 6.4, we will present a concrete implementation of the presented approach. Concrete *cost-function*, being an essential part of the presented approach, will be listed in section 6.5. In section 6.6, we will discuss a possible extension of the proposed approach, allowing us to calculate similarity based upon different *matching modes*. Finally, in section 6.7, we will summarize pros and cons and also discuss the properties and the computational complexity of so-defined similarity measures.

6.1 Basic considerations and evolving requirements

In section 1.3, we have listed three general demands on similarity searching in event spaces. Again, before getting serious about a possible approach, let us discuss some further aspects specific to event-sequence similarity-searching. In section 6.1.1 , we will discuss the concept of *dimensions* of event-sequence similarity. In section 6.1.2, we will differentiate between four so-called *matching modes*. Finally, in section 6.1.3, we will summarize evolving requirements.

6.1.1 The basic dimensions of event-sequence similarity

Events as defined in section 3 are complex entities, and even more so are sequences thereof. Yet, when comparing two sequences of events, one will hardly take *all* given characteristics and properties into account. Depending on the actual context and interest, one might instead focus on certain “aspects”, such as, for instance, the order or events or event-level similarities.

With the requirement of *generality* in mind, we claim that an approach on event-sequence similarity should allow handling possible “aspects” of event-sequences in a largely independent and decoupled manner. Thus, when defining a concrete similarity measure, an analyst may comprise only those aspects that he or she considers relevant. Also, we claim that an approach on event-sequence similarity should allow *weighting* selected aspects, i.e., defining their impact on the overall similarity. In the following, we will refer to those aspects that are covered by a certain approach as *dimensions* of the so-defined event-sequence similarity.

Let us now define four essential dimensions of event-sequence similarity that we consider required for any feasible approach on event-sequence similarity. For practical use cases, the reader may refer to Suntinger [49].

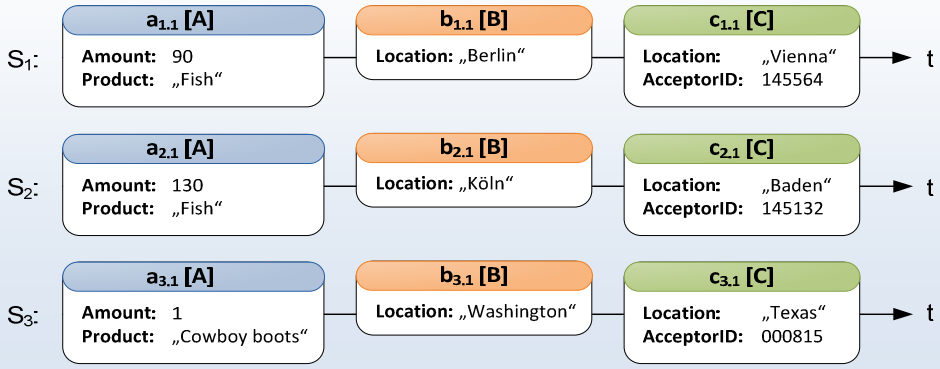
6.1.1.1 Single-event similarities

Single events are the building blocks of event-sequences, and event-attributes make event-sequences a certainly more powerful data-structure than sequences of “simple”, discrete values. Consider an event-

sequence from the logistics domain, representing a complete transport processes: Here, the nature and the quantity of the goods to be carried, the vehicle's ID, or, most important, waypoints and corresponding time stamps, are available through (and only through) event-attributes. It is easy to see that an expressive and flexible approach on event-sequence similarity should allow for taking *event-level similarities* into account.

Thus, we consider *single-event similarities* the first dimension of event-sequence similarity, and **consider two event-sequences S and T similar if S and T contain similar events.**

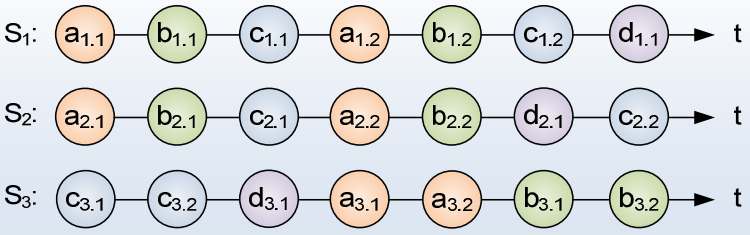
Example: Consider three event sequences S_1, S_2 and S_3 as shown below. With respect to single-event similarities, one would intuitively agree that there is a higher similarity between S_1 and S_2 than between S_1 and S_3:



6.1.1.2 Order

A dimension that is inherent to many approaches on string similarity is that of the *order*, i.e., entities are considered similar when their elements are in a similar order. Particularly useful whenever the exact temporal structure of events is of little or no relevance for the analyst, we decided to adopt the concept of *order* and consider it the second dimension of event-sequence similarity: **We consider two event sequences S and T similar if their events are in a similar order.**

Example: Consider three event sequences S_1, S_2 and S_3 as shown below. With respect to the order, one would intuitively agree that there is a higher similarity between S_1 and S_2 than between S_1 and S_3:



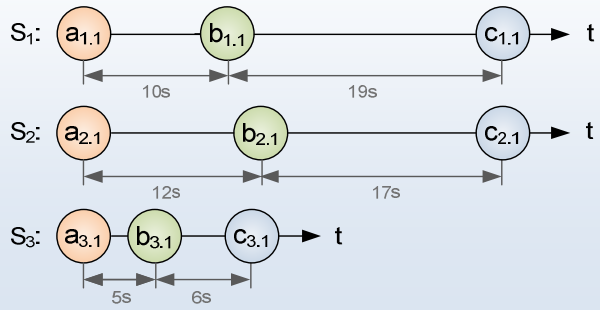
6.1.1.3 (Absolute) temporal structure

The order of events defines the general structure of an event-sequence. Yet, it is independent from the *exact* times of occurrence. In many cases, however, a business analyst may be interested in the *temporal structure* of an event sequence:

Definition: We understand the *(absolute) temporal structure* of an event-sequence S the exact time spans between the events in S, i.e., a collection of time spans $t_{1,2}, \dots, t_{|S|-1,|S|}$ with $t_{j,k} = t(e_j, e_k)$ and e_i addressing the i^{th} event in S.

We consider *absolute temporal structures* the third dimension of event-sequence similarity, and **consider two event-sequences S and T similar if their events are in a similar absolute temporal structure.**

Example: Consider three event sequences S_1, S_2 and S_3 as shown below, with labeled arrows indicating the absolute time spans between successive events. With respect to the absolute temporal structure, one would intuitively agree that there is a higher similarity between S_1 and S_2 than between S_1 and S_3:



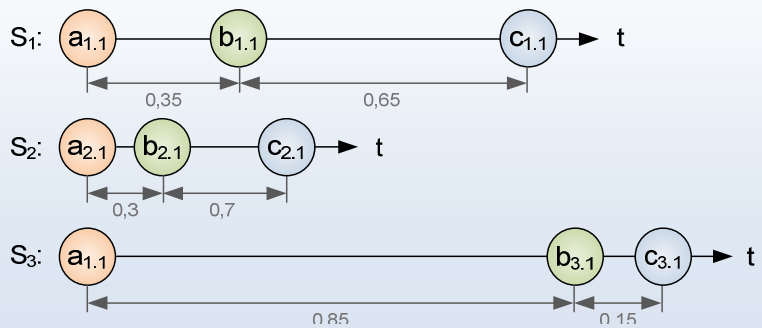
6.1.1.4 Relative temporal structure

Above, we have identified the absolute temporal structure as one possible dimension of event-sequence similarity. Scenarios may arise, though, where an analyst focuses on *relative temporal structures* rather than on absolute ones:

Definition: We understand the *relative temporal structure* of an event-sequence S the exact time spans between the events in S relative to the overall length of S, i.e., a collection $t_{r_{1,2}}, \dots, t_{r_{|S|-1,|S|}}$ with $t_{r_{j,k}} = t_r\left((e_j, e_k), S\right) = \frac{t((e_j,e_k),S)}{l(S)}$ and e_i addressing the i^{th} event in S.

We consider *relative temporal structures* the fourth and last dimension of event-sequence similarity, and **consider two event-sequences S and T similar if their events are in a similar relative temporal structure.**

Example: Consider three event sequences S_1, S_2 and S_3 as shown below, with labeled arrows indicating the relative time spans between successive events. With respect to the relative temporal structure, one would intuitively agree that there is a higher similarity between S_1 and S_2 than between S_1 and S_3:

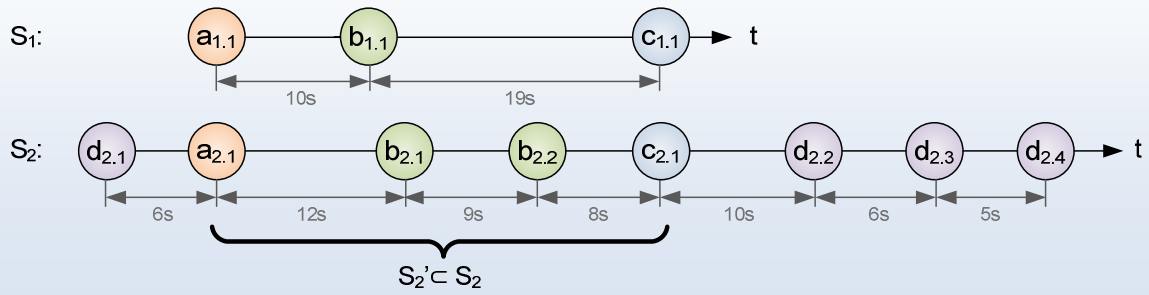


6.1.2 Full-sequence-, sub-sequence- and *-linked matching

When talking about the similarity between two sequences S and T, one naturally assumes that similarity is calculated between S and T both "as a whole" so that each single element in S and each single element in T has a however-defined impact on the overall similarity. Without a doubt, this can be considered the "regular" and very basic understanding of similarity between two sequences; throughout the following sections, we will refer to it as *full-sequence matching*.

In the context of event-sequence analysis, however, analysts may often be interested in all sequences containing a *sub-sequence* similar to a given pattern-sequence S; eventual preliminary and eventual subsequent events are considered irrelevant and are not taken into account. With this in mind, we define an alternative understanding of sequence similarity, *sub-sequence matching*, as follows: Given two sequences S and T and a regular (i.e., full sequence) similarity-measure $sim: \{S\} \times \{T\} \rightarrow [0,1]$, the sub-sequence similarity sim' of T to S can be considered the full-sequence similarity between S and the best-matching sub-sequence T'_{best} of T, i.e., $sim'(S,T) = \max_{T' \in \mathfrak{T}} sim(S,T')$ with $\mathfrak{T}$ denoting the set of all sub-sequences of T. Obviously, a similarity measure performing sub-sequence matching is always asymmetric.

Example: Consider two event-sequences S_1 and S_2 as shown below. Assuming that for a given similarity measure S_2' is the best-matching sub-sequence of S_2 with respect to S_1, the *sub-sequence similarity* between S_1 and S_2 conforms to the *full-sequence similarity* between S_1 and S_2.



Full-sequence mapping and sub-sequence matching are by far the most relevant understandings of sequence-similarity. In special cases, something "in between" *full-sequence matching* and *sub-sequence matching* may be required, though: A business analyst may be interested in sequences with a *beginning* similar to a certain pattern sequence, while preliminary events have no impact on the overall similarity. Similarly, a business analyst may be interested in sequences with an *ending* similar to the pattern. We refer to these matching-modes as *start-linked matching* and *end-linked matching* respectively. For the sake of brevity, we omit both formal definitions and examples. Again, both matching modes can be considered asymmetric in general.

It is easy to see that in general, any similarity measure that allows full-sequence matching can be used for all other matching modes as well: In accordance with the above, formal definition of sub-sequence matching, on could calculate full-sequence similarities between the pattern sequence S and any valid sub-sequence of a sequence T in a fully independent manner. In the on-hand thesis, however, we seek for an approach on event-sequence similarity that allows utilizing possible cross-sub-sequence redundancies and, thus, calculating alternative matching modes efficiently.

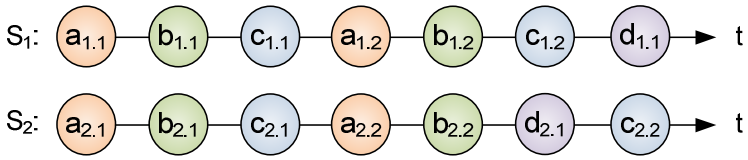
6.1.3 Requirements

From the above discussion and the preliminary notes on similarity searching, we derive the following requirements for an approach on event-sequence similarity:

1. An approach on event-sequence similarity should be generic and flexible enough to apply for all possible kinds of event-sequences and all possible business scenarios these sequences are generated in.
2. An approach on event-sequence similarity allows handling different "aspects" of event-sequences separately and generally independently from each other. Featuring *single-event similarities*, *order*, *absolute* and *relative temporal structures*, it allows the analyst choosing from the four basic dimensions of event-sequence similarity.
3. An approach on event-sequence similarity allows the efficient calculation of full-sequence-, sub-sequence-, start- and end-linked similarities.
4. A similarity measure is always defined based upon a certain interest.
5. A similarity measure is always defined under the active involvement of the domain-expert.
6. A framework for event-sequence similarity allows both defining and managing similarity-measures in a quick and straightforward fashion.

6.2 An assignment-based approach on sequence similarity

In section 6.1.1.2, we have shown two event sequences S_1 and S_2 that are apparently similar with respect to the aspect of order:



So, what *exactly* makes S_1 appear similar to S_2?

We suppose that more less unconsciously, viewers establish a set of "cognitive connections" between the elements of the compared sequences. With that assignment, each element of the pattern sequence is mapped to (at most) one distinct element of the target sequence. The target-sequence element can then be considered a *representation* of the according pattern-sequence element *in* the target sequence. Figure 12 below shows an exemplary assignment of S_2 to S_1:

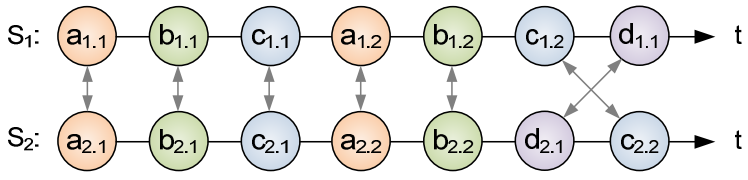


Figure 12: A possible assignment between two sequences of events

Still focusing on the aspect of order, the above assignment seems somewhat "natural".[21] Keep in mind, however, that several other assignments are possible as well. Consider, for instance, an alternative assignment of S_2 to S_1 as shown in Figure 13:

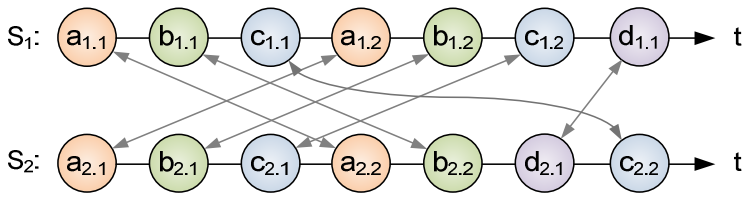
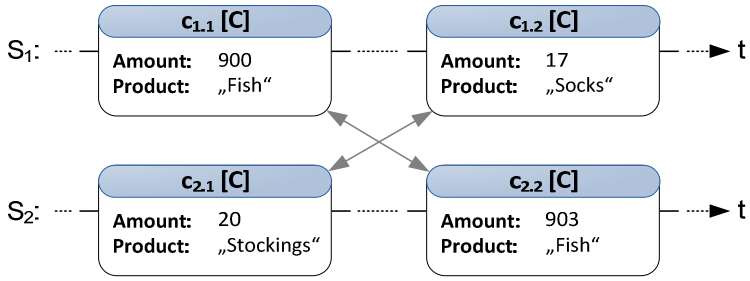


Figure 13: An alternative assignment between two sequences of events

The latter assignment is certainly less natural regarding the aspect of order; it might, however, link events with higher element-level similarities. Consider the following event attributes for $a_{1.1}$, $a_{1.2}$, $a_{2.1}$ and $a_{2.2}$:



The reader will agree that depending on the certain interest and focus, different assignments can be considered "most obvious".

As a basis for our approach on event-sequence similarity, we suppose that the overall similarity between two event sequences is derived from however-defined deviations and commonalities between the "originals and representations" in a cognitively established assignment from the target sequence to the pattern sequence. We furthermore suppose that thereby, the viewer intuitively chooses an assignment that is *optimal* with respect to the viewer's certain interest and focus.

6.3 Measuring event-sequence similarity

For our approach on event-sequence similarity, we build upon the above-presented, assignment-based idea of sequence similarity: Given a pattern sequence S_p and a target sequence S_t, we calculate the overall similarity of S_t to S_p from the quality, i.e., from the however-defined overall costs, of the best-possible assignment of S_t to S_p. The quality/costs of an assignment are thereby calculated from (and only from) the mappings defined in the certain assignment.

With the basic ideas clarified, it's time to go into some more detail: First, to avoid possible ambiguities, let us reject the (somewhat common) term "assignment" and use the term "solution" instead. Given a pattern sequence S_p and a target sequence S_t, we understand a solution a function $s: S_p \rightarrow S_t \cup \{\varepsilon\}$ that - plainly

[21] Note that linked events do not *necessarily* have to be of one and the same event type. Yet, when understanding linked events as *representations* of one another, such restriction is apparently natural. We will address the question of whether an event *e* is generally "considerable" to represent another event f in section 6.3.3. Implicitly used above, we will also define the *event-type compatibility* building upon the subtype-relationship between event-types.

spoken - maps each event of S_p either to "nowhere" ("ε") or to one distinct event of S_t. A more formal definition of solutions will be given in section 6.3.1 below.

So, how do we find the "best-possible solution" of S_t for S_p? Conceptually, we calculate n so-called *cost-factors* for any valid solution $s: S_p \rightarrow S_t \cup \{\varepsilon\}$ of S_t for S_p. These cost-factors express the solution's overall distance from an ideal solution s_0 in an n-dimensional space.[22] The exact amount and "nature" of the certain cost-factors depends on the focus of the similarity-search, and thus may vary from case to case; yet, in accordance with the general requirements presented in section 6.1, we allow the analyst choosing from four so-called *cost-functions,* deriving cost-factors from the four key dimensions of event-sequence similarity, i.e., from *single-event similarities*, *order*, and from *absolute* and *relative temporal structures*. In any case, the "best-possible" solution $s_{best}: S_p \rightarrow S_t \cup \{\varepsilon\}$ is that with lowest overall costs, i.e., with the smallest distance to s_0.

Finding the best-possible solution of a certain target sequence can indeed be considered the key part of our approach. Therefore, throughout the following sections, we will examine that part of our approach in great detail. Calculating the sequence-level similarity of S_t for S_p from the overall costs of the best-possible solution, however, is almost trivial: Here, we apply a natural transformation from distance/costs to similarities as presented in section 3.1.2. Thus, on the level of solutions, the on-hand approach can be considered *distance-based*, i.e., *geometric*.

In section 6.3.1, we will give formal definitions of the term "solution" and concepts related therewith. In section 6.3.2, we will go into more details on the assignment of costs to solutions. What makes one solution *valid* and another one *invalid* will be discussed in section 6.3.3. Finally, in section 6.3.4, we will summarize the building blocks of our approach.

6.3.1 Basic terms and concepts

Above, we have introduced *solutions* as a building block of the on-hand approach on event-sequence similarity. To a large extent, a solution can be considered an injective mapping from the events of a pattern-sequence to events of a target-sequence. Yet, to allow one or more events of the pattern sequence *not* being mapped to events of the target sequence, we decided to introduce an auxiliary unit ε, indicating that there is no "counterpart" for the certain event of the pattern sequence.[23] Formally, a solution is thus defined as follows:

> **Definition:** Given two event-sequences S_p and S_t, we refer to a function $s: S_p \rightarrow S_t \cup \{\varepsilon\}$ with $s(e) \neq s(f)\ \forall\ e, f \in \{g | g \in S_p, s(g) \neq \varepsilon\}$ as a *solution* of S_t for S_p. We refer to S_p and S_t as *pattern-sequence* and *target-sequence*, respectively. Despite being an auxiliary construct rather than an event in the strict sense, $\varepsilon \notin \mathbb{E}$, we refer to ε as the *null-event*.

A so-defined solution $s: S_p \rightarrow S_t \cup \{\varepsilon\}$ can be considered *injective* on S_t and *non-injective* on $\{\varepsilon\}$. Hence, for two event-sequences S_p and S_t,

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

[22] An "ideal" solution we understand a mapping between sequences that are equal in all relevant respects.
[23] Mappings to ε and their meaning for the on-hand approach will be discussed in more detail in section 6.3.1.2 below.

different solutions of S_t exist for S_p.[24] In the following, given a pattern sequence S_p and a target sequence S_t, let $\mathcal{S}(S_t, S_p) = \left\{ s \middle| s: S_p \to S_t \cup \{\varepsilon\}, s(e) \neq s(f)\ \forall\ e, f \in \left\{ g \middle| g \in S_p, s(g) \neq \varepsilon \right\} \right\}$ denote the set of all possible solutions of S_t for S_p.

In many cases, we will be interested in the result of a solution when "applied" to one particular event of the pattern-sequence. We therefore define the notion of *mapping*s:

Definition: Given a solution $s: S_p \to S_t \cup \{\varepsilon\}$, we refer to a pair of events $\big(e, s(e)\big)$, $e \in S_p$, $s(e) \in S_t \cup \{\varepsilon\}$, as *mapping*. Consequently, given an event $e \in S_p$, we refer to $s(e) \in S_t \cup \{\varepsilon\}$ as the *mapping for e in s*. If $s(e) = \varepsilon$, we refer to a mapping as *null-mapping*. Otherwise, if $s(e) \neq \varepsilon$, we refer to it as a *normal mapping*.

As mappings can be considered "associated" with the events of the pattern-sequence, one can easily adopt the order of pattern-sequence events (in their certain pattern-sequence) for mappings:

Definition: Given a solution $s: S_p \to S_t \cup \{\varepsilon\}$ and a mapping $\big(e, s(e)\big)$, $e \in S_p$, with $pos\big(e, S_p\big) = i$, we refer to $\big(e, s(e)\big)$ as the i^{th} *mapping in* s. Consequently, two mappings $\big(f, s(f)\big)$ and $\big(g, s(g)\big)$, $f, g \in S_p$, we refer to as *successive* if $d\big(f, g, S_p\big) = 1$.

It is easy to see that a solution $s: S_p \to S_t \cup \{\varepsilon\}$ can be defined by listing all mappings defined in s; one could, for instance, write $s = \left\{ \big(e, s(e)\big) \middle| e \in S_p \right\}$. Alternatively, if S_p is clear from the context, one can list the results of s for all pattern-sequence events, ordered in accordance with the pattern-sequence event's positions in S_p. We will use this notion in the following example, and also throughout later sections.

Example: Consider two event sequences S_p and S_t, $S_p = (e, f, g)$, $S_t = (h, i, j)$. Among others, the following solutions of S_t exist for S_p: $s_1 = (h, i, j)$, $s_2 = (i, h, j)$, $s_3 = (h, i, \varepsilon)$, $s_4 = (\varepsilon, j, \varepsilon)$, $s_5 = (\varepsilon, \varepsilon, i)$, $s_6 = (\varepsilon, \varepsilon, \varepsilon)$.

6.3.1.1 Illustration

In many cases, graphic illustration solutions may be much easier to understand than formal descriptions as shown above. Throughout the following sections, we will illustrate a solution $s: S_p \to S_t \cup \{\varepsilon\}$ by literally "linking" those events in S_p and S_t that are mapped to each other in s. In case of null-mappings, no connection is shown for the according pattern-sequence event. Figure 14 below illustrates an exemplary solution $s: S_p \to S_p \cup \{\varepsilon\}$ with $s = (i, k, \varepsilon, j)$.

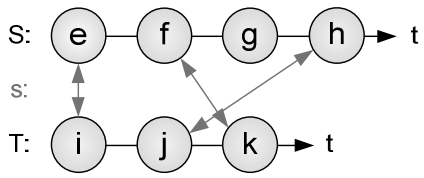


Figure 14: Illustration of an exemplary solution

[24] Not claiming mathematical rigor, this is an "m out of n"-thing without replacement of ε, with ordering.

Alternatively, if the pattern-sequence is clear from the context and the focus is on the order of target-sequence events in a solution, we will depict solutions by connecting those events of the target-sequence that are mapped to successive pattern-sequence events, i.e., to pattern-sequence events e and f, $e, f \in S$, with $d((e,f)S) = 1$. Figure 15 below shows a so-defined illustration of s:

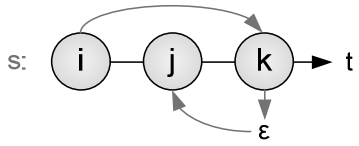


Figure 15: Reduced illustration of an exemplary solution

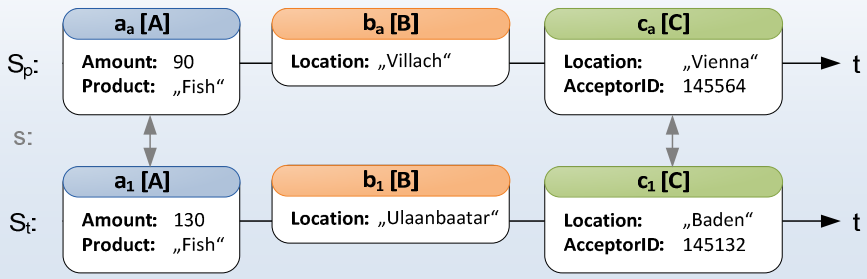
It is easy to see that null-mappings cannot be depicted in the described way. In such cases, we add an "ε"-symbol to the path as shown above.

6.3.1.2 Null-mappings

Null-mappings as defined above are may not be immediately intuitive for readers. Yet, for two reasons, they are an essential part of the on-hand approach on event-sequence similarity. On the one hand, null-mappings are required in all solutions $s: S_p \rightarrow S_t \cup \{\varepsilon\}$ where $|S_p| > |S_t|$, i.e., when there are more events in the pattern sequence than in the target sequence. Here, as a solution is injective on the target-sequence, there are no distinct counterparts in S_t for $k = |S_p| - |S_t|$ events of the pattern sequence. Thus, in a solution of S_t for S_p, at least k events of the pattern sequence must be mapped to ε. Note that with compatibilities as defined in section 6.3.3, the number of required null-mappings can further increase.

On the other hand, null-mappings may play an important role in finding the best-possible solution of a certain target-sequence: Given a pattern-sequence event e, scenarios may arise were all possible "normal" mappings for e are extremely costly with respect to the analyst's certain interest; there may, for instance, be very low event-level similarities. Here - again depending on the analyst's interest - it may be more natural not to map e at all, but instead consider it *missing* in the target sequence. Thus, when assigning costs to solutions as described in section 6.3.2, the fact that an event is missing in the target sequence may be considered less costly than available but "poor" representations.

Example: Consider two event sequences S_t and $S_p$as shown below. Obviously, as there is a very low event-level similarity between b_a and b_1, one might consider a solution $s: S_p \rightarrow S_t \cup \{\varepsilon\}$ with $s(b_a) = \varepsilon$, i.e., not defining a representation of b_a in S_t as "best-possible" with respect to the dimension of *single-event similarities*.



6.3.2 Assigning costs to solutions

We have stated that the on-hand approach on event-sequence similarity builds upon finding the best-possible solution of the certain target sequence. So, where do the costs of a solution "come from"? What makes one solution costly and another one cheap?

6.3.2.1 Assigning costs to single mappings

Naturally, costs may arise from (however-defined) relations between "corresponding" events, i.e., events that are mapped to each other. As part of a similarity-measure for sequences of events, let us therefore introduce cost-functions that apply on single mappings:

> **Definition:** Given a pattern sequence S_p, we refer to a function $c: S_p \times (\mathbb{E} \cup \{\varepsilon\}) \rightarrow {\mathbb{R}_0}^+$ as a *cost function on single mappings*. Given a solution $s: S_p \rightarrow S_t \cup \{\varepsilon\}$, we refer to the result of a cost function for a mapping $\big(e, s(e)\big)$, $e \in S_p$, as the *cost-factor of c for e in s*.

A possible cost-function on single mappings builds upon the single-event (dis-)similarity between mapped events. We will discuss this issue in section 6.4.4.

6.3.2.2 Assigning costs to pairs of mappings

The second possible source for cost might seem a little more subtle; yet, it will become clear with concrete cost-functions as described in section 6.5.2, 6.5.3 and 6.5.4: Costs may arise from (however-defined) relations between successive mappings, i.e., between mappings for pattern-sequence events that succeed each other in the pattern-sequence. Therefore, as a further part of an event-sequence similarity-measure, let us introduce cost-functions that apply on pairs of mappings.

> **Definition:** Given a pattern sequence S_p, we refer to a function $c: \Big(S_p \times (\mathbb{E} \cup \{\varepsilon\})\Big) \times \Big(S_p \times (\mathbb{E} \cup \{\varepsilon\})\Big) \rightarrow {\mathbb{R}_0}^+$ as a *cost function on pairs of mappings*. Given a solution $s: S_p \rightarrow S_t \cup \{\varepsilon\}$, we refer to the result of a cost function for a pair of mappings $\Big(\big(e, s(e)\big), \big(e, s(e)\big)\Big)$, $e, f \in S_p$, as the *cost-factor of c for e and f in s*.

6.3.2.3 Calculating the overall costs of a solution

With *cost-functions*, we have presented mechanisms for calculating cost-factors from single mappings and pairs of mappings. Now, how do we calculate the overall costs of a solution?

Consider a solution $s: S_p \rightarrow S_t \cup \{\varepsilon\}$ together with a collection of cost-functions on single mappings $c_{s_1}, c_{s_2}, \dots, c_{s_m}$ and a collection of cost-functions on pairs of mappings $c_{p_1}, c_{p_2}, \dots, c_{p_n}$. When each cost-function on single mappings is applied to each single mapping $\big(e, s(e)\big)$ in s, $e \in S_p$, and each cost-function on pairs of mappings is applied to each pair of successive mappings $\Big(\big(f, s(f)\big), \big(g, s(g)\big)\Big)$ in s, $f, g \in S_p$, $d(f, g, S_p) = 1$, $k = \Big(m * |S_p| + n * \big(|S_p| - 1\big)\Big)$ *cost-factors* $c_1, c_2, \dots, c_k$ are calculated. As these cost-factors define the relative positioning of s and an ideal solution s_0 in a k-dimensional space, overall costs can be calculated in a straightforward manner; one might, for instance, calculated the Euclidean distance between s and s_0.

Keep in mind, however, that we consider *generality* a major requirement on any valuable approach on event-sequence similarity. In addition to the various cost-functions on single mappings and pairs of mappings, we

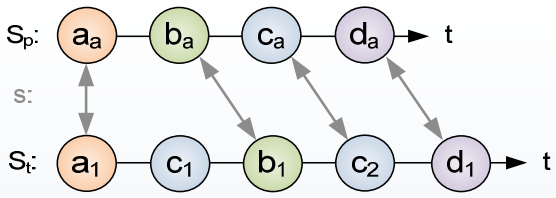
therefore let the user define a collection of weights $\omega_1, \omega_2, \dots, \omega_k$ with $\omega_j \in [0,1]\ \forall\, j = 1 \dots k$ and $\sum_{j=1}^{k} \omega_j = 1$, i.e., that sum to unity. Thereby, ω_i defines the impact of the i^{th} cost-factor c_i on the overall costs of s. Given an however defined aggregation function $f: {\mathbb{R}_0}^{+k} \times [0,1]^k \to {\mathbb{R}_0}^{+}$, a function $cost: S_p \times (S_p \to \mathbb{E} \cup \{\varepsilon\})$ calculating the overall costs of a solution $s: S_p \to S_t \cup \{\varepsilon\}$ is defined as follows:

$$cost(S_p, s) = f(c_1, \dots, c_k, \omega_1, \dots, \omega_k) =$$
$$= f\left(\bigcup_{i=1}^{n}\left(c_{s_i}(e_1, s(e_1)), \dots, c_{s_i}\left(e_{|S_p|}, s\left(e_{|S_p|}\right)\right)\right) \cup \bigcup_{i=1}^{m}\left(c_{p_i}(e_1, s(e_1), e_2, s(e_2)), \dots, c_{p_i}\left(e_{|S_p|-1}, s\left(e_{|S_p|-1}\right), e_{|S_p|}, s\left(e_{|S_p|}\right)\right)\right), (\omega_1, \dots, \omega_k)\right)$$

A function $cost_{wa}: S_p \times (S_p \to \mathbb{E} \cup \{\varepsilon\})$ calculating the overall costs based upon the *weighted average* of cost factors is thus defined as follows:[25]

$$cost_{wa}(S_p, s) = \sum_{i=1}^{m}\sum_{j=1}^{|S_p|}\left(c_{s_i}\left(e_j, s(e_j)\right) * \omega_{(i-1)*|S_p|+j}\right) + \sum_{k=1}^{n}\sum_{l=1}^{|S_p|-1}\left(c_{p_k}\left((e_l, s(e_l)), (e_{l+1}, s(e_{l+1}))\right) * \omega_{m*|S_p|+(n-1)*(|S_p|-1)+l}\right)$$

Example: Consider a solution $s: S_p \to S_t \cup \{\varepsilon\}$ as shown below.



Given two however-defined cost-functions on single mappings c_{s_1} and c_{s_2} and a however-defined cost-function on pairs of mappings c_p, 11 cost factors are calculated from s. Table 8 below lists exemplary cost-factors together with corresponding, user-defined weights. From these, overall costs are calculated based upon the *weighted average.*

	(a_a, a_1)	(b_a, b_1)	(c_a, c_2)	(d_a, d_1)	(a_a, a_1), (b_a, b_1)	(b_a, b_1), (c_a, c_2)	(c_a, c_2), (d_a, d_1)
c_{s_1}	33 / 0.1	85 / 0.05	10 / 0.05	22 / 0.1	-	-	-
c_{s_2}	60 / 0.2	10 / 0.02	4 / 0.02	1 / 0.06	-	-	-
c_p	-	-	-	-	106 / 0	121 / 0	3 / 0.4
							$cost_\omega(S_p, s) = 24,33$

Table 8: Exemplary cost-factors and weights

[25] At this point of the on-hand thesis, $cost_{wa}$ serves as valuable example. Note, however, that it will play a central role in the concrete implementation of our approach as presented in section 6.4.

6.3.3 Compatibilities and valid solutions

In section 6.3.1, we have stated that given a pattern sequence S_p and a target sequence S_t, $\sum_{k=0}^{\min(|S_p|,|S_t|)} \left(\frac{(|S_p|)!}{(|S_p|-k)!}\right) * \binom{|S_t|}{k}$ different solutions of S_t exist for S_p. Unfortunately, that's quite a lot: For a pattern sequence with $|S_p| = 10$ and a target sequence with $|S_t| = 12$, for instance, 2581284541 solutions exist.

It is easy to see that in order to find an optimal solution efficiently, one should try to restrict the set of "considerable" solutions as much as possible. So, which kind of solution is "considerable", and which is not? Keep in mind that the optimal solution should comprise those parts of the target sequence that, as a whole, fit the pattern sequence best: Thus, one might argue that some events of the target sequence can not and in no way correspond to a certain event of the pattern sequence: One might, for instance, argue that an event must be *substitutable* by its representation in an object-oriented sense, i.e., that an event of type T must be mapped to events either of T or of a subtype U of T, $T{:>}\, U$. The definition of which events are generally "compatible" to each other is, however, up to the user and certainly depends on the given context.[26]

Definition: Given a pattern sequence S_p, we refer to a function $c{:}\, S_p \times (\mathbb{E} \cup \{\varepsilon\}) \rightarrow \{0,1\}$ as *compatibility*. Two entities e and f, $e \in S_p$, $f \in \mathbb{E} \cup \{\varepsilon\}$ with $c(e,f) = 1$ we refer to as *compatible* with respect to c. Otherwise, if $c(e,f) = 0$, we refer to e and f as *incompatible* with respect to c.

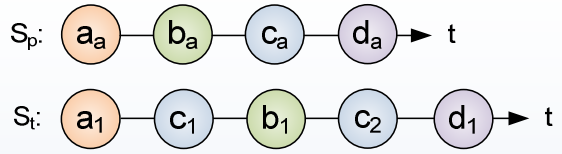
Definition: Given a solution $s{:}\, S_p \rightarrow S_t \cup \{\varepsilon\}$ and a compatibility c, we refer to a mapping $\big(e, s(e)\big)$, $e \in S_p$, as valid with respect to c if $c\big(e, s(e)\big) = 1$. Otherwise, if $c\big(e, s(e)\big) = 0$, we refer to the mapping as *invalid* with respect to c.

Definition: Given a solution $s{:}\, S_p \rightarrow S_t \cup \{\varepsilon\}$ and a compatibility c, we refer to s as *valid* with respect to c if all mappings in s are *valid*, i.e., if $c\big(e, s(e)\big) = 1\ \forall\ e \epsilon S_p$. Otherwise, if $c\big(e, s(e)\big) = 0\ \exists\ e \epsilon S_p$, we refer to s as *invalid* with respect to c.

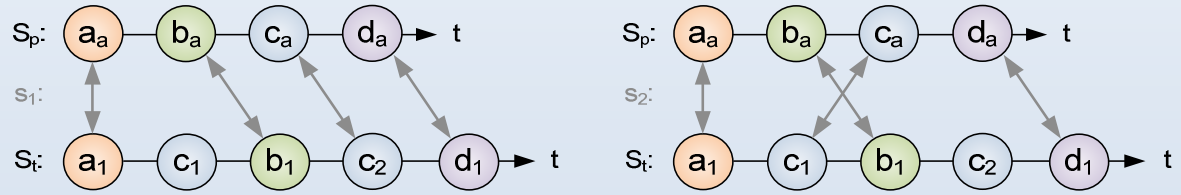
As costs can be assigned to both single mappings and pairs of mappings, it might seem natural to simply assign *infinite* costs to mappings between incompatible events; an invalid solution should then have infinite costs as well. This approach, however, fails for certain weighting configurations (zero-weights, to be exact) as discussed in section 6.3.2.3. Hence, we do not take such mappings between incompatible events into account at all: Solutions that comprise such mappings are not evaluated but instead are omitted from the start. Considering incompatible pairs of events may therefore result in a considerable reduction of "considerable" – i.e., *valid* – solutions:

[26] In most cases, the described, type-based understanding of compatibility should meet the analyst's requirements. Consider a pattern-sequence S_p and a function $c_\varepsilon{:}\, S_p \rightarrow \{0,1\}$ defining whether an event of the pattern sequence is compatible to ε, i.e., whether null-mappings are valid for a pattern-sequence event e, we refer to a compatibility $c{:}\, S_p \times (\mathbb{E} \cup \{\varepsilon\}) \rightarrow \{0,1\}$ with $c(e,f) = c_\varepsilon(e)$ if $f = \varepsilon$ and $c(e,f) = \begin{cases} 1, & typeof(e){:>}\, typeof(f) \\ 0, & otherwise \end{cases}$ otherwise, as *event-type compatibility*. Unless otherwise stated, we will presume *event-type compatibility* to apply throughout the following sections and examples.

Example: Consider two event sequences S_p and S_t as shown below.



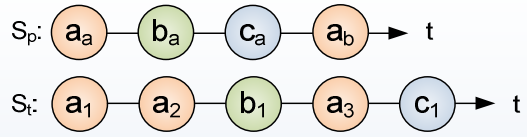
Without any restrictions on event compatibility, 501 solutions of S_t for S_p exist (for obvious reasons, we omit depicting them in detail). On the understanding, however, that a.) all null-mappings are considered invalid and b.) two events are compatible iff they are of the same event type, only two solutions of S_t for S_p, s_1 and s_2, are considered valid:



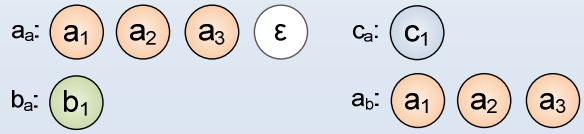
In real-world scenarios, it is usually more efficient to list those mappings that are *valid* rather than those that are not. We therefore introduce the notion of *matches*:

Definition: Given a pattern sequence S_p, a target sequence S_t, two entities e and f with $e \in S_p$ and $f \in S_t \cup \{\varepsilon\}$ and a compatibility c, we refer to f as a *match* for e with respect to c if e is compatible to f, i.e., if $c(e,f) = 1$. The set M of target-sequence events that are compatible to e, $M = \{g | g \in S_t \cup \{\varepsilon\}, c(e,g) = 1\}$, we refer to as *matches for e in S_t with respect to c*.

Example: Consider the following sequences S_p and S_t:



On the understanding, however, that a.) null-mappings are considered valid for a_a and invalid for all other events of the pattern sequence and b.) two events are compatible iff they are of the same event type, the following *matches* exist for the given pattern-sequence events:



6.3.4 Summary

Before presenting a concrete implementation, let us recapitulate the presented approach on event-sequence similarity: Given a pattern-sequence S_p, configuring a similarity-measure on sequences of events can be considered defining a 6-tuple

$$\langle C_s, C_p, W, f, c, m \rangle,$$

where

- C_s is a collection of cost-functions on single mappings $c_{s_1}, c_{s_2}, \dots, c_{s_n}$ with $c_{s_i}: S_p \times (\mathbb{E} \cup \{\varepsilon\}) \to {\mathbb{R}_0}^+ \; \forall \; i = 1 \dots n$,
- C_p is a collection of cost-functions on pairs of mappings $c_{p_1}, c_{p_2}, \dots, c_{p_m}$ with $c_{p_i}: \left(S_p \times (\mathbb{E} \cup \{\varepsilon\})\right) \times \left(S_p \times (\mathbb{E} \cup \{\varepsilon\})\right) \to {\mathbb{R}_0}^+ \; \forall \; i = 1 \dots m$,
- W is a collection of weights $\omega_1, \omega_2, \dots, \omega_{m*|S_p|+n*(|S_p|-1)}$ with $\omega_i \in [0,1] \; \forall \; i = 1 \dots m * |S_p| + n * \left(|S_p| - 1\right)$ and $\sum_{j=1}^{m*|S_p|+n*(|S_p|-1)} \omega_i = 1$; each defining the impact of the corresponding cost-factor on the overall costs of solutions for S_p,
- f is an aggregation function $f: {\mathbb{R}_0}^{+^{m*|S_p|+n*(|S_p|-1)}} \times [0,1]^{m*|S_p|+n*(|S_p|-1)} \to {\mathbb{R}_0}^+$ that allows calculating the overall costs of a solution from cost-factors and weights, and
- c is a compatibility $c: S_p \times (\mathbb{E} \cup \{\varepsilon\}) \to \{0,1\}$ defining the set of valid solutions of the certain target sequence.
- m is a matching mode; *full-sequence matching*, *sub-sequence matching*, *start-linked matching*, or *end-linked matching*.

The similarity between S_p and a target sequence S_t is then calculated from the overall costs of the best-possible valid solution $s: S_p \to S_t \cup \{\varepsilon\}$.

6.4 The base algorithm

Thus far, we have presented a solution-based approach on event-sequence similarity in a general and algorithm-independent manner. In the on-hand section, we will propose a possible implementation of the presented approach, building upon *Dynamic Programming*. As the basic ideas of our approach are strictly decoupled from concrete cost-functions, the presented approach can be considered a "structural framework" that may be extended by actual cost-function implementations. We will describe these parts of the algorithm in section 6.4.4.

In section 6.4.1, we will present an intuitive, tree-based approach on finding all valid solutions with respect to a given compatibility. From this, we will derive the actual algorithm in section 6.4.2. In section 6.4.3, we will address the issue of performance by introducing a Branch & Bound strategy.

6.4.1 Finding all valid solutions

In the above sections, we have introduced the concept of *compatibility*, allowing us a notable reduction of relevant solutions: A certain solution is considered only if all comprised mappings are *valid*, i.e., if the mapped unit *matches* the certain pattern-sequence event. Yet, we have not addressed the set of solutions that

"remain" with respect to a certain set of matches. With the upcoming steps of our approach on event-sequence similarity in mind, we calculate the set valid solutions with a tree-based algorithm:

To a root node, we add nodes representing all matches for the first pattern-sequence event. To each of these nodes, we add nodes representing all matches for the second event of the pattern sequence, and so on. The levels of the tree therefore "correspond" to the events of the pattern sequence. Yet, to ensure *injectivity* on the target sequence, a node representing a target-sequence event that is already part of a certain path (from the root node to a certain leaf) is *not* added to this path again.[27] Thus, at the end of the algorithm, the set of paths from the root node to the leafs of the tree represents the overall set of solutions that are valid with respect to the given compatibilities.

In pseudo code, the algorithm can be described as follows:

Name:	**createSolutionsTree**	
Description:	Creates a tree representing the set of valid solutions. If no such solution exists, the tree remains incomplete. With *root* being a tree node, the execution of the recursive algorithm is initiated by calling *createSolutionsTree(root, 1).*	
Input:	**node:** **index:**	The parent node. The current level of the tree.
Output:	-	
Variables:	**i:** **match:** **child:**	An index. A match at the current level of the tree. A tree node representing the current match.
State:	**matches:**	A field containing set of matches in the order of the corresponding pattern-sequence events

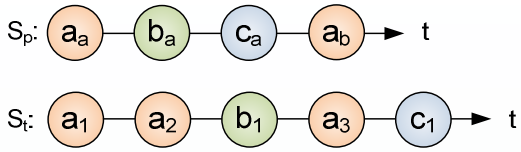
```
01:     // Iterate through the matches for the corresponding pattern-sequence events
02:     for i = 1 to matches[index].length step 1
03:             Event match = matches[index][i];
04:
05:             // Check whether event is already part of the so-far path
06:             if ((match ≠ ε) and (parent.isInPathToRoot(match)))  then
07:                     continue;
08:             end
09:
10:             // Create child node and add to parent
11:             TreeNode child = new TreeNode(match);
12:             parent.add(child);
13:
14:             // Do recursive method call
15:             if (index < matches.length) then
16:                     createSolutionsTree(child, index + 1, matches)
17:             end
18:     end
```

Algorithm 2: Calculating valid solutions from sets of matches

[27] Note that such validation is not necessary for nodes representing null-matches.

The below example depicts the algorithm's result for certain event-sequences and compatibilities. Here and in all further examples and figures, we will depict tree nodes in the same way as the represented target-sequence events. Null-matches will be depicted as white circles with a gray border. Also, the "levels" or the tree will be separated by dashed lines, with labels referring to the corresponding pattern-sequence events.

Example: Consider two event sequences S_t and S_p as shown below:



On the understanding that a.) events of the same event type are compatible to each other, and b.) null-mappings are invalid, the following tree is generated with the above algorithm. Note that the created solutions (i.e., paths from the root node to the leaf nodes) do not contain "duplicate" nodes, i.e., mappings that comprise one and the same target-sequence event:

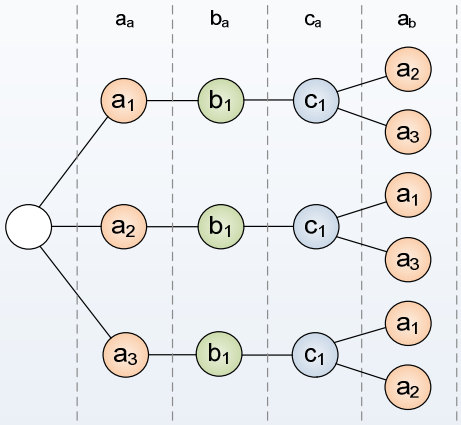


Figure 16: Exemplary results of Algorithm 2

Based upon the above definitions, six distinct solutions of S_t for S_p exist. When null-mappings are assessed to be valid for all pattern-sequence events, however, 52 solutions exist. Figure 17 shows parts of the corresponding tree:

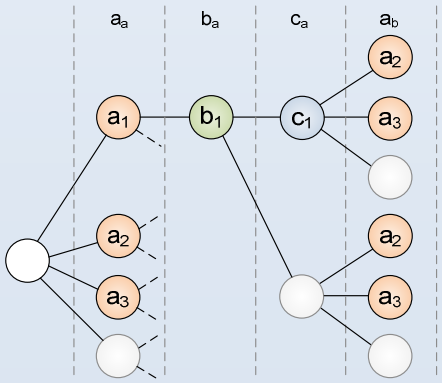


Figure 17: Exemplary result of Algorithm 2, including null-mappings

6.4.2 Calculating the overall costs of solutions

In section 6.3, we have introduced the overall costs of a solution as a combination of weights and corresponding cost factors, calculated from both single mappings and pairs of mappings. Given a pair of sequences, cost functions and weights, one could, of course, calculate the overall costs of all valid solutions in a separate and fully independent manner. Before doing so, let us come back to the tree-based algorithm above: When using a linear combination of cost-factors and weights as an aggregation function, calculating costs along the branches of such tree gives us a notable reduction of redundant calculations:

Consider, for instance, the *weighted average,* a natural and highly valuable aggregation function. Given a pattern sequence S_p, cost functions on single mappings $c_{s_1}, c_{s_2}, \dots, c_{s_m}$, cost functions on pairs of mappings $c_{p_1}, c_{p_2}, \dots, c_{p_n}$ and weights $\omega_1, \omega_2, \dots, \omega_{m*|S_p|+n*(|S_p|-1)}$, a function $cost_{wa}: S_p \times (S_p \to \mathbb{E} \cup \{\varepsilon\})$ calculating the overall costs of a solution $s: S_p \to S_t \cup \{\varepsilon\}$ from weighted average of cost factors is defined as follows:

$$cost_{wa}(S_p s) = \sum_{i=1}^{m} \sum_{j=1}^{|S_p|} \left(c_{s_i}\left(e_j, s(e_j)\right) * w_{(i-1)*|S_p|+j} \right) + \sum_{k=1}^{n} \sum_{l=1}^{|S_p|-1} \left(c_{p_k}\left(\left(e_l, s(e_l)\right), \left(e_{l+1}, s(e_{l+1})\right) \right) * w_{m*|S_p|+(n-1)*(|S_p|-1)+l} \right)$$

Hereby, note that

$$\sum_{i=1}^{m} \sum_{j=1}^{o} \left(c_{s_i}\left(e_j, s(e_j)\right) * w_{(i-1)*|S_p|+j} \right)$$

and

$$\sum_{i=1}^{n} \sum_{j=1}^{o-1} \left(c_{p_i}\left(e_j, s(e_j), e_{j+1}, s(e_{j+1})\right) * w_{m*|S_p|+(n-1)*(|S_p|-1)+j} \right)$$

solely depend on the first *o* mappings comprised in a certain solution $s: S_p \to S_t \cup \{\varepsilon\}$; these values are equal for all solutions that have the first k mappings in common. Therefore, when calculating a tree as shown in Algorithm 2, the overall costs of solutions can be calculated stepwise along paths from the root node to the various leafs.

We thus extend the above algorithm, letting each tree node hold the sum of costs that are associated to the (single and pairs of) mappings represented by itself and its predecessors in a property *sum*. It is easy to see that for each node, *sum* can be calculated by summarizing

a. the weighted cost factors calculated from the represented, i^{th} mapping,
b. for $i > 1$, the weighted cost factors calculated from the pair of mappings that is constituted by the i^{th} and the $(i-1)^{\text{th}}$ mapping, and
c. for $i > 1$, *sum* of the node's direct predecessor.

In pseudo-code, the resulting algorithm can be described as follows:

Name:	createSolutionsTree2	
Description:	Creates a tree representing the set of valid solutions; if no such solution exists, the tree remains incomplete. Each node has a property *sum* holding the so-far costs of the represented solution(s). With *root* being a tree node, the execution of the recursive algorithm is initiated by calling *createSolutionsTree2(root, 1).*	
Input:	**node:** **index:**	The parent node. The current level of the tree.
Output:	-	
State:	**pattern:** **matches:** **cfSingle:** **cfPairs:** **weights:**	A field of events representing the pattern sequence. A field containing set of matches in the order of the corresponding pattern-sequence events. A field of cost functions on single mappings. A field of cost functions of pairs of mappings. A field of weights.
Variables:	**i:** **match:** **child:** **cs:** **cp:**	An index. A match at the current level of the tree. A tree node representing the current match. Weighted cost factors for single mappings. Weighted cost factors for pairs of mappings.

```
01:  Event lastPEvent = pattern[index - 1]; Event pEvent = pattern[index];
02:
03:  // Iterate through the matches for the corresponding pattern-sequence events
04:  for i = 1 to matches[index].length step 1
05:      Event match = matches[index][i];
06:
07:      // Check whether event is already part of the so-far path; see Algorithm 2
08:      ...
09:
10:      // Calculate weighted cost factors for single mappings and pairs of mappings
11:      double cs = 1;  double cp = 0;
12:      for j = 1 to cfSingle.length step 1
13:          cs += cfSingle[i](patternEvent, match) * getCorrespondingWeight(i, j);
14:      end
15:      if (index > 1) then
16:          for j = 1 to cfPairs.length step 1
17:              cs += cfPairs[i](pEvent, match, lastPEvent, parent.Match) *
18:                  getCorrespondingWeight(i, j);
19:          end
20:      end
21:
22:      // Create child node and add to parent
23:      TreeNode child = new TreeNode(match);
24:      parent.add(child);
25:      // Set "so far" costs to child node
26:      child.Sum = parent.Sum + cs + cp;
27:
28:      // Do recursive method call
29:      if (index < matches.length) then
30:          createSolutionsTree2(child, index + 1);
31:      end
32:  end
```

Algorithm 3: Calculating the overall costs of valid solutions

In the following, we will depict the algorithm's intermediate results by listing the variables cs, cp and sum right below the corresponding tree-node, or, for cp, right below the connection between a pair of tree-nodes.

Example, continued: Let us come back to the example presented in section 6.4.1, and apply the extended version of the original algorithm. Therefore, consider two cost functions, c_s (on single mappings) and c_p (on pairs of mappings). The resulting, weighted cost-factors are listed below:

	(a_a, a_1)	(a_a, a_2)	(a_a, a_3)	(a_b, a_1)	(a_b, a_2)	(a_b, a_3)	(a_a, b_1)	(c_a, c_1)
c_s	4	100	3	8	12	6	6	5

Table 9: Weighted cost factors as resulting from c_s

	(a_a, a_1), (b_a, b_1)	(a_a, a_2), (b_a, b_1)	(a_a, a_3), (a_a, a_1)	(b_a, b_1), (c_a, c_1)	(c_a, c_1), (a_b, a_1)	(c_a, c_1), (a_b, a_2)	(c_a, c_1), (a_b, a_3)
c_p	4	7	6	8	2	6	6

Table 10: Weighted cost factors as resulting from c_p

Figure 18 below shows the algorithm's results. It is easy to see that for the given configuration, $s_3 = (a_3, b_1, c_1, a_1)$ is the best-possible solution of S_t for S_p.

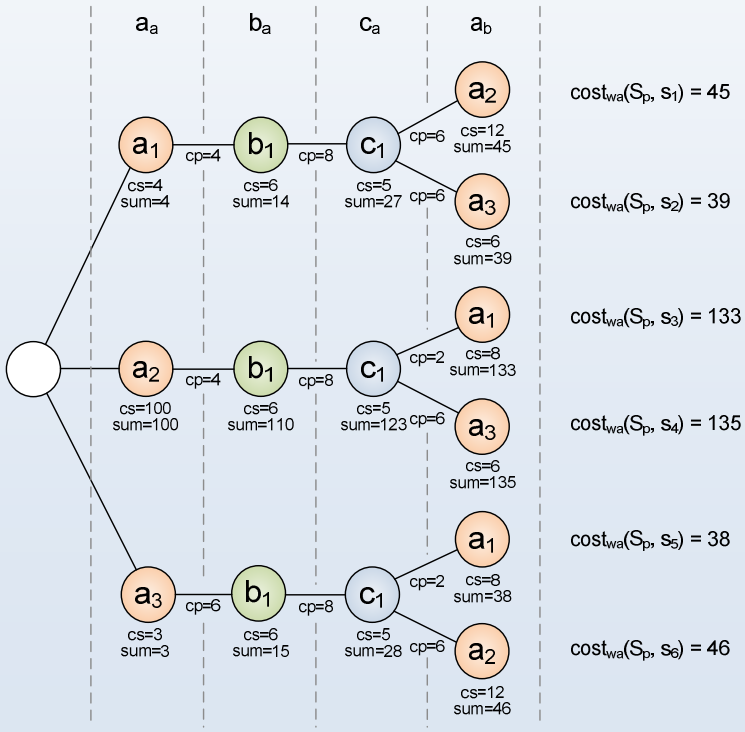


Figure 18: Calculating overall costs of solutions with Algorithm 3

6.4.3 Branch & Bound

The above algorithm calculates the overall costs for all valid solutions. Keep in mind, however, that with respect to our approach on event-sequence similarity, we are not interested in *all* possible solutions, but only in the best one, i.e., in the "cheapest" one.

Consider the following intermediate step in calculating the tree from section 6.4.2:

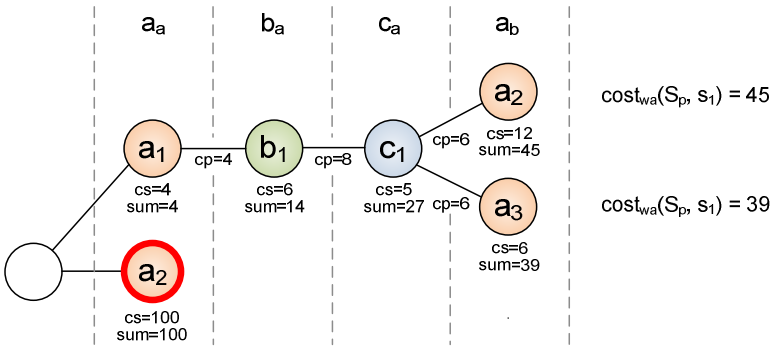


Figure 19: Intermediate results of Algorithm 3

Here, as the marked node is associated with very high costs, it is already clear that not even in the *best case*, i.e., with matches for b_a and c_a with costs of 0, overall costs lower than the best solution so far will be achieved proceeding from a_2. From this, it follows that *no* solution (i.e., no leaf) proceeding from a_2 will affect the overall similarity between S_p and S_t: The execution of paths proceeding from a_2 can hence be omitted without affecting the optimal solution.

It is easy to see that at this point, a *Branch-&-Bound* strategy is valuable.[28] We thus extend the current approach by a dynamic threshold t, being initialized with a (user-defined) value $t_{initial}$. A node is added to the tree only if the best-case similarity proceeding from it is higher than the actual threshold. Hence, if an end node is reached, the resulting overall costs c of the certain solution s is certainly lower than t; we then update t and set $t = c$. If $t = t_{initial}$ at the end of the algorithm, no solution with costs lower than the initial threshold was found; we assume an event-sequence similarity of 0. Otherwise, t holds the costs of the optimal solution.

It is easy to see that in order to chose an adequate value for $t_{initial}$, one must balance performance and accuracy: The lower $t_{initial}$, the more solutions can be excluded early in the calculation. Yet, for all solutions with overall costs higher than $t_{initial}$, information about the *exact* costs (i.e., the exact similarity value) is lost.

In pseudo-code, the resulting algorithm can be described as follows:

[28] At this point, experienced readers might suggest an alternative and certainly faster approach building upon *Dynamic-Programming.* We will discuss this issue in section 6.7.3.1 below.

Name:	createSolutionsTree3	
Description:	Calculates the overall costs of the best-possible solution for a given set of matches. If no valid solution exists with overall costs below the initial threshold, the threshold remains its initial value; otherwise, threshold holds the best-possible costs. With *root* being a tree node, the execution of the recursive algorithm is initiated by calling *createSolutionsTree3(root, 1).*	
Input:	**node:** **index:**	The parent node. The current level of the tree.
Output:	-	
Variables:	**lastPEvent:** **pEvent:** **i:** **match:** **child:** **cs:** **cp:**	The previous pattern-sequence event. The current pattern-sequence event. An index. A match at the current level of the tree. A tree node representing the current match. Weighted cost factors for single mappings. Weighted cost factors for pairs of mappings.
State:	**pattern:** **matches:** **cfSingle:** **cfPairs:** **weights:** **threshold:**	A field of events representing the pattern sequence. A field containing sets of matches in the order of the corresponding pattern-sequence events. A field of cost functions on single mappings. A field of cost functions of pairs of mappings. A field of weights. The current threshold; initialized with a used-defined value $t_{initial}$

```
01:     Event lastPEvent = pattern[index - 1]; Event pEvent = pattern[index];
02:
03:     // Iterate through the matches for the corresponding pattern-sequence events
04:     for i = 1 to matches[index].length step 1
05:         Event match = matches[index][i];
06:
07:         // Check whether event is already part of the so-far path; see Algorithm 2
08:         ...
09:
10:         // Calculate weighted cost factors for single mappings and pairs of mappings;
11:         // see  Algorithm 3
12:         ...
13:
14:         // Check whether so-far costs are below the current threshold
15:         if (parent.Sum + cs + cp < threshold) then
16:             // Create child node and add to parent
17:             TreeNode child = new TreeNode(match);
18:             parent.add(child);
19:             // Set "so far" costs to child node
20:             child.Sum = parent.Sum + cs + cp;
21:
22:             // Do recursive method call or set threshold if a leaf is reached
23:             if (index < matches.length) then
24:                 createSolutionsTree3(child, index + 1);
25:             else
26:                 threshold = child.Sum;
27:             end
28:         end
29:     end
```

Algorithm 4: Finding the best-possible solution with a user-defined threshold

Example, continued: Let us repeat the above example, but use a threshold in order to improve the calculation's performance. Based upon our certain interest, we choose an initial threshold of 40. For sub-nodes of a common predecessor, the vertical order depicts the order of creation. Nodes that exceed the threshold are marked red.

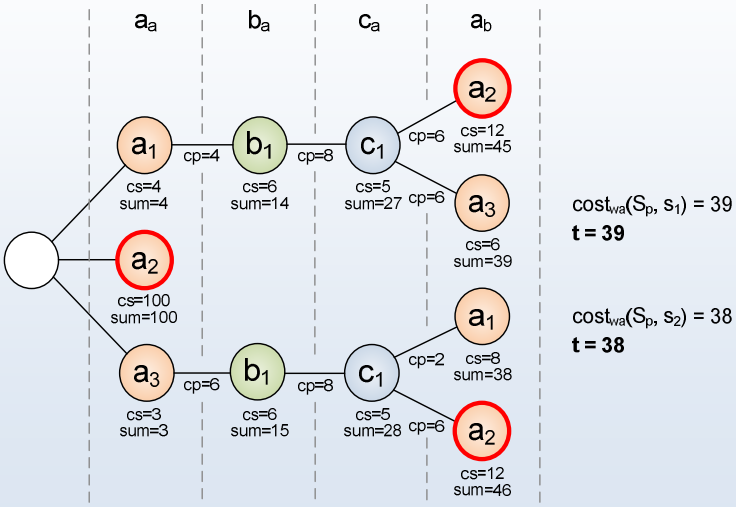


Figure 20: Exemplary results of

6.4.4 A restriction to sub-sequence matching

Note that given a pattern sequence S_p and a target sequence S_t, the set $\mathcal{S}(S_p, S_t)$ of all possible solutions of S_t for S_p comprises all possible solutions of all sub-sequences $\mathcal{P}(S_t)$ of S_t, i.e., $\mathcal{S}(S_p, S_t) \supseteq \mathcal{S}(S_p, S_t') \; \forall \; S_t' \in \mathcal{P}(S_t)$. Also, note that the above, tree-based algorithm finds the best-possible solution from *all* (valid) solutions of a given target sequence S_t. As a consequence, the so-calculated, best-possible solution of S_t can be considered *best-possible* not only across S_t but also across all sub-sequences $\mathcal{P}(S_t)$ of S_t. Not immediately intuitive, a similarity-measure building upon overall costs as resulting from the above algorithm therefore implements *sub-sequence matching* as defined in section 6.1.2.

From all matching-modes, one may usually consider *sub-sequence matching* the most complex one and most difficult to calculate. "Deriving" other matching modes from the present algorithm is not trivial, though. We will present a possible extension, allowing to perform both full-sequence matching and *-linked matching, in section 6.6. Yet, as building upon the concrete cost-function implementations from section 6.5, the given approach cannot be considered "fully general" with respect to the basic ideas presented thus far.

6.5 Cost functions

Thus far, we have clarified both the underlying concepts and the basic implementation of our approach on event-sequence similarity. It's now the time, however, to put some "flesh on the bones": In the on-hand section, we will discuss four concrete cost-functions in accordance with the four essential dimensions of event-sequence similarity as presented in section 6.1.1, i.e., for *single-event similarities* (6.5.1), *order* (6.5.2), and *absolute* (6.5.3) and *relative temporal structures* (6.5.4). Finally, in section 6.5.5, we will show that with the proposed cost-functions, a so-defined similarity measure on sequences of events could be defined "per event-sequence signature".

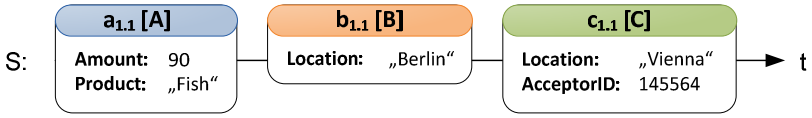
6.5.1 Cost-function A: Single-event similarities

In a first step, let us define a cost-function c_{sim} in accordance with the dimension of *single-event similarities*, so that low cost-factors are calculated for sequences that contain similar events and high cost-factors otherwise. As likely based upon single-event similarities as discussed in section c, calculating costs for "normal" pairs of events is almost trivial. Null-mappings, however, require exceptional handling. The according strategy described in section 6.5.1.2 is simple yet serves as a basis for further, certainly more complex cost-functions.

6.5.1.1 From event-level similarities to costs

When single-event similarities shall serve as the criterion of whether two event sequences are similar to each other, the costs of a single mapping should inversely relate to the event-level similarity between pairs of mapped events. In other words, a mapping comprising events that have a high similarity value should have low costs, and vice versa.

Consider an exemplary pattern sequence S_p as shown below:



It is easy to see that for mappings comprising the pattern sequence's *A*-event, the assessment of event-similarity will be based upon different criteria than for mappings comprising, for instance, the *B*-event – if only because of typing issues. For highest expressiveness, we therefore let the user define a collection of event-level similarity-measures $sim_1, sim_2, \dots, sim_{|S_p|}$ with $sim_i: \mathbb{E} \times \mathbb{E} \to [0,1] \; \forall \; i = 1 \dots |S_p|$, where sim_i reflects the specific semantics of the i^{th} event of the pattern sequence. Given a solution $s: S_p \to S_t \cup \{\varepsilon\}$, the event-similarity for a mapping $(e, s(e))$, with e being the j^{th} event of S_p is then calculated as $sim_j(e, f)$.

The proposed approach on event-sequence similarity, however, requires a single cost-function on single mappings instead of similarity measures. Luckily enough, we have discussed a natural transformation between distance/costs and similarities in section 3.1.2: Given a set of entities A and a similarity-measure $sim: A \times A \to [0,1]$, a corresponding distance function $d: A \times A \to \mathbb{R}_0^+$ is defined as follows:

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

Thus, given a pattern sequence S_p and a set of similarity measures $sim_1, \dots, sim_{|S_p|}$ as defined above, a function $c_{sim}{}': S_p \to \mathbb{E}$ introducing the costs of single-event dissimilarities is defined as follows:

$$c_{sim}{}'(e, f) = -\ln\left(sim_{pos(e,S_p)}(e, f)\right)$$

It is easy to see that the results of the above function strongly depend on the set of event-level similarity-measures. Together with weights, these similarity-measures therefore serve as a major configuration option for all those event-sequence similarity-measures that take event-level similarities into account. Note, however, that the described cost function does not necessarily depend on single-event similarity-measures as proposed in section 5, but can instead build upon any similarity-measure that meets the specified requirements.

6.5.1.2 Null-mappings

The above function returns valid costs for "normal" mappings, i.e., for mappings that link a certain event of the target sequence to a certain event of the pattern sequence. Anyway, depending on whether null-mappings are considered valid or not, events of the pattern sequence may also be linked to ε. Obviously, an implementation of the above formula will fail on such input.

In addition to event-level similarity measures, we therefore let the user define a collection of "explicit" cost-factors for null-mappings $c_{\varepsilon_1} c_{\varepsilon_2}, \dots, c_{\varepsilon_{|S_p|}}$ with $c_{\varepsilon_i} \in \mathbb{R}_0{}^+ \; \forall \; 0 < i \leq |S_p|$.[29] Thereby, c_{ε_j} defines the costs of mappings from the j^{th} event of the pattern sequence to ε. In case of null-mappings, the cost function returns the certain explicit cost-factor instead of fruitlessly applying the original formula. An extended and thus "full-featured" version of c_{sim}', $c_{sim}: S_p \rightarrow \mathbb{E} \cup \{\varepsilon\}$, is therefore defined as follows:

$$c_{sim}(e, f) = \begin{cases} sim_{pos(e,S_p)}(e, f), & f = \varepsilon \\ c_{\varepsilon_{pos(e,S_p)}}, & f \neq \varepsilon \end{cases}$$

So, which costs are suitable in case of null-mappings? Once again, let us repeat the meaning of *solutions* with respect to the on-hand approach on event-similarity: A solution $s: S_p \rightarrow S_t \cup \{\varepsilon\}$ defines a certain "representation" of a pattern-sequence S_p in a target-sequence S_t. Hence, the "cheapest" solution defines those events of S_t that constitute the best-possible representation of S_p. In case of null-mappings, a solution s does not define "counterparts" for certain events of the pattern sequence; in other words, these pattern-sequence events are *missing* in s. At a certain degree of dissimilarity between pattern-sequence events and matched target-sequence events, however, s may be considered better than a solution s' not comprising null-mappings. Therefore, in order to find adequate costs in case of null-mappings, one can ask the following question:

> *At which degree of dissimilarity between a pattern-sequence event e and a target sequence event f is it more natural to prefer a null-mapping over a mapping to f?*

From that threshold, a cost-factor can be calculated with the above transformation from similarities to costs.

[29] In practice, cost factors are only required for those null-mappings that are considered value based upon the given compatibility.

6.5.1.3 Pseudo code

In pseudo code, an implementation of the proposed cost-function can be described as follows:

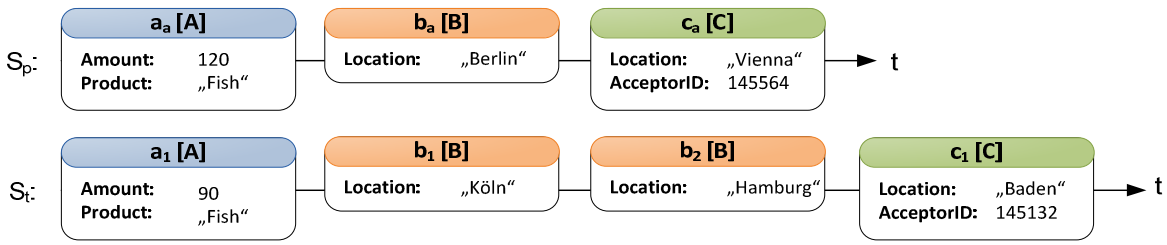
Name:	**getCostsA**	
Description:	Calculates a cost-factor based upon single-event similarities.	
Input:	**pEvent:**	An event of the pattern sequence.
	tEvent:	The corresponding event of the target sequence, or ε.
	simMeasures:	A map linking the events of the pattern sequence to their user-defined similarity measures.
	nullMappingCosts:	A map linking the events of the pattern sequence to the corresponding cost-factors for null-mappings.
Output:	A cost-factor from $\mathbb{R}_0{}^+$.	
Variables:	**similarity:**	The similarity between two events.
	distance:	The distance calculated from **similarity**.

```
01:     // Check if "normal mapping" or "null mapping"
02:     if (tEvent ≠ ε) then
03:         // Calculate similarity and distance
04:         double similarity = simMeasures[pEvent](pEvent, tEvent);
05:         double distance = -ln(similarity);
06:         return distance;
07:     else
08:         // Return corresponding cost-factor for null-mappings
09:         return nullMappingCosts[pEvent];
10:     end
```

Algorithm 5: A cost-function for the aspect of single-event similarities

6.5.1.4 Example

Consider two event sequences S_p and S_t as shown below:



Using S_p as a pattern-sequence, the business analyst defines single-event similarity-measures for the events in S_p as shown in Table 11. Also, as null-mappings will be considered valid for $a_a \in S_p$, he or she defines a cost-factor for null-mappings of a_a, $c_{\varepsilon_1} = 0.8$.

$e \in S_p$	Event-level sim. measure	Attribute	Attribute-level sim. measure	Weight
a_a	sim_1	**Fish**	Edit Distance	*Required*
		Amount	Normalized absolute difference (min: 0, max:120)	1,0
b_a	sim_2	**Location**	Lookup table similarity Berlin – Hamburg ➔ 0.7 Berlin – Köln ➔ 0.6	1,0
c_a	sim_3	**Location**	Lookup table similarity Wien – Baden ➔ 0.9	1,0

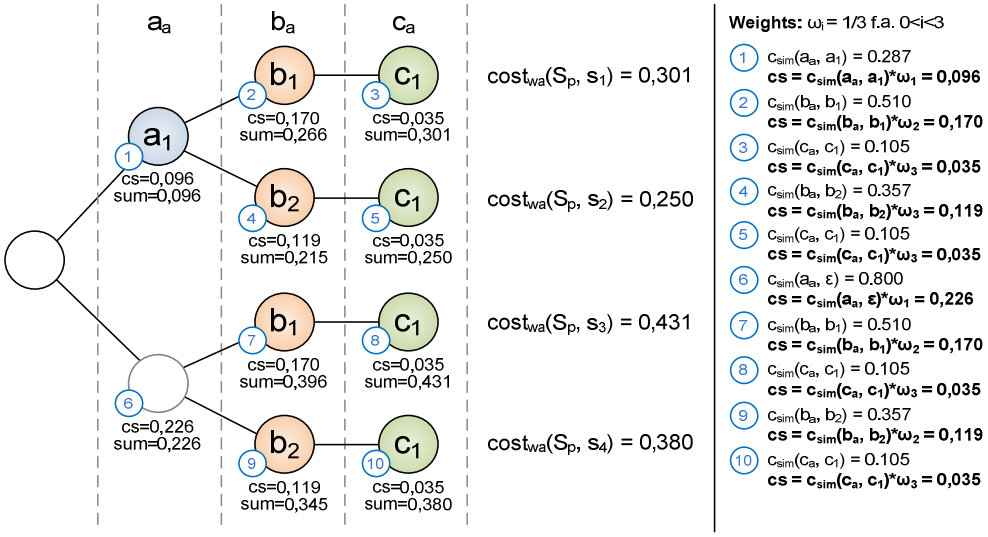
Table 11: Exemplary single-event similarity-measures

Thus, with c_{sim} as defined above, the following cost-factors are calculated:

$e \in S_p$	$f \in S_p \cup \{\varepsilon\}$	$sim_{pos(e,S_p)}(e,f)$	$c_{sim}(e,f)$
a_a	a_1	0.75	0.287
a_a	ε	-	0.8
b_a	b_1	0.6	0.510
b_a	b_2	0.7	0.357
c_a	c_1	0.9	0.105

Table 12: Exemplary cost-factors as calculated with c_{sim}

Assuming a cost-function c_{sim} as defined above, an event-type compatibility that allows null-mappings for a_a and uniformly distributed weights, we calculate the overall costs of a best-possible solution $s_{best}: S_p \rightarrow S_t \cup \{\varepsilon\}$ as follows:[30]



Therefore, with respect to the given configuration, the best-possible solution of $s_2: S_p \rightarrow S_t \cup \{\varepsilon\}$ has overall costs of 0.250.

[30] Certainly outside the scope of the on-hand example, we will not use a threshold as described in section 6.4.3. This will apply to all further cost-function examples as well.

6.5.2 Cost-function B: Order

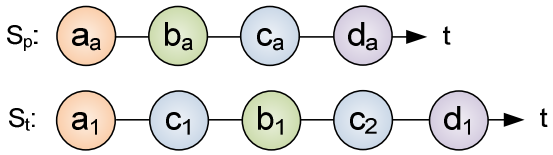
As a second cost-function, let us define c_{order} in accordance with the dimension of *order*, so that low cost-factors are calculated for sequences whose events are in a similar order, and high cost-factors otherwise. Now, what makes a solution costly with respect to the order? How can costs be derived from the orders of two sequences? In the following sub-sections, will introduce the *degree of order* ("orderdness") of solutions, and show how the overall orderdness of a solution can be derived from what we call "local" degrees or order. Finally, we will show that to integrate the aspect of order into our approach on event-sequence similarity, cost-functions on pairs of mappings come into play.

Note that the approach on null-mappings as discussed in section 6.5.2.4 is generally equivalent for the proposed cost-functions on the aspects of order, absolute temporal structure and relative temporal structures. The approach will be presented in full detail here; sections 6.5.3.4 and 0, however, will skip most of the common parts.

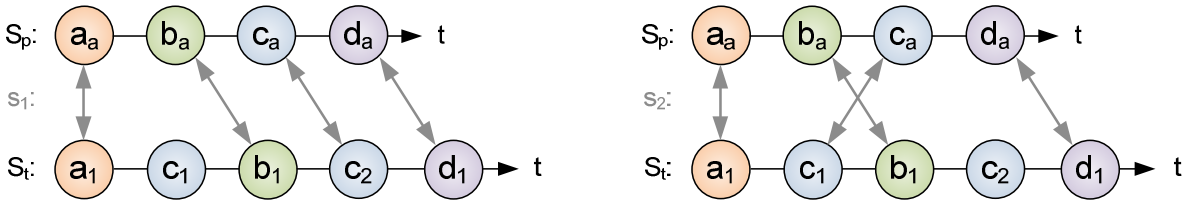
6.5.2.1 The orderdness of solutions

In short, a solution $s: S_p \rightarrow S_t \cup \{\varepsilon\}$ defines mappings between the events of a certain pattern sequence S_p to events of a certain target sequence S_t. Thereby, both the pattern-sequence events and those events of the target sequence that are comprised in s, $\{f | f \in S_t, s(e) = f \; \exists \; e \in S_p\}$, can be considered in a certain order as defined by their certain event-sequence, i.e., in an certain order in S_p and S_t, respectively. By "linking" events of both sequences, s establishes a relation between these two orders: The order of target-sequence events can more or less comply with the order of their "corresponding" events in S_t. We understand the described relation as the *degree or order*, or *orderdness*, of a solution.

Let us demonstrate the above considerations in a concrete example. Consider two event sequences S_p and S_t as shown below:



Assuming event-type compatibility as described in section 6.3.3, the following solutions, s_1 and s_2, exist:



It is easy to see that for s_1, the pattern-sequence events are in the same order in S_p as their representations in S_t, i.e., $pos(s_1(e), S_t) < pos(s_1(f), S_t)$ holds for each pair of events (e, f) with $e, f \in S_p$ and $pos(e, S_p) < pos(f, S_p)$. For s_2, however, this is not the case. Here, $pos(s_2(b_a), S_t) > pos(s_2(c_a), S_t)$ holds while $pos(b_a, S_p) < pos(c_a, S_p)$; in other words, b_a and c_a are in different order in S_p than their representations, b_1 and c_1, in S_t. Figure 21 depicts this deviation by connecting target-sequence events that are mapped to successive pattern-sequence events, with arrows indicating the order of the corresponding pattern-sequence events.

Figure 21: Comparing the order of solutions

6.5.2.2 Assessing orderdness locally

In the above example, we have assessed the degree of order of a solution $s: S_p \rightarrow S_t \cup \{\varepsilon\}$ by comparing the positioning of the pattern-sequence events with the positioning of the corresponding target sequence events. We've done so in a fairly informal matter, though. In a first step, let us now do that comparison "locally", i.e., for two pairs of mappings $(e, s(e))$ and $(f, s(f))$, $e, f \in S_p$, in s: By comparing the relative positioning of e and f in S_p with the relative positioning of $s(e)$ and $s(f)$ in S_t, we can assess the orderdness of s locally with respect to the given pairs of events.

For a clear terminology throughout the following sections, let us introduce the concept of *distance* between pattern-sequence events *in* a certain solution.

Definition: Given a solution $s: S_p \rightarrow S_t \cup \{\varepsilon\}$ and two pattern-sequence events $e, f \epsilon S_p$, we refer to the result of a function $d((e, f), s) = pos(s(f), S_t) - pos(s(e), S_t)$ as the *distance* of e and f *in* s.

In the following, we will examine the various scenarios that may arise in solutions. With later considerations in mind, we will restrict our discussion to pairs of successive mappings. Figure 22a depicts a case where the distance between two successive pattern-sequence events e and f in a certain solution s is 1, i.e., where the match for the succeeding pattern-sequence element directly succeeds the match for the preceding pattern-sequence element. It is easy to see that with respect to a.) the order, and b.) the certain pair of mappings, S_p and its solution s of S_t fit perfectly. The distance in s may, of course, also be greater than 1 (Figure 22b), or even negative (Figure 22c, Figure 22d).

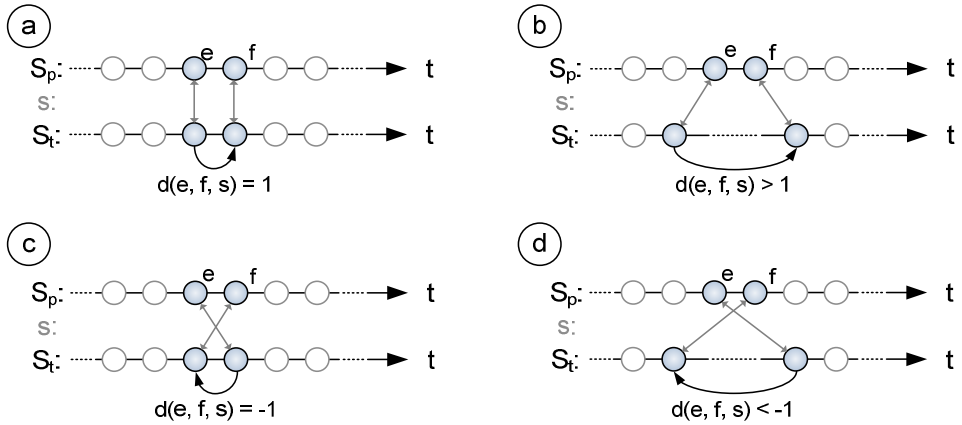


Figure 22: Comparing the distances between succeeding mappings

In cases where the distance in s is greater than 1, e and f are in a "correct" order in s, i.e., $pos(s(e), S_t) < pos(s(f), S_t)$. Yet, as $n = d(e, f, s) - 1$ other events (one might call them "additional" in the given context) are positioned between $s(e)$ and $s(f)$ in S_t, f does not succeed e *directly* in s. In cases where the distance in s

is negative, e and f are in a "wrong" order in s, i.e., $pos(s(e), S_t) > pos(s(f), S_t)$ while $pos(e, S_p) < pos(f, S_p)$. Furthermore, if $d(e, f, s) < -1, n = 1 - d(e, f, s)$ additional events are positioned between $s(e)$ and $s(f)$ in S_t. In all of the above scenarios, deviations of more or less "extent" exist between S_p and its solution s of S_t, again, of course, with respect to the order and the certain locality.

6.5.2.3 From local deviations to costs

In the above section, we have assessed the orderdness of solutions purely locally, i.e., with respect to certain pairs of mappings. So, let us consider a case where for a certain solution s, the local "degrees of order" for *all pairs of succeeding mappings* in s are given. It is easy to that - at least in some respect - the overall order of s is characterized by the sum of local orders: A solution that comprises several "large" distances between succeeding pattern-sequence events, for instance, will be considered more or less improper with respect to the order. Instead, a solution where each such distance is 1 clearly fits the pattern-sequence perfectly.

Deriving the overall orderdness of a solution from local (and generally independent) orders allows us to integrate a so-defined order into the proposed, dynamic algorithm. Consider a solution $s: S_p \rightarrow S_t \cup \{\varepsilon\}$ and two pattern-sequence events e and $f, e, f \in S_p$. Obviously, the absolute difference between $d(e, f, S_p)$ and $d(e, f, s)$, i.e., absolute difference between the distances from e to f in S_p and s, may serve as starting point for such cost function. As in S_p, the distance between two succeeding pattern-sequence events is always 1, a cost-function $c_{order}{}': S_p \times \mathbb{E} \times S_p \times \mathbb{E} \rightarrow \mathbb{R}_0{}^+$ on pairs of mappings may be a however-defined transformation $t: \mathbb{R}_0{}^+ \rightarrow \mathbb{R}_0{}^+$ of $|d(e, f, s) - 1|$:

$$c_{order}{}'(e, s(e), f, s(f)) = t(|d(e, f, s) - 1|) = t(|d(s(e), s(f), S_t) - 1|)$$

Note that t has two functions: First, it can be used to adapt the range of $c_{order}{}'$ to other cost-functions. More interesting, it may be used to further specify the business analyst's certain priorities: For a business analyst, for instance, scenarios where the distance between two succeeding pattern-sequence events e and f in a solution s is negative, i.e., where e and f are in the "wrong order" in s, may have much stronger impact on the perceived, overall similarity than scenarios where there are "just" additional events between $s(e)$ and $s(f)$.

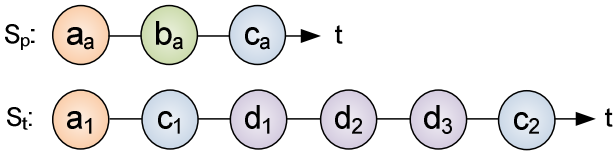
6.5.2.4 Null-mappings

As yet, we have assumed that only normal mappings are valid, i.e., that all valid mappings comprise certain events of the target sequence. The above formula will, of course, fail if the input comprises one or two *null-mappings*.

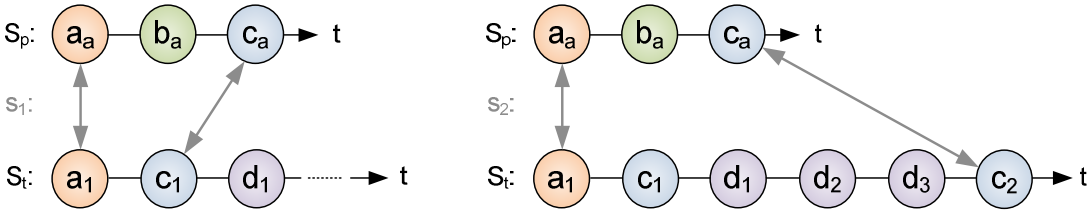
In section 6.5.1.2, we have used explicitly-defined costs in order to extend the original cost-function for event-level similarities. Generally, this strategy applies in the present case as well: Given a pattern sequence S_p, we let the define a collection of explicit cost-factors in case of null-mappings $c_{\varepsilon_1}, c_{\varepsilon_2}, \ldots, c_{\varepsilon_{|S_p|-1}}$. With s being a however-defined solution $s: S_p \rightarrow S_t \cup \{\varepsilon\}$ and e_i addressing the i^{th} event of S_p, c_{ε_j} defines the costs of pairs of mappings $\left(\left(e_j, s(e_j)\right), \left(e_{j+1}, s(e_{j+1})\right)\right), s(x) = \varepsilon \; \exists \; x \in \{e_j, e_{j+1}\}$, i.e., comprising one or two null-mappings. In order to find adequate costs in case of null-mappings, one can now ask the following question:

> *At which degree of local disorderdness shall it be more efficient to prefer a null-mapping over a given, normal mapping?*

The above approach has shortcomings, though: Consider a pattern-sequence S_p and a target-sequence S_t as shown below:



Assuming event-type compatibility, two solutions, s_1 and s_2, of S_t exists for S_p:

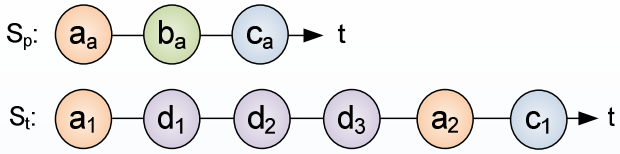


With the above approach on null-mappings, equal overall costs are calculated for s_1 and s_2. As there are additional events between $s_2(a_a)$ and $s_2(c_a)$, this clearly conflicts with common expectations, though. We therefore adapt the original approach as follows:

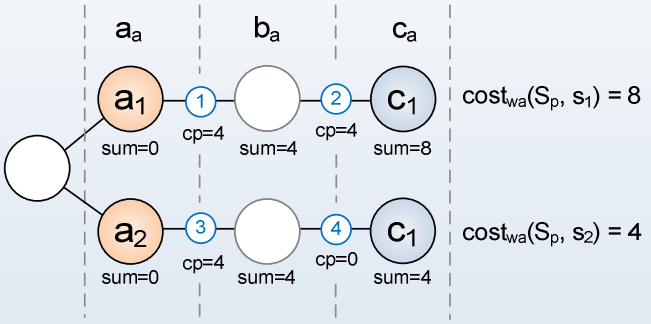
Consider a solution $s: S_p \to S_t \cup \{\varepsilon\}$. For a pair of succeeding mappings in s, $\big((e, s(e)), (f, s(f))\big)$, with $d(e, f, S_t) = 1$ and $s(f) = \varepsilon$, i.e., where the succeeding mapping is a null-mapping, we return the certain, explicitly defined cost-factor as shown above. For the further steps of the algorithm, however, we "virtually" assign f the *position in s* of its predecessor e, i.e., we "virtually" set $pos(f, s) = pos(e, s)$. Consequently, in case of further null-mappings, the described virtual position in s is carried forward, i.e., for a pattern-sequence event g with $d(f, g, S_t) = 1$ and $s(g) = \varepsilon$, we set $pos(g, s) = pos(f, s) = pos(e, s)$.

Henceforth, when a normal mapping succeeds a (virtually positioned) null-mapping, we do *not* return an explicitly defined cost-factor, but instead calculate costs based upon the null-mappings *virtual* position in s. If a solution begins with one or more null-mappings, no virtual position is available, though. Here, when a normal mapping succeeds such null-mapping, the certain, explicitly-defined cost-factor is returned anyway.

Example: Consider two event sequences S_t and S_p as shown below:



Given a cost-function $c_{order}: S_p \times (\mathbb{E} \cup \{\varepsilon\}) \times S_p \times (\mathbb{E} \cup \{\varepsilon\}) \rightarrow {\mathbb{R}_0}^+$ based upon $c_{order}{}'\big(e, s(e), f, s(f)\big) = 2 * \big(1 - d(e, f, s)\big)$, uniformly distributed weights an and explicit cost-factor in case of null-mappings, $c_{\varepsilon_1} = 8$, the following, overall costs are calculated:



Weights: $\omega_i = 0{,}5$ f.a. $0<i<3$

(1) In the path's solutions, b_a is mapped to ε; we therefore use c_{ε_1} as the (yet to be weighted) cost-factor:
cp = $8*\omega_1 = 4$

Furthermore, for all solutions S following from the according tree-node, we set
pos(b_a, s) = pos(a_a, s) = 1 f.a. s∈S.

(2) As resulting from (1), we assume pos(b_a,s) = 1 and calculate a cost-factor just as usual:
$|d(b_a, c_a, s) - 1| = 4$
cp = $4*2*\omega_2 = 4$

(3) Again, in the path's solutions, b_a is mapped to ε. Now, for all solutions S' following from the according tree-node, we set
pos(b_a, s) = pos(a_a, s) = 3 f.a. s∈S'.

(4) As resulting from (3), we assume pos(b_a,s) = 1 and calculate a cost-factor just as usual:
$|d(c_a, d_a, s) - 1| = 0$
cp = $0*2*\omega_2 = 0$

6.5.2.5 Pseudo code

In pseudo code, the proposed cost-function can be described as follows:

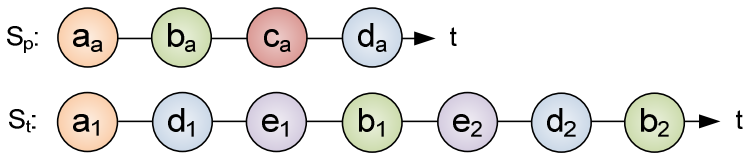
Name:	**getCostsB**	
Description:	Calculates a cost-factor from the local orderdness of two successive mappings. Requires that the tree is created depth-first.	
Input:	**prevPEvent:**	An event of the pattern sequence.
	prevTEvent:	The corresponding event of **prevPEvent** in the target sequence, or ε.
	pEvent:	The event succeeding **prevPEvent** in the pattern sequence
	tEvent:	The corresponding event of **pEvent** in the target sequence, or ε.
	index:	The current level of the tree.
	nullMappingCosts:	A map linking the first event of a pair of successive pattern-sequence events to the corresponding cost-factors for null-mappings.
	t:	A user defined transformation function.
Output:	A cost-factor from $\mathbb{R}_0{}^+$.	
Variables:	**posTEvent:**	The position of **tEvent** in the target sequence.
	lastPosition:	The position of the last target-sequence event in the given solution.
State:	**lastPositions:**	A field holding the position of the last target-sequence event in a solution for each level of the tree. Used for the "virtual positioning" of null-mappings. Initialized with lastPositions[1] = null;

```
01:     // Set lastPosition, update lastPositions
02:     Integer lastPosition;
03:     if (prevTEvent = ε) then
04:         lastPosition = lastPositions[index – 1];
05:     else
06:         // For a normal mapping, set "last position" to the position of the pair's first target-
07:         // sequence event.
08:         lastPosition = getPositionInTargetSequence(prevTEvent);
09:     end
10:     lastPositions[index] = lastPosition;
11:
12:     // Check whether both position are available
13:     if (tEvent = ε) or (lastPosition = null) then
14:         // Return user-defined cost factor for null-mappings
15:         return nullMappingCosts[prevPEvent];
16:     else
17:         // Calculate cost factor "as usual"
18:         int posTEvent = getPositionInTargetSequence(tEvent);
19:         return t(|1 – (posTEvent – lastPosition)|);
20:     end
```

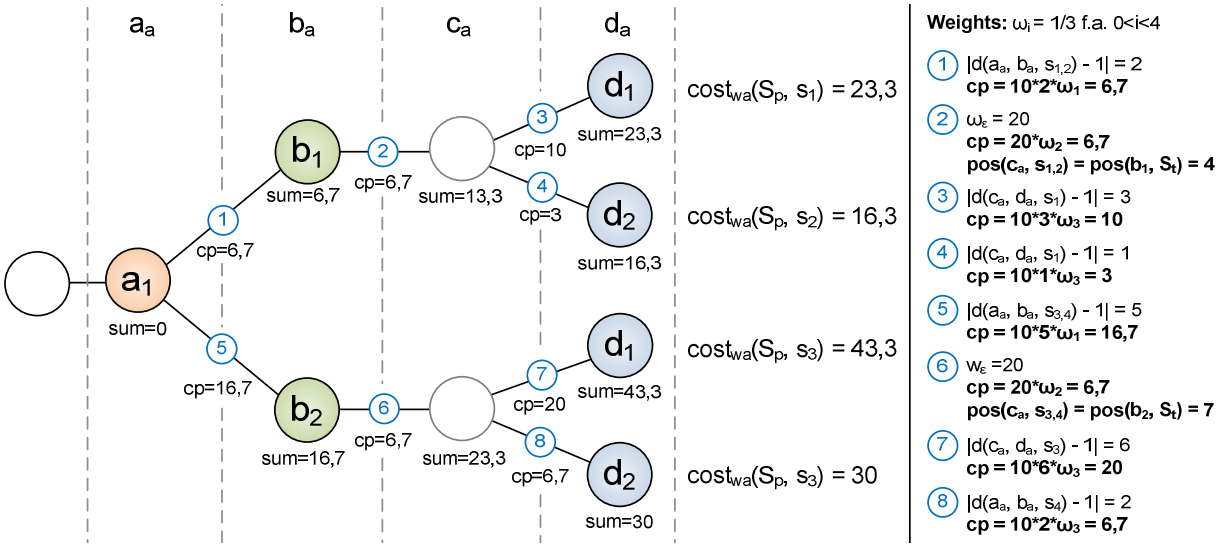
Algorithm 6: A cost-function for the aspect of order

6.5.2.6 Example

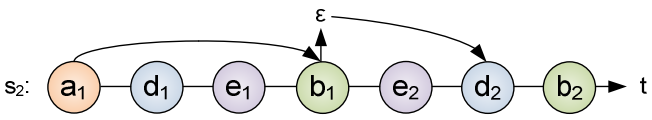
Consider two event sequences S_p and S_t as shown below:



As the business analyst knows about the data's certain characteristics, he or she considers null-mappings valid only for c_a and defines an explicit cost-factor $c_{\varepsilon_2} = 20$. With a cost-function c_{order} based upon $c_{order}'\big(e, s(e), f, s(f)\big) = 10 * \big(1 - d(e, f, s)\big)$ and uniformly distributed weights, the best-possible solution $s_{best}: S_p \rightarrow S_t \cup \{\varepsilon\}$ is then calculated as follows:



Thus, with overall costs of 16,3, the best-possible solution $s_2: S_p \rightarrow S_t \cup \{\varepsilon\}$ comprises the following target-sequence events:



6.5.3 Cost-function C: Absolute temporal structure

As a third cost-function, let us define c_{atime} in accordance with the dimension of *absolute temporal structures*, so that low cost-factors are calculated for sequences whose events are in a similar absolute temporal structure, and high cost-factors otherwise. In many respects, the approach presented here follows the same principles as the approach on order as discussed in section 6.5.2: From a conceptual point of view, the (overall or local) *temporal quality* of a solution corresponds to the (overall or local) *orderdness* of a solution. With this relation in mind, we won't step into too much detail here and instead keep things brief and simple.

6.5.3.1 The temporal quality of solutions

In a first step, let us walk through the general idea of a *temporal quality* of solutions: Given a solution $s: S_p \rightarrow S_t \cup \{\varepsilon\}$, both the pattern-sequence events, $\{e | e \in S_p\}$, and those events of the target sequence that are comprised in s, $\{f | f \in S_t \cap s(e) = f \exists\, e \in S_p\}$, can be considered in a certain temporal structure. By "linking" events of both sequences, s establishes a relation between the two structures: The temporal structure of target-sequence events can more or less comply with the temporal structure of their "corresponding" events in S_t, i.e., of the events they are mapped to in s. We understand the described relation as the *temporal quality* of a solution.

6.5.3.2 Assessing the temporal quality locally

In section 6.5.2.2, we have derived the overall order of a solution from what we called "local orders". Thereby, local orders are calculated by comparing the relative positioning of two succeeding pattern-sequence events in the pattern-sequence and in the certain solution. Regarding the temporal quality of a solution, a very similar strategy can be applied. First, however, let us introduce the *time span* between two pattern-sequence events *in* a solution:

Definition: Given an event sequence S, we refer to the result of a function $v_t: S \times (S \to \mathbb{E})$ with $v_t(e,s) = v_t\big(s(e)\big)$ as the *time of occurrence of e in s*. Note that for a function $s': S \to \mathbb{E} \cup \{\varepsilon\}$ with $s'(e) = \varepsilon$, $v_t(e,s')$ is not defined.

Definition: Given an event sequence S, we refer to the result of a function $t: S \times S \times (S \to \mathbb{E})$ with $t(e,f,s) = t\big(s(e),s(f)\big) = v_t\big(s(f)\big) - v_t\big(s(e)\big)$ as the *time span between e and f in s*. Again, for a function $s': S \to \mathbb{E} \cup \{\varepsilon\}$ with $s'(e) = \varepsilon$ and/or $s'(f) = \varepsilon$, $t(e,f,s')$ is not defined.

Consider a solution $s: S_p \to S_t \cup \{\varepsilon\}$ and two succeeding pattern-sequence events e and f; $e,f \in S_p$, $d\big(e,f,S_p\big) = 1$. One can now assess the quality of s - *locally* with respect the given mappings - by comparing $t\big(e,f,S_p\big)$ and $t(e,f,s)$, i.e., by comparing the time span between e and f in S_p and in s.

6.5.3.3 From local deviations to costs

Given the above context, it is easy to see that the absolute difference between $t(e,f)$ and $t(e,f,s)$, $|t(e,f) - t(e,f,s)|$, can serve as a metric for the local temporal quality of a solution s. A cost-function $c_{atime}{}': S_p \times \mathbb{E} \times S_p \times \mathbb{E} \to \mathbb{R}_0{}^+$ on pairs of mappings may hence be a however-defined transformation $u: \mathbb{R}_0{}^+ \to \mathbb{R}_0{}^+$ of $|t(e,f) - t(e,f,s)|$:

$$c_{atime}{}'\big(e,s(e),f,s(f)\big) = u(|t(e,f) - t(e,f,s)|) = u\big(\big|t(e,f) - t\big(s(e),s(f)\big)\big|\big)$$

Again, u may be used to adapt the range of $c_{atime}{}'$ to other cost-functions, and also to further specify the business analyst's certain priorities.

6.5.3.4 Null-mappings

As the above formula's domain is restricted to pairs of normal mappings, input comprising one or two null-mappings requires special handling. The proposed strategy, however, clearly resembles the approach presented in section 6.5.2.4:

For a pair of succeeding mappings in s, $\Big(\big(e,s(e)\big),\big(f,s(f)\big)\Big)$, with $d(e,f,S_t) = 1$ and $s(f) = \varepsilon$, i.e., where the succeeding mapping is a null-mapping, we return a certain, explicitly-defined cost-factor. For the further steps of the algorithm, we "virtually" assign f the time of occurrence in s of its predecessor e, i.e., we set $v_t(f,s) = v_t(e,s)$.[31] Henceforth, when a normal mapping succeeds a (virtually timed) null-mapping, we do not return an explicitly defined cost-factor, but instead calculate costs based upon the null-mappings "virtual" time of occurrence in s. If a solution begins with one or more null-mappings, no virtual occurrence time is available,

[31] This is, in fact, equivalent to what we have done in section 6.5.2.4, where we have set a "virtual" position in s.

though. Here, when a normal mapping succeeds such a null-mapping, the certain, explicitly-defined cost-factor is returned anyway.

6.5.3.5 Pseudo code

In pseudo code, the proposed cost-function can be described as follows:

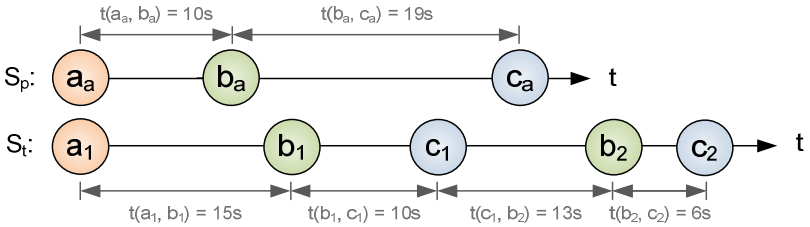
Name:	**getCostsC**	
Description:	Calculates a cost-factor from the absolute temporal deviations in two successive mappings. Requires that the tree is created depth-first.	
Input:	**prevPEvent:**	An event of the pattern sequence.
	prevTEvent:	The corresponding event of **prevPEvent** in the target sequence, or ε.
	pEvent:	The event succeeding **prevPEvent** in the pattern sequence
	tEvent:	The corresponding event of **pEvent** in the target sequence, or ε.
	index:	The current level of the tree.
	nullMappingCosts:	A map linking the first event of a pair of successive pattern-sequence events to the corresponding cost-factors for null-mappings.
	t:	A user defined transformation function.
Output:	A cost-factor from $\mathbb{R}_0{}^+$.	
Variables:	**pATimeSpan:**	The absolute time span between between **prevPEvent** and **pEvent**.
	tATimeSpan:	The absolute time span between between **lastTimeStamp** and **tEvent**.
	lastTimeStamp:	The time stamp of the last target-sequence event in the given solution.
State:	**lastTimeStamps:**	A field holding the time stamp of the last target-sequence event in a solution for each level of the tree. Used for the "virtual positioning" of null-mappings. Initialized with lastTimeStamps[1] = null;

```
01:     // Set lastTimeStamp, update lastTimeStamps
02:     TimeStamp lastTimeStamp;
03:     if (prevTEvent = ε) then
04:         lastTimeStamp = lastTimeStamps[index – 1];
05:     else
06:         // For a normal mapping, set "last time stamp" to the time stamp of the pair's first
07:         // target-sequence event.
08:         lastTimeStamp = v_t(prevTEvent);
09:     end
10:     lastTimeStamps[index] = lastTimeStamp;
11:
12:     // Check whether both position are available
13:     if (tEvent = ε) or (lastTimeStamp = null) then
14:         // Return user-defined cost factor for null-mappings
15:         return nullMappingCosts[prevPEvent];
16.     else
17:         // Calculate cost factor "as usual"
18:         double pATimeSpan = v_t(pEvent) - v_t(prevPEvent);
19:         double tATimeSpan = v_t(pEvent) - lastTimeStamp;
20:         return t(|pATimeSpan – tATimespan|);
21:     end
```

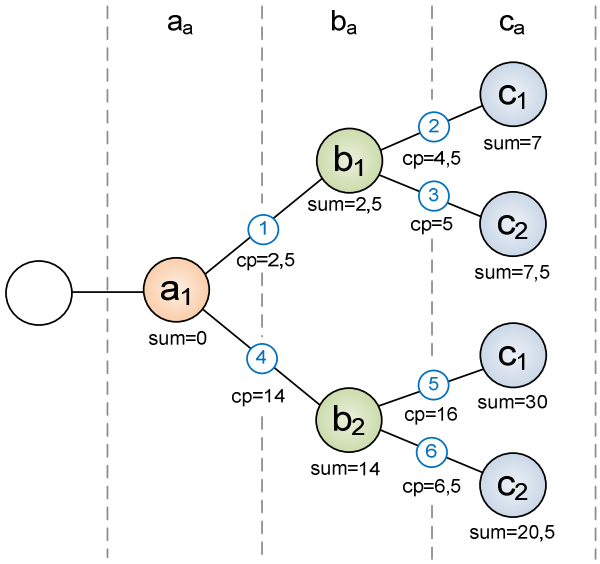
Algorithm 7: A cost-function for the aspect of absolute temporal structures

6.5.3.6 Example

Consider two event sequences S_p and S_t as shown below:



The business considers null-mappings invalid for all events of the pattern sequence. With a cost-function c_{atime} based upon $c_{atime}'(e, s(e), f, s(f)) = \frac{|t(e,f) - t(s(e),s(f))|}{100}$ and uniformly distributed weights, the best-possible solution $s_{best}: S_p \rightarrow S_t \cup \{\varepsilon\}$ is then calculated as follows:



$cost_{wa}(S_p, s_1) = 7$

$cost_{wa}(S_p, s_2) = 7,5$

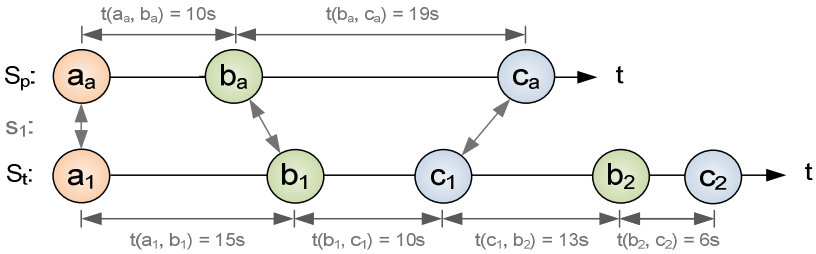
$cost_{wa}(S_p, s_3) = 30$

$cost_{wa}(S_p, s_4) = 20,5$

Weights: $\omega_i = 0,5$ f.a. $0<i<3$

1. $t(a_a, b_a) = 1000$ millis
   $t(a_1, b_1) = 1500$ millis
   $|t(a_a, b_a) - t(a_a, b_a, s_{1,2})| = 500$ millis
   cp = (500/100)*ω_1 = 2,5
2. $t(b_a, c_a) = 1900$ millis
   $t(b_1, c_1) = 1000$ millis
   $|t(b_a, c_a) - t(b_a, c_a, s_1)| = 900$ millis
   cp = (900/100)*ω_2 = 4,5
3. $t(b_a, c_a) = 1900$ millis
   $t(b_1, c_2) = 2900$ millis
   $|t(b_a, c_a) - t((b_a, c_a), s_2)| = 1000$ millis
   cp = (1000/100)*ω_2 = 5
4. $t(a_a, b_a) = 1000$ millis
   $t(a_1, b_2) = 3800$ millis
   $|t(a_a, b_a) - t((a_a, b_a), s_{3,4})| = 2800$ millis
   cp = (2800/100)*ω_1 = 14
5. $t(b_a, c_a) = 1900$ millis
   $t(b_2, c_1) = -1300$ millis
   $|t(b_a, c_a) - t((b_a, c_a), s_3)| = 3200$ millis
   cp = (3200/100)*ω_2 = 16
6. $t(b_a, c_a) = 1900$ millis
   $t(b_2, c_2) = 600$ millis
   $|t(a_a, b_a) - t((a_a, b_a), s_3)| = 1300$ millis
   cp = (1300/100)*ω_2 = 6,5

Thus, with overall costs of 7, the best-possible solution $s_1: S_p \rightarrow S_t \cup \{\varepsilon\}$ comprises the following target-sequence events:



6.5.4 Cost-function D: Relative temporal structure

As a fourth and final cost-function, let us now define c_{rtime} in accordance with the dimension of *relative temporal structures*, so that low cost-factors are calculated for sequences whose events are in a similar relative temporal structure, and high cost-factors otherwise. Note that from the four cost-functions presented in the on-hand section, c_{rtie} plays a somewhat "special role": Unlike c_{sim}, c_{order} and c_{atime}, it is applicable in case of *full-sequence matching* only. We will discuss this issue in detail section 6.5.4.1. Our approach on full-sequence matching and *-linked matching will be presented in section 6.6.

Generally, it seems highly natural that comparing the absolute and comparing the relative temporal structure of two event sequences should follow mainly equivalent principles. In accordance with the *absolute time span* between two events, let us therefore introduce the *relative time span* between two pattern-sequence events *in* a solution:

Definition: Given a solution $s: S_p \rightarrow S_t \cup \{\varepsilon\}$, with e_i addressing the i^{th} event in S_p, we refer to the result of a function $l: \left(S_p \rightarrow \mathbb{E} \cup \{\varepsilon\}\right) \rightarrow {\mathbb{R}_0}^+$ with

$$l(s) = \max_{j=1\ldots|S_p| \wedge s(e_j)\neq\varepsilon} \left(pos\left(s\left(e_j\right), S_t\right)\right) - \min_{j=1\ldots|S_p| \wedge s(e_k)\neq\varepsilon} \left(pos(s(e_k), S_t)\right)$$

as the *overall length* of s.

Definition: Given a solution $s: S_p \rightarrow S_t \cup \{\varepsilon\}$ and two pattern-sequence events e and f, $e, f \epsilon S_p$, we refer to the result of a function $t_r: S_p \times S_p \times \left(S_p \rightarrow \mathbb{E}\right)$ with $t_r(e, f, s) = \frac{v_t(s(f)) - v_t(s(e))}{l(s)}$ as the *relative time span* between e and f *in* s. For a function $s': S \rightarrow \mathbb{E} \cup \{\varepsilon\}$ with $s'(e) = \varepsilon$ and/or $s'(f) = \varepsilon$, $t(e, f, s')$ is not defined.

We have now defined relative time spans between events, both in event-sequences (section 3.3) and in solutions. Given a solution $s: S_p \rightarrow S_t \cup \{\varepsilon\}$ and two succeeding pattern-sequence events e and f, $e, f \in S_p$, $d\left(e, f, S_p\right) = 1$, we therefore calculate the local quality of s by comparing $t_r\left(e, f, S_p\right)$ and $t_r(e, f, s)$, i.e., by comparing the relative time span between e and f in S_p and in s. A cost-function $c_{rtime}{}': S_p \times \mathbb{E} \times S_p \times \mathbb{E} \rightarrow {\mathbb{R}_0}^+$ on pairs of mappings that introduces the relative temporal structure is then defined in equivalence to $c_{atime}{}'$ as described in section 6.5.3, but builds upon relative time spans instead of absolute ones. Given the above context and a however-defined transformation $u: {\mathbb{R}_0}^+ \rightarrow {\mathbb{R}_0}^+$, $c_{rtime}{}'$ is thus defined as follows:

$$c_{rtime}{}'\left((e, f), s\right) = u\left(\left|t_r\left((e, f), S_p\right) - t_r\left((e, f), s\right)\right|\right).$$

Null-mappings, finally, are handled as described in section 6.5.3.4.

6.5.4.1 Restriction to full-sequence matching

It is easy to see that in order to calculate the relative time span between two pattern-sequence events in a solution $s: S_p \rightarrow S_t \cup \{\varepsilon\}$, the overall length $l(s)$ must be available. In the proposed, dynamic algorithm, however, each cost-factor is potentially calculated for several, distinct solutions. The only case were the overall length of all solutions that follow from a certain tree-node is a.) equal and b.) known a-priori is *full-sequence mapping*. Here, for all solutions of a target sequence S, $l(s) = l(S)$ holds. Therefore, with the on-hand approach, the relative temporal structure can be taken into account in case of full-sequence matching only.

6.5.4.2 Pseudo code

In pseudo code, the proposed cost-function can be described as follows:

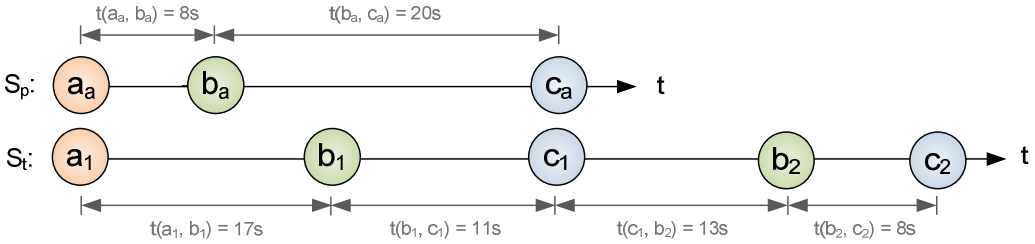
Name:	**getCostsD**	
Description:	Calculates a cost-factor from the relative temporal deviations in two successive mappings. Requires that the tree is created depth-first.	
Input:	**prevPEvent:**	An event of the pattern sequence.
	prevTEvent:	The corresponding event of **prevPEvent** in the target sequence, or ε.
	pEvent:	The event succeeding **prevPEvent** in the pattern sequence
	tEvent:	The corresponding event of **pEvent** in the target sequence, or ε.
	index:	The current level of the tree.
	nullMappingCosts:	A map linking the first event of a pair of successive pattern-sequence events to the corresponding cost-factors for null-mappings.
	t:	A user defined transformation function.
Output:	A cost-factor from $\mathbb{R}_0{}^+$.	
Variables:	**pRTimeSpan:**	The relative time span between between **prevPEvent** and **pEvent**.
	tRTimeSpan	The relative time span between between **lastTimeStamp** and **tEvent**.
	lastTimeStamp:	The time stamp of the last target-sequence event in the given solution.
State:	**lastTimeStamps:**	A field holding the time stamp of the last target-sequence event in a solution for each level of the tree. Used for the "virtual positioning" of null-mappings. Initialized with lastTimeStamps[1] = null;

```
01:     // Set lastTimeStamp, update lastTimeStamps
02:     TimeStamp lastTimeStamp;
03:     if (prevTEvent = ε) then
04:         lastTimeStamp = lastTimeStamps[index – 1];
05:     else
06:         // For a normal mapping, set "last time stamp" to the time stamp of the pair's first
07:         // target-sequence event.
08:         lastTimeStamp = v_t(prevTEvent);
09:     end
10:     lastTimeStamps[index] = lastTimeStamp;
11:
12:     // Check whether both position are available
13:     if (tEvent = ε) or (lastTimeStamp = null) then
14:         // Return user-defined cost factor for null-mappings
15:         return nullMappingCosts[prevPEvent];
16.     else
17:         // Calculate cost factor "as usual"
18:         double pRTimeSpan = getRelativeToPatternSequence(v_t(pEvent) - v_t(prevPEvent));
19:         double tRTimeSpan = getRelativeToTargetSequence(v_t(pEvent) - lastTimeStamp);
20:         return t(|pRTimeSpan – tRTimespan|);
21:     end
```

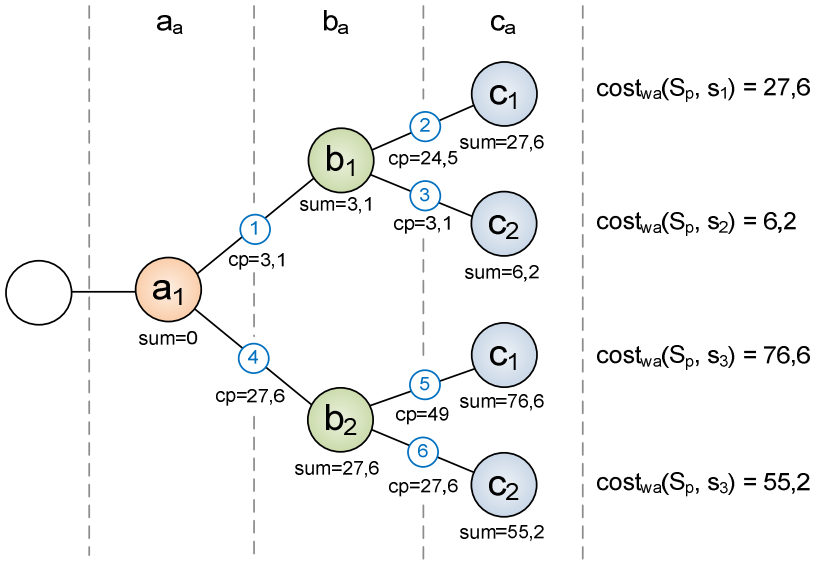
Algorithm 8: A cost-function for the aspect of relative temporal structures

6.5.4.3 Example

Consider two event sequences S_p, $l(S_p) = 28s$, and S_t, $l(S_t) = 49s$, as shown below:



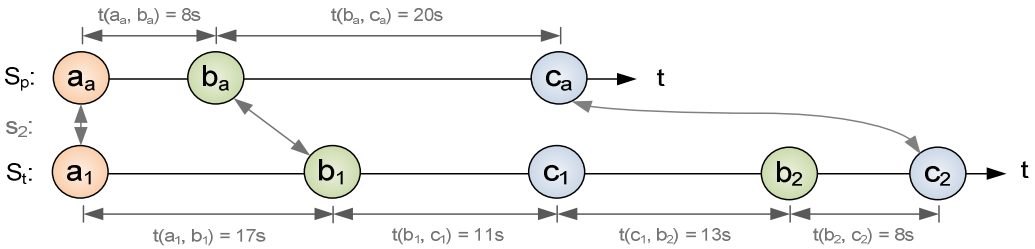
The business considers null-mappings invalid for all events of the pattern sequence. With a cost-function c_{rtime} based upon $c_{rtime}' = 100 * (|t_r(e,f,S_p) - t_r(e,f,s)|)$ and uniformly distributed weights, the best-possible solution $s_{best}: S_P \rightarrow S_t \cup \{\varepsilon\}$ is calculated as follows:



Weights: $\omega_i = 0,5$ f.a. $0<i<3$

1. $t_r(a_a, b_a, S_p) = 8/28 = 0,29$
   $t_r(a_1, b_1, S_t) = 17/49 = 0,35$
   $|t_r(a_a, b_a, S_p) - t_r(a_a, b_a, s_{1,2})| = 0,06$
   $cp = 100*0,06*\omega_1 = 3,1$
2. $t_r(b_a, c_a, S_p) = 20/28 = 0,71$
   $t_r(b_1, c_1, S_t) = 11/49 = 0,22$
   $|t_r(b_a, c_a, S_p) - t_r(b_a, c_a, s_1)| = 0,49$
   $cp = 100*0,49*\omega_2 = 24,5$
3. $t_r(b_a, c_a, S_p) = 20/28 = 0,71$
   $t_r(b_1, c_2, S_t) = 32/49 = 0,65$
   $|t_r(b_a, c_a, S_p) - t_r(b_a, c_a, s_2)| = 0,06$
   $cp = 100*0,06*\omega_2 = 3,1$
4. $t_r(a_a, b_a, S_p) = 8/28 = 0,29$
   $t_r(a_1, b_1, S_t) = 41/49 = 0,84$
   $|t(a_a, b_a, S_p) - t(a_a, b_a, s_{3,4})| = 0,55$
   $cp = 100*0,55*\omega_1 = 27,6$
5. $t_r(b_a, c_a, S_p) = 20/28 = 0,71$
   $t_r(b_2, c_1, S_t) = -13/49 = -0,27$
   $|t_r(b_a, c_a, S_p) - t_r(b_a, c_a, s_3)| = 0,98$
   $cp = 100*0.98*\omega_2 = 49,0$
6. $t_r(b_a, c_a, S_p) = 20/28 = 0,71$
   $t_r(b_2, c_2, S_t) = 8/49 = 0,16$
   $|t_r(b_a, c_a, S_p) - t_r(b_a, c_a, s_4)| = 0,55$
   $cp = 100*0,55*\omega_2 = 27,6$

Thus, with overall costs of $6,2$, the best-possible solution $s_2: S_P \rightarrow S_t \cup \{\varepsilon\}$ comprises the following mappings:



6.5.5 Similarity measures and event-sequence signatures

In section 6.3, we have stated that similarity measures on sequences of events as described in the on-hand thesis are defined “per pattern sequence”. Keep in mind, however, that a certain collection of *weights* could be

used for all pattern-sequences of a certain size. Also, c_{order}, c_{atime} and c_{rtime} as defined above are generally independent from the concrete pattern-sequence and the events therein.

Therefore, with single-event similarity-measures that are defined per *event type* (such as, for instance, described in section 6.5.1), a similarity-measure building upon the above cost-functions could be defined "per event-sequence signature". It is easy to see that this makes similarity-measures reusable across all pattern sequences having a certain signature and thus a certainly more powerful toolkit for business analysts. Yet, when other, event-specific cost-functions come into play, this may not be possible anymore. In neither case there is any kind of restriction on target sequences.

6.6 From sub-sequence matching to full-sequence matching

Throughout the last few sections, we have used the proposed algorithm solely for what we call *sub-sequence matching*, i.e., for finding the similarity of best-matching sub-sequence of a certain target-sequence. In the on hand section, we will show how the algorithm can be used for full-sequence matching, and also for start-linked matching and end-linked matching. It is essential to note, however, that the on-hand approach was designed in accordance with the above-defined, concrete cost-functions in mind; thus, it cannot be considered "fully general" with respect to the base algorithm as described in section 6.3. For possible other cost-functions, the present approach may be inappropriate; here, a wide adaption of the algorithm may be required.

6.6.1 Additional characteristics

Obviously, in case of full-sequence matching, certain characteristics of the target-sequence must be taken into account that are ignored in case of sub-sequence mapping. Consider a solution $s: S_p \rightarrow S_t \cup \{\varepsilon\}$ as shown below:

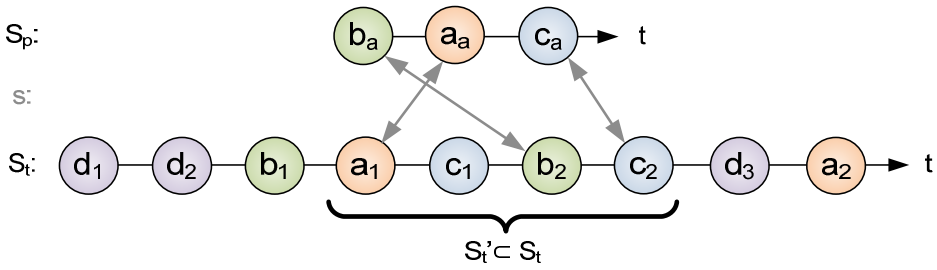


Figure 23: Exemplary sub-sequence matching

It is easy to see that in case of *sub-sequence matching*, certain events of the target-sequence - d_1, d_2, b_1, d_3, and a_2 - are not taken into account at all.[32] Thus, from a calculation point of view, it doesn't make a difference whether s is of S_t or of a sub-sequence $S_t{}'$ of S_t as marked in the above figure.[33]

Now, let us reconsider the above example, assuming that *full-sequence matching* is requested by the business analyst. What else characteristics of S_t must be taken into account? Let us observe the problem with respect

[32] Note that c_1 is taken into account, even though it is not comprised in mappings as defined in s. Yet, it affects the distances between b_a and a_a, and a_a and c_a, in s, and therefore has an impact on the overall costs of s.

[33] More generally, given a solution $s: S_p \rightarrow S_t \cup \{\varepsilon\}$, with e_i addressing the i^{th} event of S_t, calculating the overall costs is equal for all sub-sequences $S_t{}'$ of S_t, $S_t' = (e_j, \dots, e_k)$, with $j \leq \min_{f \in S_p}(pos(s(f), S_t))$, $k \geq \max_{f \in S_p}(pos(s(f), S_t))$.

to the various aspects of event-sequence similarity. In the following, e_{p_i} and e_{t_i} address the i^{th} event of S_p and S_t, respectively.

Regarding **single-event similarities**, there is no difference between sub-sequence matching and full sequence mapping. In both cases, the events of the pattern-sequence are compared to their certain counterparts in the target-sequence.

Regarding **order**, the fact that there are

a. n events before $s\left(e_{p_1}\right)$ in S_t, and
b. m events after $s\left(e_{p_{|S_p|}}\right)$ in S_t,

must be taken into account. In the above example, $n = 3$ (d_1, d_2, and b_1), and $m = 2$ (d_3 and a_2).

Regarding the **temporal structure**, the time spans between

a. $s\left(e_{p_1}\right)$ and e_{t_1}, i.e., $t\left(s\left(e_{p_1}\right), e_{t_1}\right)$, and
b. $s\left(e_{p_{|S_p|}}\right)$ and $e_{t_{|S_t|}}$, i.e., $t\left(s\left(e_{p_{|S_p|}}\right), e_{t_{|S_t|}}\right)$,

must be taken into account. Figure 24 below depicts these time spans in the given example:

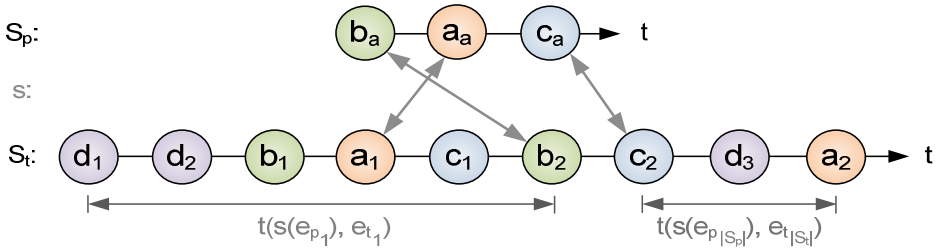


Figure 24: Temporal structures in case of full-sequence matching

6.6.2 Introducing start- and end-events

In order to provide *full-sequence matching* as been described in section 6.1.2, the above characteristics must affect the overall costs of a solution. In the on-hand section, we will introduce a simple "trick", allowing us considering the described characteristics while leaving both the base algorithm and the proposed cost-functions unchanged.

Consider a solution $s: S_p \rightarrow S_t \cup \{\varepsilon\}$. In case of full-sequence mapping, let us "conceptually" extend both the pattern-sequence and the target-sequence by two virtual events e_s (the start-event) and e_e (the end-event). Thereby, with e_i addressing the i^{th} event of an event-sequence S, e_s and e_e have the following characteristics:

$$pos(e_s, S) = 0$$

$$v_t(e_s) = v_t(e_1)$$

$$pos(e_e, S) = |S| + 1$$

$$v_t(e_e) = v_t(e_{|S|})$$

Thus, according to their positions in the certain event-sequence, one may refer to start- and end-events as the 0^{th} and the $(|S|+1)^{th}$ event in S, respectively.

Furthermore, we assume that the start-event of the target-sequence is the only match for the start-event of the pattern-sequence, and that the end-event of the target-sequence is the only match for the end-event of the pattern-sequence. Hence, each solution $s: S_p \rightarrow S_t \cup \{\varepsilon\}$ comprises two "additional" mappings, linking a.) the two start-events, and b.) the two end-events to one another. Further on, we will refer to the additional mappings as start- and end-mappings, respectively.

Example: Consider a solution $s: S_p \rightarrow S_t \cup \{\varepsilon\}$ as shown in Figure 23. In case of full-sequence matching, both event sequences S_p and S_t are extended by virtual start- and end-events. These events are then linked in start- and end-mappings as illustrated below:

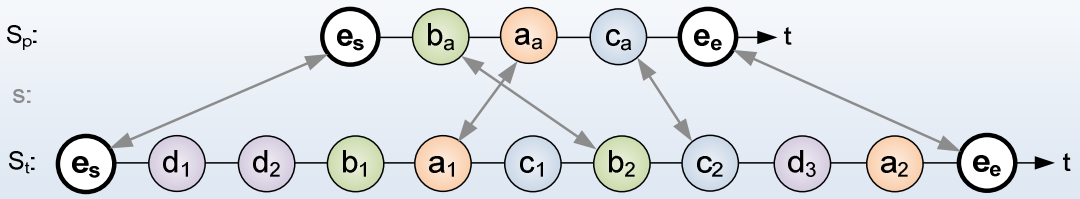
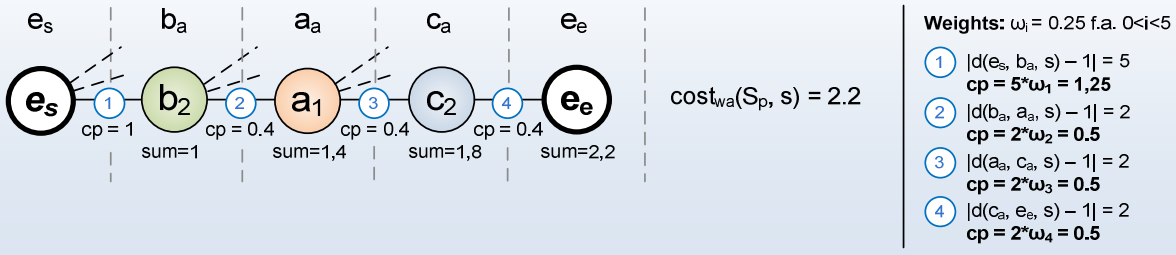


Figure 25: Full-sequence matching by assuming virtual start- and end-events

Let e_i address the i^{th} event of S_p. As part of the proposed algorithm, we now apply each given cost-function on pairs of mappings on both $\left((e_0, s(e_0)), (e_1, s(e_1))\right)$ and $\left(\left(e_{|S_p|}, s\left(e_{|S_p|}\right)\right), \left(e_{|S_p|+1}, s\left(e_{|S_p|+1}\right)\right)\right)$. With the thereby generated, additional cost-factors, all relevant characteristics as discussed in section 6.6.1 gain impact on the overall costs of s. Note that since start-events and end-evens are auxiliary constructs that have no attributes, c_{sim} is not applied to according mappings.

Example, continued: Consider a cost-function c_{order} based upon $c_{order}{}'\big(e, s(e), f, s(f)\big) = |1 - d(e, f, s) - 1|$. With *full-sequence matching* and uniformly distributed weights, the algorithm results in the following tree-path:



6.6.3 Adapted weighting

In the above section, we have calculated additional cost-factors by comprising mappings between start- and end-events. These additional cost-factors must, of course, be weighted. We therefore adapt the weighting model as proposed in section 6.3.2.3 as follows:

Consider a solution $s: S_p \to S_t \cup \{\varepsilon\}$ together with a collection of cost-functions on single mappings $c_{s_1}, c_{s_2}, \dots, c_{s_m}$ and a collection of cost-functions on pairs of mappings $c_{p_1}, c_{p_2}, \dots, c_{p_n}$. In case of full-sequence matching, we allow the user defining $k = \left(m * |S_p| + n * \left(|S_p| + 1\right)\right)$ weights $\omega_1, \omega_2, \dots, \omega_k$ with $\omega_j \in [0,1]\ \forall\ j = 1 \dots k$ and $\sum_{j=1}^{k} \omega_j = 1$, i.e., that sum to unity. With a however-defined aggregation function $f: {\mathbb{R}_0}^{+k} \times [0,1]^k \to {\mathbb{R}_0}^{+}$ and c_i addressing the i^{th} cost-factor calculated from s, a function $cost: S_p \times \left(S_p \to \mathbb{E} \cup \{\varepsilon\}\right)$ calculating the overall costs of a solution $s: S_p \to S_t \cup \{\varepsilon\}$ is then still defined as follows:

$$cost\left(S_p, s\right) = f(c_1, \dots c_k, \omega_1, \dots \omega_k)$$

6.6.4 A "mainly" consistent approach on matching modes

It is easy to see that both *start-linked matching* and *end-linked matching* can be implemented in full accordance with the above-presented approach on full-sequence mapping. Consider a solution $s: S_p \to S_t \cup \{\varepsilon\}$, and let e_i address the i^{th} event of S_p. In case of start-linked mapping, we simply weight those cost factors that are calculated from (e_0, e_1) by zero. Consequently, in case of end-linked mapping, we weight those cost-factors that are calculated from $\left(e_{|S_p|}, e_{|S_p|+1}\right)$ by zero. Weighting <u>all</u> additional cost-factors by zero, however, allows us performing sub-sequence matching as well.

Obviously, as according weights can be become arbitrarily small, the borders between the various matching modes are somewhat floating. Thus, by introducing virtual start- and end-events as defined above, a fully-consistent approach on matching modes is provided for c_{sim}, c_{order} and c_{atime} as presented in section 6.4.4. For c_{rtime}, however, this is not the case: Here, even if cost-factors arising from (e_0, e_1) and $\left(e_{|S_p|}, e_{|S_p|+1}\right)$ are weighted by zero, the according mappings affect the length of the resulting solution and, therefore, the relative time span between the solution's target-sequence events. As this clearly conflicts with common expectations, sub-sequence matching including zero-weighted start- and end-mappings is conceptually wrong when relative temporal structures shall be taken into account.

Figure 26 below illustrates the described problem: Even when the mappings between start- and end-events are weighted by zero, $l(S_t)$ is used for calculating the relative time spans in s. Yet, in case of sub-sequence matching, one would expect $l(S_t')$ instead.

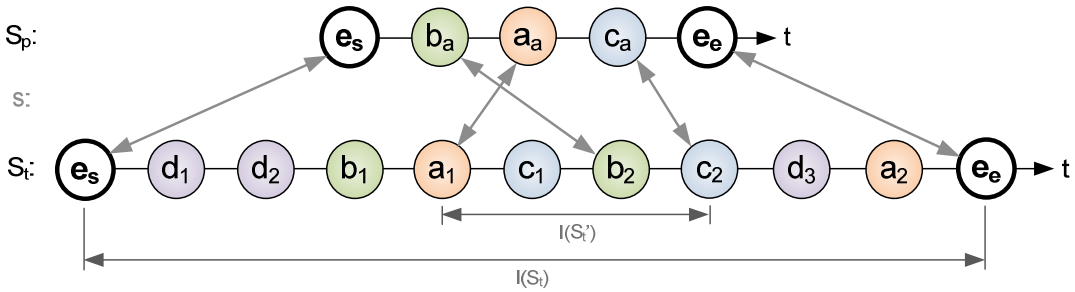


Figure 26: Conceptual problems in case of full-sequence matching

From an algorithm point of view, as the length of the various solutions is not available throughout the algorithm, c_{rtime} is applicable in case of however-weighted but "real and aware" *full-sequence matching* only. We have discussed this issue in section 6.5.4.1 above.

6.7 Discussion

Again, let us end this section with discussing the various "pros and cons" of the proposed approach, its properties and its complexity.

6.7.1 Pros and Cons

The presented approach on event-sequence similarity builds upon an intuitive, assignment-based understanding of sequence similarity. Thus, despite of an undeniable degree of complexity, it should appear somewhat natural to business analysts. Also, by letting the analyst choosing cost-functions, weights and compatibilities freely, it allows great flexibility and should be applicable in most scenarios: With the basic cost-functions as proposed in section 6.5, the essential dimensions of event-sequence similarity are covered. When other, previously undiscovered aspects gain relevance in the given context, an arbitrary number of additional cost-function may be defined. Furthermore, the conceptually clear separation of concerns into possible assignments (compatibilities), sources of costs (cost-functions) and impact of costs (weights) is a solid basis for further extensions of the algorithm - especially when the set of *valid solution* shall further be reduced. Suntinger [49] presents the particularly interesting concept of "blocks" in his master thesis.

Finally, note that for the proposed algorithm, *sub-sequence matching* can be considered the "default matching mode" and the starting point for all other matching modes. Applicable (at least) for the proposed, basic cost-functions, the presented approach on full-sequence matching allows a widely consistent and somewhat "fluent" handling of matching modes. Consequently, the proposed algorithm executes in (de facto) equal runtime for all four matching modes.

Let us now continue to the disadvantages of the on-hand approach: In section 6.6.4, we have stated that the dimension of relative temporal structures can only be calculated in case of full-sequence matching. There is, however, no such restriction "in general", and other approaches could allow considering that aspect in case of sub-sequence matching and *-linked matching as well. Also, as before for single-event similarity-measures, the presented approach requires lots of configuration and an even stronger involvement of the domain expert. All in all, the user has to define and configure

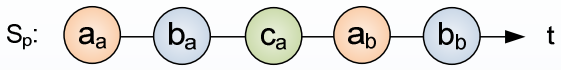
- adequate cost-functions,
- an adequate weighting configuration, and
- an adequate compatibility.

Again, a framework for event-sequence similarity-searching should comprise a library-like, persistent management facility allowing the reuse of proven similarity measures.[34] Eventually, the Branch-&-Bound-based algorithm requires considerable computational effort. We will discuss this issue in section 6.7.3 below.

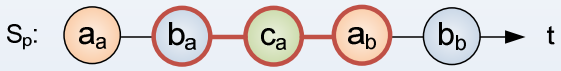
6.7.1.1 Weighting pairs of mappings

Clearly making the presented approach more flexible and applicable in a variety of use cases, we have listed the weighting model a notable pro above. Unfortunately, weighting pairs of mappings is fairly unnatural and also leads to some conceptual weaknesses. Let us demonstrate the most problematic case with a simple example:

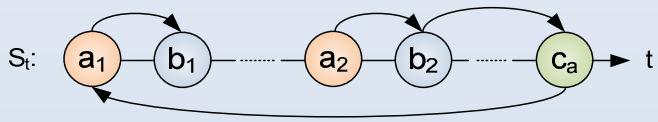
Example: Consider a pattern sequence S_p as shown below together a similarity measure sim_{order} comprising c_{order} as its only cost-function, i.e., calculating similarity solely from the aspect of order.



A business analyst may now consider $\boldsymbol{c_a}$'s position *in* a target sequence $\boldsymbol{S_t}$ as irrelevant for the overall similarity of $\boldsymbol{S_t}$. So, how can this be accomplished? Actually, the business analyst has to weight all cost-factors calculated from $(\boldsymbol{b_a}, \boldsymbol{c_a})$ and $(\boldsymbol{c_a}, \boldsymbol{a_a})$, i.e., the cost-factors "around" $\boldsymbol{c_a}$, by zero. Below, the according pairs of mappings are marked red.



With those weightings, however, not only the position of $\boldsymbol{c_a}$ hasn't any impact on the overall similarity, but also the relative positioning of $\boldsymbol{b_a}$ and $\boldsymbol{a_a}$ in the target sequence. Therefore, the following solution $s: S_p \rightarrow S_t \cup \{\varepsilon\}$ has overall costs of zero, which results in the *certain equality* of S_t and clearly conflicts with common expectations:



As yet, we have not found a solution for the presented scenario.

6.7.2 Properties

We have stated that per definition, similarity measures performing sub-sequence matching and *-linked matching are asymmetric. Thus, let us focus on cases where the proposed algorithm is used for full-sequence

[34] In the SENACTIVE EventAnalyzer, we allow the user choosing from all stored similarity-measures compatible with a given pattern sequence. Also, we allow the user adding various meta-information, such as, for instance, the informal description of a measure's semantics. For configuring c_{sim} (on single-event similarities) the repository is tightly coupled to the corresponding repository for single-event similarities.

matching. Here, in cases where the pattern sequence S_p and the target sequence S_t are of the same size, $|S_p| = |S_t|$, a similarity-measure as proposed in the on-hand section can be considered a strong similarity measure in the strict sense iff

a. the used compatibility is symmetric,
b. the aggregation function is a linear combination of cost-factors and weights, and
c. all comprised cost-functions can be considered *metrics*.

For a weak similarity measure in the strict sense, conditions can be defined accordingly. Yet, in the more general case that pattern- and target sequence are of different sizes, $|S_p| \neq |S_t|$, "missing events" and, correspondingly, "additional events" do not necessarily affect the overall similarity between two sequences of events in one and the same extent. Consequently, similarity-measures as proposed in the on-hand section can be considered *asymmetric,* and, following from that, *similarity measures in the common sense*.

6.7.3 Complexity

It is easy to see that in general, the runtime of the proposed algorithm depends on the number of times the loop from Algorithm 4, line 13, is executed. More plastically, this number can be considered the number of tree-nodes in the resulting tree. The loop's body, i.e., for the calculation of cost-factors, however, we assume requiring a constant time c.

For the proposed algorithm, the *best case* is that where there are no valid mappings for the first event of the pattern sequence. Here, a similarity of zero can be returned immediately. Though not explicitly described in Algorithm 4, we assume that given a sets of matches for all events of the pattern-sequence, this can be found out in a constant time c_{best}. With n and m addressing the number of events in the pattern- and the target sequence, the best-case runtime of the proposed algorithm is thus defined as follows:

$$T_{best}(n, m) = c_{best}$$

The worst case, by contrast, occurs where

a. each event of the target sequence and
b. the null-event

match each event of the pattern sequence and the threshold does not apply throughout the whole calculation. So, how many tree-nodes are evaluated in a so-defined scenario? In section 6.3.1, we have stated that without additional restrictions through a however-defined compatibility, $\sum_{k=0}^{\min(|S_p|,|S_t|)} \left(\frac{|S_p|!}{(|S_p|-k)!}\right) * \binom{|S_t|}{k}$ solutions of a target sequence S_t exist for a pattern sequence S_p. Consequently, as each leaf represents a distinct solution, the "worst-case" T_{worst} of S_p and S_t has $\sum_{k=0}^{\min(|S_p|,|S_t|)} \left(\frac{|S_p|!}{(|S_p|-k)!}\right) * \binom{|S_t|}{k}$ leaf nodes. Also, note that given a tree T as calculated from S_p, S_t and a compatibility c in Algorithm 3, the sub-tree T_i constituted from the nodes of level 1 to level i in T is equivalent to a tree T' as calculated from S_t, c and a subsequence $S_{p_i} \subseteq S_p$ constituted from the first i events in S_p. Following from that, the j^{th} level of the worst-case tree T_{worst} of S_p and S_t contains $\sum_{k=0}^{\min(j,|S_t|)} \left(\frac{j!}{(j-k)!}\right) * \binom{|S_t|}{k}$ tree nodes. In sum, T_{worst} contains $\sum_{i=1}^{|S_p|} \left(\sum_{k=0}^{\min(i,|S_t|)} \left(\frac{i!}{(i-k)!}\right) * \binom{|S_t|}{k}\right)$ nodes. The worst-case runtime of the proposed algorithm is therefore defined as follows:

$$T_{worst}(n,m) = \sum_{i=1}^{n} \left(\sum_{k=0}^{\min(i,m)} \left(\frac{i!}{(i-k)!} \right) * \binom{m}{k} \right) * c$$

In practice, worst-case runtime is avoided through

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

As solely depending on the given pair of sequences, stating a universally valid, average runtime is non-trivial and outside the scope of this thesis. In general, however, the typical proportion of the above-said worst-case runtime to "practical runtimes" can be considered equivalent to other Branch-&-Bound algorithms.

6.7.3.1 Branch & Bound vs. Dynamic Programming

As we have already touched upon, the given optimization problem - to find the best-possible solution of the target sequence for the pattern sequence - could also be solved using Dynamic Programming. Dynamic Programming clearly outperforms Branch & Bound strategies and allows to solve according problems in (pseudo) polynomial runtimes instead of exponential ones.

We opted for the presented Branch-&-Bound-based algorithm in view of possible extensions of the base algorithm, as, for instance, presented by Suntinger [49]. Consider Suntinger's "arbitrary order" block, weakening the order of certain pattern-sequence events: Here, resulting from dependencies across n events of the pattern sequence, a Dynamic-Programming-based approach might require notable workarounds most likely conflicts with the idea of a clear separation of concerns between the base algorithm and plug-in-like extensions.

An adaption of the presented algorithm towards Dynamic Programming or a however-defined hybrid solution is nevertheless crucial when similarity searching shall be applied to long event-sequences starting from several hundreds of events, e.g., complete customer interactions over several years: In the evaluation part of this thesis presented in section 7, we will see the presented approach is impracticable here, with execution performance being one of the most obvious reasons. It is easy to see that the above-said adaption of the presented, basic ideas of event-sequence similarity should be considered a subject of future research projects.

6.7.3.2 Enhanced branch-selection strategies

The presented Branch-&-Bound-based algorithm uses a very simple branch-selection strategy by choosing target-sequence events in their order of occurrence. Yet, as the algorithm strongly depends on the quick detection of "rather good" solutions (and thus reducing the threshold), a more sophisticated branch-selection strategy might result in notable performance improvements. Again, the investigation and evaluation of tailored selection strategies should be a subject for future work.

7 Application and results

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, we defined different objectives for each evaluation scenario. These 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.

The presented approaches were implemented in C#, a programming language from Microsoft's .Net-framework. As the implementation serves as a prototype and is part of the EventAnalyzer current production version, we decided to abandon a sophisticated threading concept.

7.1 C1 - Online gambling: User activity histories

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 27.

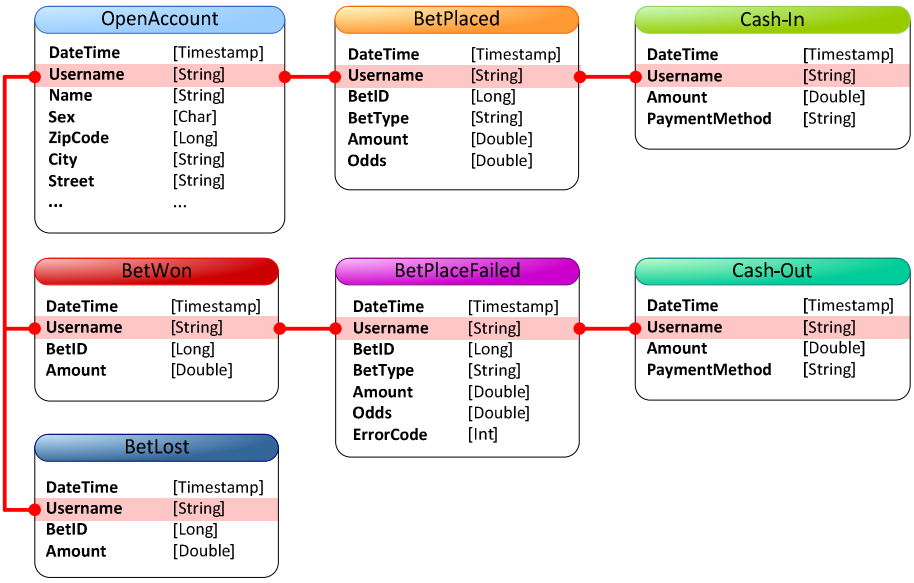


Figure 27: Event types and correlations in evaluation scenario C1 – Online gambling

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 bet placements are simulated only 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 both in the number of events occurring and in certain event attribute's values.

One possible pattern 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.

7.1.1 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.

7.1.2 C1.a - Order and sub-sequence matching

In the first scenario, we define a single-event similarity measures that exclusively incorporates the aspect of order. 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.

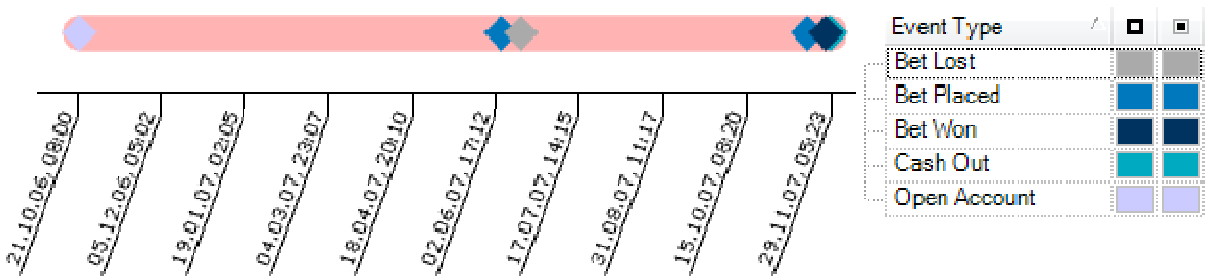


Figure 28: Search pattern for evaluation case C1.a

Figure 29 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 while the rest is ignored.

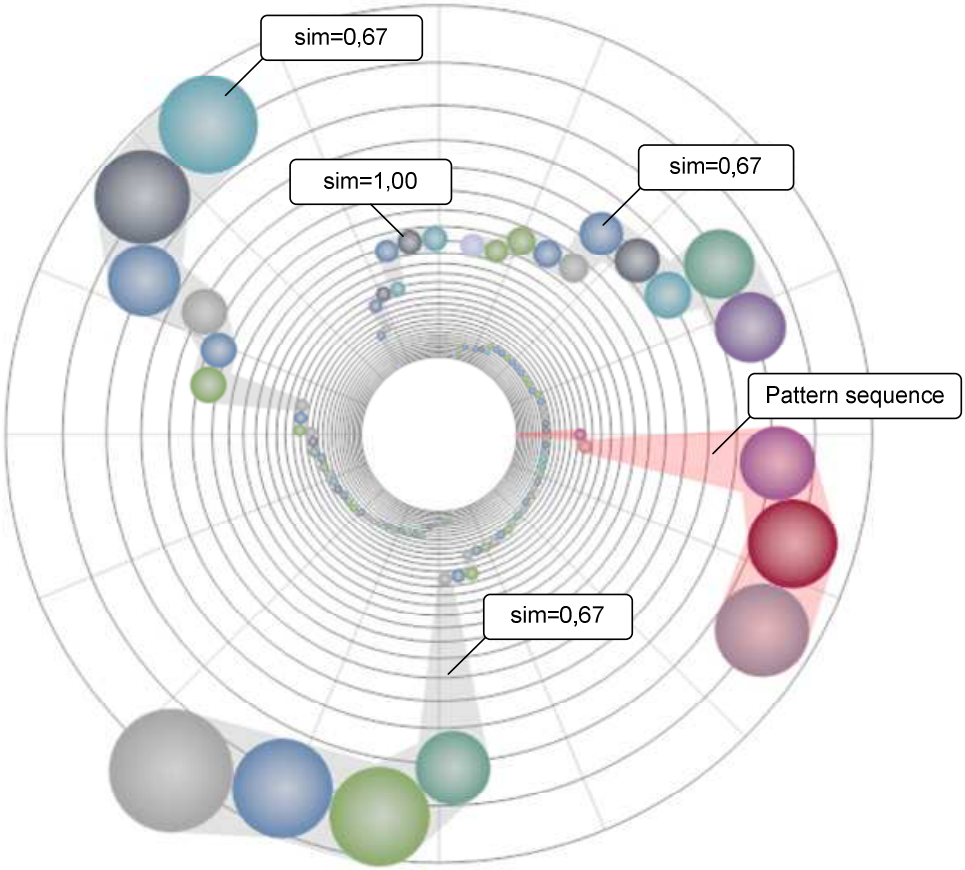


Figure 29: Best search results for scenario C1.a visualized in the Event Tunnel

7.1.2.1 C1.b - Order and full-sequence matching

Scenario C1.a showed that subsequence searching may lead to intuitively incorrect results for the given dataset. This scenario is defined equally to scenario C1.a, but performs full-sequence matching instead of sub-sequence matching. 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 30.

This scenario already fulfils 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.

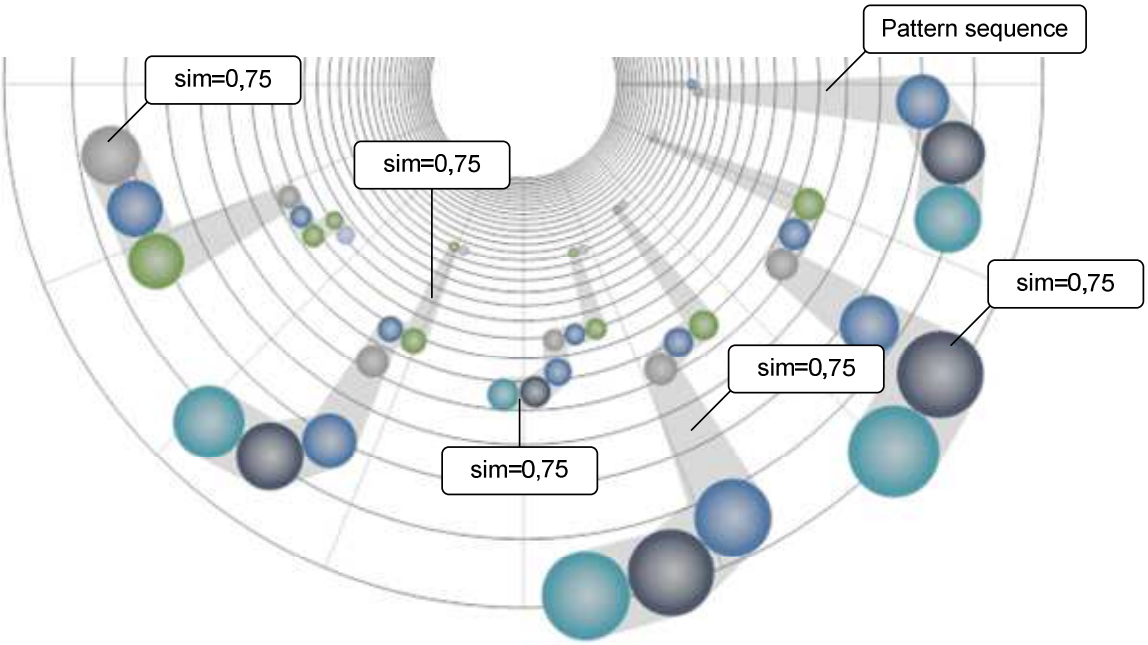


Figure 30: Best search results for scenario C1.b visualized in the Event Tunnel

7.1.3 C1.c – Order, temporal structures and full-sequence matching

For scenario C1.c we use the same search pattern as before, but incorporate the aspect of temporal structures.

The given evaluation scenario showed that the cost-function for absolute temporal structures 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.

Considering relative temporal structures works out significantly better for the described scenario. Still, the best matches in the previous scenario already had a very similar temporal structure (see Figure 30) so that again these sequences have been discovered as the best matches.

7.1.4 C1.d - Order and single-event similarities

In this scenario, the following event-attributes are considered via c_{sim}; as an attribute-level similarity technique, we decided to use the normalized absolute difference as described in section 5.5.1.3:

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

Again, the discovered sequences for this evaluation case again differ only slightly 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 has minimal impact 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 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 order deviations allowing null-mappings return reasonable result. The problem is similar as with temporal 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.

7.1.5 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[35]	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

Table 13: Performance results for evaluation scenario C1 without initial threshold

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

Table 14: Performance results for evaluation scenario C1 with initial threshold

[35] 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.

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.

7.2 C2 - Trouble tickets: Change history sequences

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 has 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 31 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.

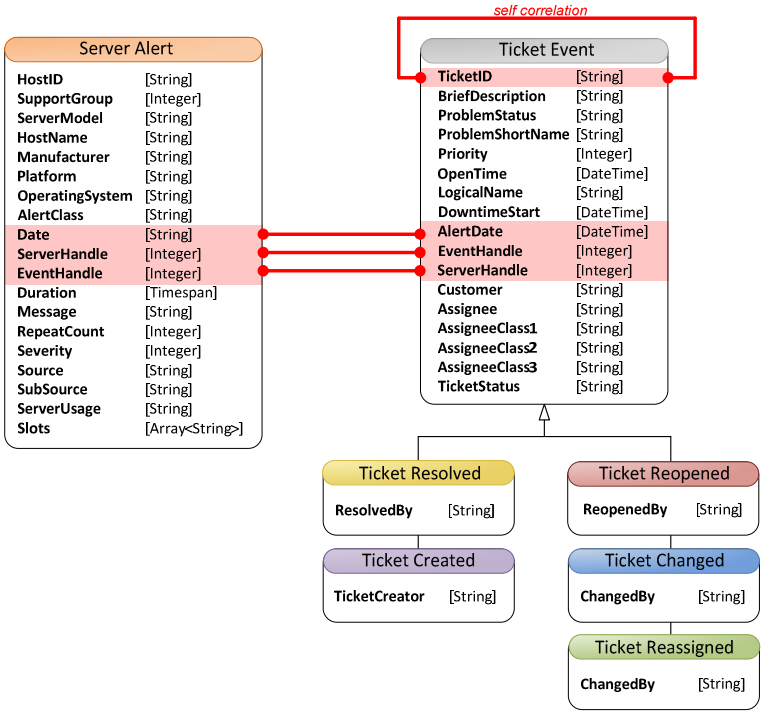


Figure 31: Event types and correlations in evaluation scenario C2 – Trouble tickets

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?
- Which 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 again forwarded to support group Z who's members finally resolve 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 whether this suspicion holds on the historic data.

7.2.1 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

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

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 32 shows that this ticket has a significantly long history with several ticket changes and also reassignments (blue, and green events).

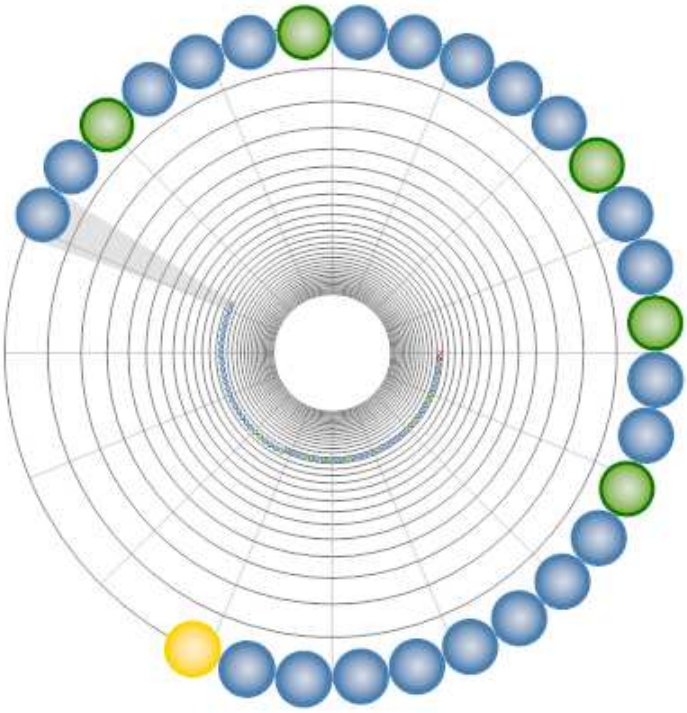


Figure 32: Activity history for a known incident ticket plotted in the event tunnel

For the given use case, the sequence of reassignment is of particular interest. Figure 33 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.

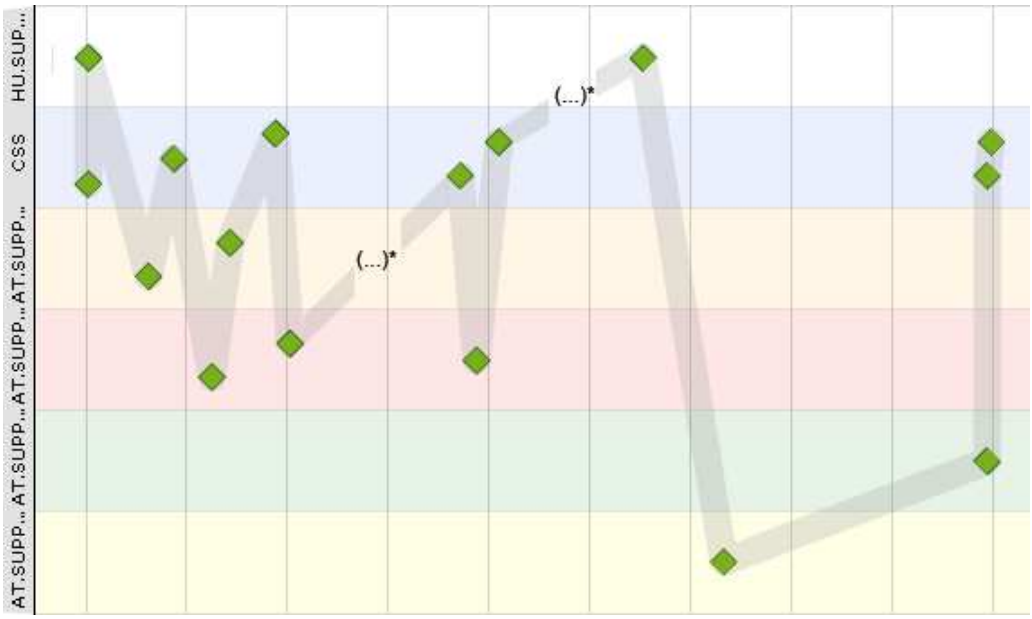


Figure 33: Sequence of ticket reassignment events over time (x-axis) by assigned support groups (y-axis)[36]

[36] 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.

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: Levenstein string similarity on *TicketReassigned.Assignee*

7.2.2.1 Search results and discussion

The search for the known, very long ticket event sequence returned no matches. This means, with the given configuration, no solution could be found with sufficiently low costs to be at least 50% similar. Our investigations on this result showed that there is 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.

7.2.2.2 Performance summary

- Total number of events: 165841
- Total number of event sequences: 87241
- 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 15: Performance results for evaluation scenario C2.a

7.2.3 C2.b – Finding reassignment scenarios

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 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.

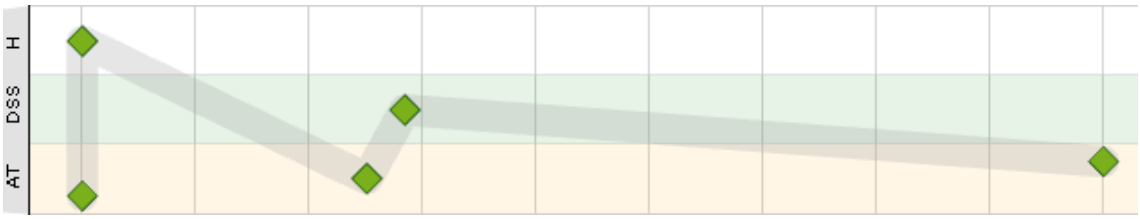
For the scenario, we furthermore chose a shorter reassignment sequence, with reassignments among 3 support departments “AT”, “DSS” and “H”. Within each of these departments support groups exist, such as

"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.

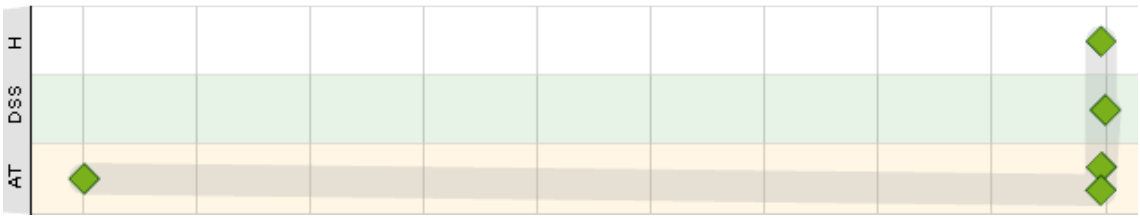
In scenario C2.a we searched the complete data set. In fact, this is not required: event sequences containing only an alert event but no ticket events because for this alert no ticket had been opened, as well as sequences with less than 2 reassignment events can be pre-filtered.

7.2.3.1 Search results and discussion

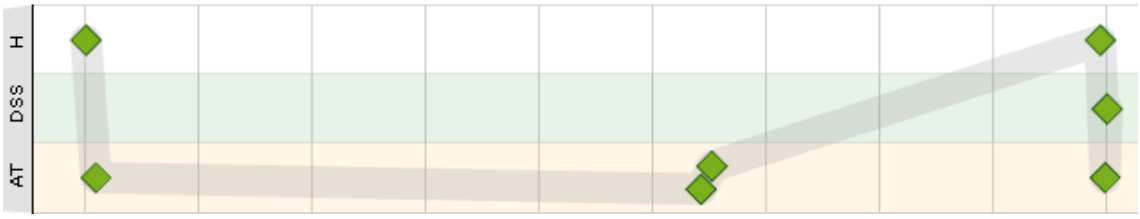
Using the above described settings to narrow the search scope, the whole searching process executes more than 10 times faster than in scenario C2.a. The results have been rather surprising: We discovered that more than 8% of all tickets had a match of 75% similarity and higher. Figure 34 depicts the 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 dimension.



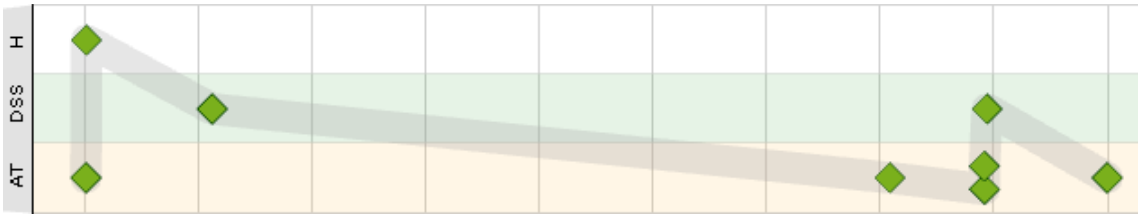
(a) – Ticket reassignments in search pattern



(b) – Match with sim=0.91



(c) – Match with sim=0.89



(d) – Match with sim=0.87

Figure 34: 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.

7.2.3.2 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 16: Performance results for evaluation scenario C2.b

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

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.

7.2.4.1 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 35 (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 35 (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.

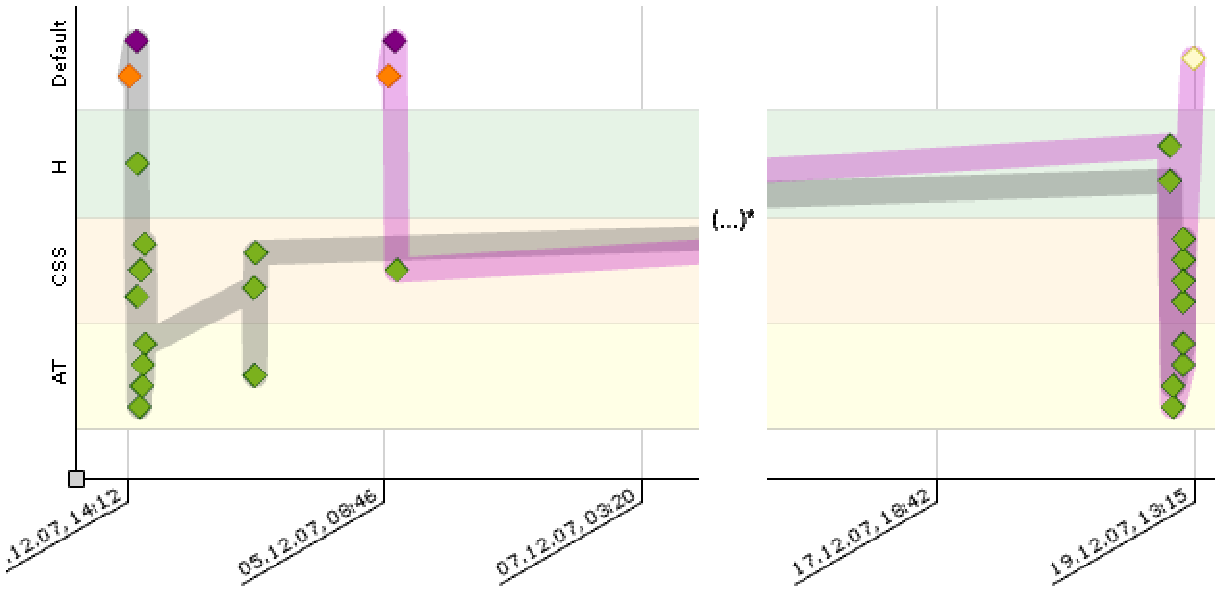


Figure 35: Search pattern and best match for evaluation scenario C2.c37

7.2.4.2 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	00:00:09.37	00:00:07.63	1077,37	39,70	1323,07	48,75

Table 17: Performance results for evaluation scenario C2.c

7.3 C3 - Credit card transaction: Sequences of purchases

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 36 shows the occurring types of events and their attributes.

[37] 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.

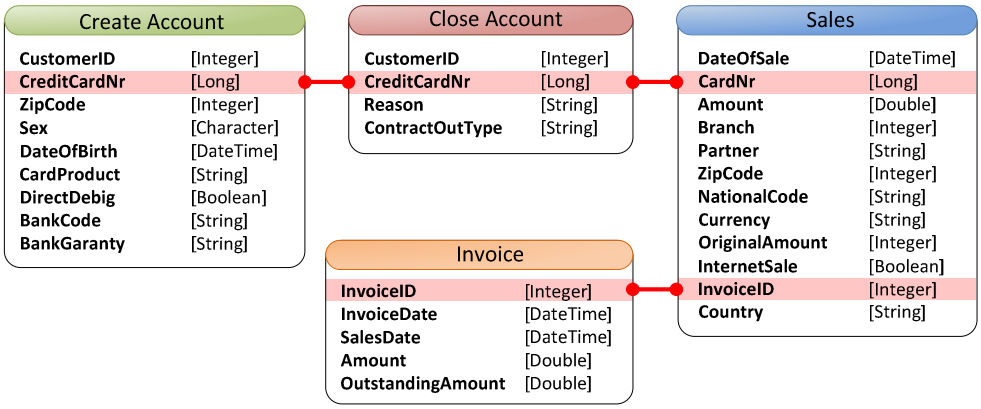


Figure 36: Event types and correlations in evaluation scenario C3 – credit card transactions

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 so, 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.[38] 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.

7.3.1 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

7.3.2 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.

7.3.2.1 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 as working with the discrete values was impossible.

[38] 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.

7.3.2.2 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 attributes functions as described in 5.4.1 can be used to extract such values on the fly.

7.3.2.3 Summary and discussion

With the use of attribute function, the effort for preprocessing is minimal in our approach. Discrete values don’t need to be categorized and attribute functions add “virtual” 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.

7.3.3 C3.b – Getting started with the mining process

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

7.3.3.1 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.

7.3.3.2 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.

7.3.3.3 Summary and discussion

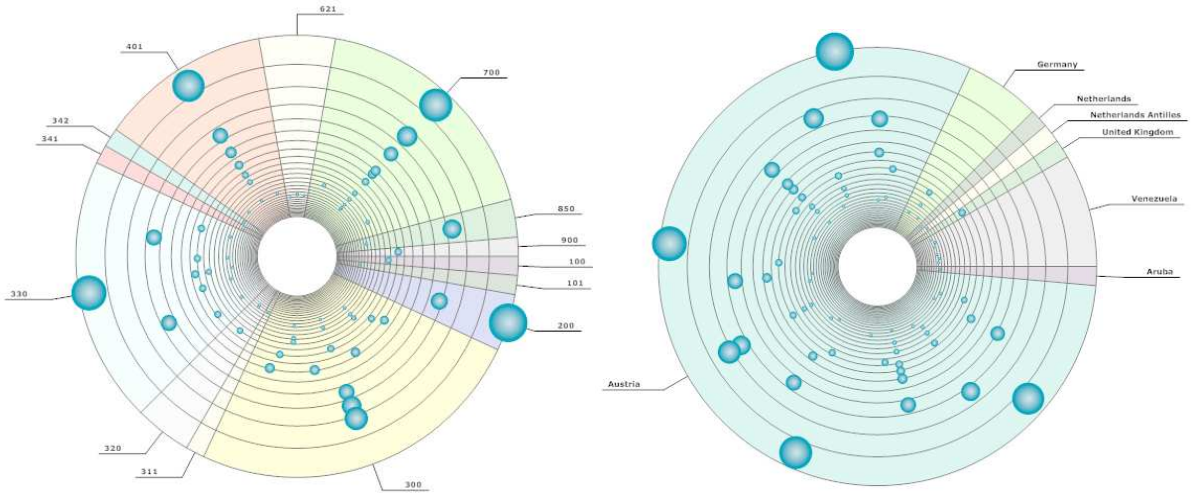
The use case shows the necessity to embed the similarity search in a greater context, for instance in the form of a clustering algorithm, which forms groups of similar sequences based on multiple similarity comparisons. As is, only a punctual search is possible. Without initial knowledge on the dataset,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

7.3.4 C3.c – Finding sequences of purchases

Finally, taken said limitation that we can only pick certain pattern sequences by chance and not automatically investigate the whole data set into account, 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, the Levenstein string similarity (which actually 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 37 shows how the sales events in the selected pattern sequence are distributed with respect to the product branch (Figure 37a) and the country (Figure 37b).



(a) (b)

Figure 37: Search pattern events for evaluation scenario C3.c

7.3.4.1 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 into account.
- The length of the event sequences in the data set varies from 10 up to 530 events. For such 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 Conclusion and future work

In the on-hand thesis, we presented flexible yet natural approaches for similarity searching in event data, both within single events and sequences thereof. Serving as a basis for our considerations, we presented a rich set of definitions and described events, event-sequences and related concept on a very high level of abstraction. Also, we gave a brief introduction on SARI, the CEP-infrastructure underlying our concrete implementation. Eventually, we evaluated our implementation in three real-world scenarios.

Our approach on single-event similarity builds upon geometric ideas of similarity, with, at least basically, event attribute values defining the relative positioning of two events, i.e., the distance between them. Without a doubt, a so-defined understanding of single-event similarity is highly intuitive and somewhat obvious, and similar approaches have been presented, for instance, in the database domain. Yet, by extending the very basic, distance-based approach on vector-similarity by two user-configurable levels of abstraction, *attribute-functions* and corresponding *attribute-level similarity-measures*, the presented approach gains a broad expressiveness and generality far beyond related concepts, and thus should be applicable in the variety of scenarios and use-cases that complex business events may occur in. Also, we found out that at the time of writing, there is no comprehensive and formally well-founded study on single-event similarity in the CEP domain. We thus understand this thesis as a first groundwork for further improvements, extensions and adaptations.

The proposed approach on event-sequence similarity can clearly be considered the key part of the on-hand thesis. Created from scratch instead of setting up on an existing, less powerful approach on event-sequence similarity, it builds upon an assignment-based understanding of sequence similarity were certain units from the target sequence are considered to represent the units in the pattern sequence. Despite of an undeniable degree of complexity, it should appear somewhat natural to business analysts. As before for single-event similarities, the proposed approach on event-sequence similarity focuses on highest generality and flexibility. The conceptually clear “separation of concerns” - into possible assignments (compatibilities), sources of costs (cost-functions) and impact of costs (weights) - reflects this, and should be a solid basis for further extensions and improvements of the algorithm. Unfortunately, the remarkably high generality of the proposed algorithm requires lots of configuration and difficult “fine-tuning”, and thus the extensive involvement of the domain expert. Also, the relative simplicity of the proposed Branch-&-Bound strategy comes to the expense of performance.

Generally, the above characteristics were confirmed in the evaluation part of this thesis where we applied the algorithm in three real-world scenarios. We figured out that the algorithm performs well (regarding both the performance and the accuracy of results) for short and sharp-edged sequences where a majority of events constitute clear and significant characteristics of the event-sequence, e.g., instances of clearly defined business processes. Here, both the selection of cost-functions and the weighting process can be performed in a straightforward manner, evaluated and improved if necessary. The proposed algorithm fails, however, for long and noisy event sequences with no or very fuzzy structures. In such cases, it might be extremely difficult if not impossible for a business analyst to clearly accentuate those aspects he or she considers important over the sheer mass of events completely irrelevant for his or her certain interest. Also, as the current algorithm considers mappings regardless of whether they are weighted zero and thus have no impact on the overall costs of a solution, the algorithm executes impracticable slow even if there are only a few events that are actually relevant for the similarity computation.

It is easy to see that there is still plenty of room for further improvements of the proposed approach on event-sequence similarity. In his thesis [49], Suntinger presents *constraint blocks*, a particularly interesting extension of the proposed algorithm allowing analysts to further specify those aspects of a pattern-sequence that are relevant for event-sequence similarity. An analyst might, for instance, define that a number of pattern-sequence events must occur in the correct order in the target sequence. Also, Suntinger presents the seamless integration of a time-series similarity algorithm, addressing scenarios where a certain event attribute's values form a time series across sets of succeeding events.

Further research effort should, however, be spent in improvements of the base algorithm. One of the most promising ideas is that of adapting the algorithm towards Dynamic Programming. Yet, even for the existing Branch & Bound strategy, several performance improvements should be possible: One might, for instance, consider the preferred evaluation of those aspects that are considered most relevant by the business analyst, i.e., weighted heavily, so that weak solutions can be omitted earlier in the calculation process. As yet, a notable fall in performance occurs when those mappings that are considered most relevant occur at the very end of a pattern-sequence. Also, a variety of pre-processing steps might be valuable in order to detect those target-sequences that are *guaranteed* to be dissimilar before starting the actual algorithm. It might be valueable, for instance, to derive a certain maximum size from the given pattern-sequence that no "similar" target-sequence may exceed. Heuristic methods and their combination with the proposed base algorithm would make another interesting starting point for future work.

Besides performance issues, a second notable shortcoming is that of the enormous configuration effort required from the business analyst. In future, one therefore might consider the automatic derivation of some kind of default configuration, e.g., a suitable initial threshold and meaningful weights, from the given pattern-sequence. Without a doubt, such "self-calibration" of the proposed algorithm requires deep knowledge in statistical data mining techniques and would be a research project on its own.

Finally, let us say that on-hand thesis lacks a detailed, both quantitative and qualitative comparison of the proposed algorithm with existing yet less flexible approaches as, most prominently, presented by Moen [35] and Mannila and Seppänen [33]. For future publications on the proposed algorithm, such comparative kind of evaluation will of course be essential. Moreover, it might be highly interesting to find out whether an edit-distance-based approach as presented by Moan [35] can be extended by additional edit-operations in order to gain the expressiveness of the on-hand approach.

Index of figures

Index of tables

Index of algorithms

Bibliography

[1] Abbot A., Tsay A.: Sequence Analysis and Optimal Matching Methods in Sociology. In: *Sociological Methods & Research*, 29(1). 3 – 33, 2000.

[2] Altschult S. F., Gish W., Miller W., Myers E. W, Lipman D. J.: Basic Local Alignment Search Tool. In: *Journal of Molecular Biology*, 215(3). 403 – 410, 1990.

[3] Anderberg M. R.: Cluster Analysis for Applications. Academic Press, 1973.

[4] Agrawal R., Faloutsos C., Swami A. N.: Efficient Similarity Search in Sequence Databases. In: *Proceedings of the 4th international Conference on Foundations of Data Organization and Algorithms*. 69 – 84, 1993.

[5] Agrawal R., Srikant, R.: Mining Sequential Patterns. In: *Proceedings of the 11th International Conference on Data Engineering.* 3 – 14, 1995.

[6] Asarin E., Caspi P., Maler O.: Timed Regular Expressions. In: *Journal of the ACM*, 49(2). 172 - 206, 2002.

[7] Barron F. H., Barrett B. E.: Decision Quality Using Ranked Attribute Weights. In: *Management Science*, 42(11). 1515 – 1523, 1996.

[8] Bruno N., Chaudhuri S., Gravano L.: Top-*k* Selection Queries over Relational Databases: Mapping Strategies and Performance Evaluation. In: *ACM Transactions on Database Systems*, 27(2). 153 – 187, 2002.

[9] Chapman S.: String Similarity Metrics for Information Integration. University of Sheffield, 2006.[39]

[10] Das G., Gunopulos D., Koudas N., Tsirogiannis D.: Answering top-k queries using views. In: *Proceedings of the 32nd international Conference on Very Large Data Bases*. 451 – 462, 2006.

[11] Das G., Gunopulos D., Mannila H.: Finding Similar Time Series. In: *Proceedings of the 1st European Symposium on Principles of Data Mining and Knowledge Discovery*. 88 – 100, 1997.

[12] Dey D., Sarkar, S, De P.: A Distance-Based Approach to Entity Reconciliation in Heterogeneous Databases. In: *IEEE Transactions on Knowledge and Data Engineering*, 14(3). 567 – 582, 2002.

[13] Donjerkovic, D., Ramakrishnan, R.: Probabilistic Optimization of Top N Queries. In: *Proceedings of the 25th International Conference on Very Large Data Bases.* 411 – 422, 1999.

[14] Eckenrode, R. T.: Weighting Multiple Criteria. In: *Management Science*, 12(3). 180 – 192, 1965.

[15] Flesca S., Furfaro F., Greco S.: Weighted Path Queries on Semistructured Databases. In: *Inf. Comput.*, 204(5). 679 – 696, 2006.

[16] Gassman B., Herschel G., Bitterer A., Richardson J., Chandler N.: Cool Vendors in Business Intelligence and Performance Management. Gartner Research, 2008.

[17] Giegerich R.: Sequence Similarity and Dynamic Programming. Lecture Notes. BISS Bioinformatik Sommerschule, Universität Tübingen, 2002.[40]

[18] Goodall C.: Data analysis by regular expressions and the analysis of event sequences. In: *Bioinformatics, Images, and Wavelets*. 87 – 88, 2004.

[19] Gower J. C.: A General Coefficient of Similarity and Some of Its Properties. In: *Biometrics*, 27(4). 857 - 871, 1971.

[20] Gu L., Baxter R., Vickers D., Rainsford C.: Record Linkage: Current Practice and Future Directions. Technical report. CMIS Technical Report No. 03/83, 2003.

[21] Hernández M. A., Stolfo S. J.: The Merge/Purge Problem for Large Databases. In: *Proceedings of the 1995 ACM SIGMOD international conference on Management of Data*. 127 – 138, 1995.

[39] http://www.dcs.shef.ac.uk/~sam/stringmetrics.html, 2009/01/14.

[40] www.zbit.uni-tuebingen.de/biss2002/handouts/pdf/giegerich_part1.pdf, 2009/01/11.

[22] James W.: *The Principles of Psychology*. Dover, 1890/1950.

[23] Jensen C. S., Clifford J., Elmasri R., Gadia S. K., Hayes P., Jajodia S. [eds]: A glossary of temporal database concepts. In: *sigmod*, 23(1). 52 – 64, 1994.

[24] Kirkwood C. W., Sarin R. K.: Ranking with Partial Information: A Method and an Application. In: *Operations Research*, 33(1). 38 – 48, 1985.

[25] Lee M. L., Lu H., Ling T. W., Ko Y.T.: Cleansing Data for Mining and Warehousing. In: *Proceedings of the 1^0th International Conference on Database and Expert Systems* Applications. 751 – 760, 1999.

[26] Levenstein V.I.: Binary Codes Capable of Correcting Deletions, Insertions, and Reversals. In: *Soviet Physics-Doklady*, 10(8). 707 - 710, 1966.

[27] Luckham D.: *The Power of Events.* Addison Wesley, 2005.

[28] Luckham D., Schulte R.: *Event processing glossary*.[41]

[29] Makkonen J., Ahonen-Myka H., Salmenkivi M.: Applying semantic classes in event detection and tracking. In: *Proceeding of International Conference on Natural Language Processing.* 175 – 183, 2002.

[30] Mannila H., Toivonen H.: Discovering generalized episodes using minimal occurrences. In: Proceedings of the 2nd International Conference on Knowledge Discovery and Data Mining. 146 – 151, 1996.

[31] Mannila H., Toivonen H., Verkamo A. I.: Discovery of Frequent Episodes in Event Sequences. In: Proceedings of the 1st International Conference on Knowledge Discovery and Data Mining. 210 – 215, 1995.

[32] Mannila H., Ronkainen P.: Similarity of Event Sequences. In: *Proceedings of the 4th international Workshop on Temporal Representation and Reasoning.* 136 – 139, 1997.

[33] Mannila H., Seppänen J. K.: Finding similar situations in sequences of events via random projections. *1st SIAM International Conference on Data Mining*. 2001.

[34] Meng X., Jiang G., Zhang H., Chen H., Yoshihira, K.: Automatic Profiling of Network Event Sequences: Algorithm and Applications. In: *Proceedings of the 27th Conference on Computer Communications.* 266 – 270, 2008.

[35] Moen P.: Attribute, Event Sequence, and Event Similarity Notions for Data Mining. PhD Thesis. Department of Computer Science, University of Helsinki, 2000.

[36] Monge A. E., Elkan C.P.: The field matching problem: Algorithms and applications. In: *Proceedings of the 2nd International Conference on Knowledge Discovery and Data Mining 1996.* 267 – 270, 1996.

[37] Motro A.: *VAGUE: A User Interface to Relational Databases that Permits Vague Queries*. In: ACM Transactions on Information Systems, 6(3). 187 – 214, 1988.

[38] Philips L.: The double metaphone search algorithm. In: *C/C++ Users Journal,* 18(6). 38 – 43, 2000.

[39] Rozsnyai S.: Efficient indexing and searching in correlated business event streams. Diploma Thesis. Vienna University of Technology, 2006.

[40] Rozsnyai S.: Managing Event Streams for Querying Complex Events. Dissertation. Vienna University of Technology, 2008.

[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] Roth H., Schiefer J., Obweger H., Rozsnyai S.: Event Data Warehousing for Complex Event Processing. Upcoming research paper, 2009.

[43] Sanfilippo L., Voorhis J.V.: Categorizing event sequences using regular expressions. In: *IASSIST Quarterly*. 21(2). 36 – 41, 1997.

[41] http://complexevents.com/?p=195, 2009/01/07.

[44] 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.

[45] Schiefer J., Seufert A., Suntinger M., Obweger H.: Modelling Relationships between Business Events. Upcoming research paper, 2009.

[46] SENACTIVE Inc.: SENACTIVE InTime™. www.senactive.com.

[47] Shepard R. N.: Toward a Universal Law of Generalization for Psychological Science. In: *Science*, 237. 1317 – 1323, 1987.

[48] Sjoberg L.: A Cognitive Theory of Similarity. In: *Goteborg Psychological Reviews*, 2(10). 1972.

[49] Suntinger M., Event-Based Similarity Search and its Application in Business Analytics. Diploma Thesis. Vienna University of Technology, 2008.

[50] Suntinger M., Obweger H., Schiefer J., Gröller M.E.: The Event Tunnel: Interactive Visualization of Complex Event Streams for Business Process Pattern Analysis. In: Proceedings of the IEEE Pacific Visualization Symposium 2008. 111 – 118, 2008.

[51] Suntinger M., Schiefer J., Roth H., Obweger H., Data Warehousing versus Event-Driven BI: Data Management and Knowledge Discovery in Fraud Analysis. In: *Proceedings of the 2nd International Conference on Software, Knowledge, Information Management and Applications*. 129 – 134, 2008.

[52] Toshniwal D. and Joshi R. C.: Finding Similarity in Time Series Data by Method of Time Weighted Moments. In: *Proceedings of the 16th Australasian Database Conference.* 155 – 164, 2005.

[53] Vecera R.: Efficient Indexing, Search and Analysis of Event Streams. Diploma Thesis. Vienna University of Technology, 2007.

[54] Winkler W. E.: The State of Record Linkage and Current Research Problems. Technical report. Statistical Research Division, U.S. Bureau of the Census, 1999.

[55] Yang, K., Shahabi, C.: A PCA-based Similarity Measure for Multivariate Time Series. In: *Proceedings of the 2nd ACM international Workshop on Multimedia Databases*. 65 – 74, 2004.

[56] Zaki M. J.: SPADE: An Efficient Algorithm for Mining Frequent Sequences. In: *Machine Learning*, 42(1/2). 31 – 60, 2001.

[57] Zimmer D., Unland R.: On the semantics of complex events in active database management systems. In: *Proceedings of the 15th International Conference on Data Engineering*. 392 – 399, 1999.

[58] Zobel J., Dart P.: Phonetic string matching: lessons from information retrieval. In: *Proceedings of the 19th Annual international ACM SIGIR Conference on Research and Development in information Retrieval*. 166 – 172, 1996.