Working Paper Series
ISSN 1170-487X

ARBITRARY BOOLEAN ADVERTISEMENTS: THE FINAL STEP IN SUPPORTING THE BOOLEAN PUBLISH/SUBSCRIBE MODEL

Sven Bittner & Annika Hinze

Working Paper: 06/2006
June 9, 2006

Arbitrary Boolean Advertisements: The Final Step in Supporting the Boolean Publish/Subscribe Model

Sven Bittner and Annika Hinze

Department of Computer Science
The University of Waikato, New Zealand
{s.bittner,a.hinze}@cs.waikato.ac.nz

Abstract. Publish/subscribe systems allow for an efficient filtering of incoming information. This filtering is based on the specifications of subscriber interests, which are registered with the system as subscriptions. Publishers conversely specify advertisements, describing the messages they will send later on. What is missing so far is the support of arbitrary Boolean advertisements in publish/subscribe systems. Introducing the opportunity to specify these richer Boolean advertisements increases the accuracy of publishers to state their future messages compared to currently supported conjunctive advertisements. Thus, the amount of subscriptions forwarded in the network is reduced. Additionally, the system can more time efficiently decide whether a subscription needs to be forwarded and more space efficiently store and index advertisements.
In this paper, we introduce a publish/subscribe system that supports arbitrary Boolean advertisements and, symmetrically, arbitrary Boolean subscriptions. We show the advantages of supporting arbitrary Boolean advertisements and present an algorithm to calculate the practically required overlapping relationship among subscriptions and advertisements. Additionally, we develop the first optimization approach for arbitrary Boolean advertisements, advertisement pruning. Advertisement pruning is tailored to optimize advertisements, which is a strong contrast to current optimizations for conjunctive advertisements. These recent proposals mainly apply subscription-based optimization ideas, which is leading to the same disadvantages.
In the second part of this paper, our evaluation of practical experiments, we analyze the efficiency properties of our approach to determine the overlapping relationship. We also compare conjunctive solutions for the overlapping problem to our calculation algorithm to show its benefits. Finally, we present a detailed evaluation of the optimization potential of advertisement pruning. This includes the analysis of the effects of additionally optimizing subscriptions on the advertisement pruning optimization.

1 Introduction

Most current research activities in the area of content-based publish/subscribe (pub/sub) systems target at the extension of the main pub/sub functionalities, e.g., by integrating metadata [11, 21], by supporting higher abstraction layers for clients [1, 12], or by allowing for dynamic network reconfigurations [22]. Our work, however, rather concentrates on the basic principles and foundations of these systems. This different focus originates out of the, in our opinion, insufficiency of current solutions.

We have recently been able to show that the direct filtering on arbitrary Boolean subscriptions has advantages in respect to both the scalability and the efficiency of pub/sub services [2, 3]. In this paper, we extend our research to the advertisement-based pub/sub model. That is, this work presents the first solution to build an efficient pub/sub system supporting arbitrary Boolean subscriptions and, symmetrically, arbitrary Boolean advertisements.

Generally, advertisements foster the cooperation between publishers and subscribers. They also decrease the number of routing tables entries and thus increase the scalability of pub/sub systems, shown, e.g., in [18]. The basic idea of advertisement-based pub/sub systems is as follows: Advertisements describe the event messages that publishers will send later on; these advertisements are distributed among all brokers of the pub/sub system. Having this information about publishers, the system can then optimize the subscription forwarding process: Subscriptions are only forwarded in the direction of neighbor brokers that have previously sent an *overlapping* advertisement. Thereby, this overlapping relationship describes whether any event message described by an advertisement can fulfill a subscription.

Key factor in advertisement-based pub/sub systems is the efficient calculation of the overlapping relationships among subscriptions and advertisements, and vice versa[1]. This relationship is symmetric, i.e., if an advertisement overlaps a subscription, the subscription also overlaps the advertisement. In combination with defining subscriptions and advertisements in an arbitrary Boolean way, one universal solution for the calculation of overlappings is sufficient for both directions.

The next step after having found the means to calculate the overlappings is to develop an optimization for arbitrary Boolean advertisements. It has been proposed, e.g., in [18], to directly apply current routing optimizations for conjunctive subscriptions to advertisements. However, such an approach leads to several drawbacks, as shown in Sect. 2.2. But more importantly, these existing optimizations only work in combination with conjunctive filter expressions, which opposes the requirement of developing an optimization for arbitrary Boolean advertisements.

In this paper, we firstly propose a method to determine the overlapping relationship in advertisement-based pub/sub systems that support both arbitrary Boolean subscriptions and advertisements. Secondly, we present an optimization method for pub/sub systems that is applicable to arbitrary Boolean advertisements, advertisement pruning. The third part of this paper contains an extensive evaluation of the proposed algorithm to determine the overlappings and of the proposed advertisement pruning optimization. We use an online auction application scenario [5] throughout the whole paper to exemplify our calculation approaches as well as to generate our test settings.

The rest of this paper is structured as follows: In Sect. 2, we present and analyze related work in the area of advertisement-based pub/sub systems. Section 3 describes the semantics of event messages, and arbitrary Boolean subscriptions and advertisements in detail. These definitions are fundamental because the algorithm to determine the overlappings heavily depends on these semantics. We gradually develop this algorithm in

[1] In case of issuing new advertisements, we require to determine whether overlapping subscriptions exist. Knowledge of overlapping advertisements is needed when registering new subscriptions.

Sect. 4. Section 5 then successively proposes the advertisement pruning optimization. Our experimental evaluation is presented in Sect. 6. We finally conclude and present future work in Sect. 7.

2 Related Work

In this section, we present related approaches, and works that build the foundations for our later proposals. Section 2.1 starts with analyzing current advertisement-based pub/sub systems in general. Optimizations for advertisements are then the topic of Sect. 2.2. Section 2.3 finally reviews subscriptions pruning, which has influenced our later proposed advertisement pruning optimization and builds its basis.

We outline our algorithms on the basis of acyclic network structures (or the created minimum spanning trees for cyclic networks), as assumed by most research prototypes targeting at efficiency aspects, e.g., PADRES [17], REBECA [18], SIENA [9], and XNET [10]. In this paper, we do thus not aim at, e.g., pervasive environments [24] or automatic topology reconfigurations [22]. Instead, we assume the well-studied subscription forwarding scheme [9] in combination with the attribute/value pair pub/sub model. Extensions of our approach to P2P settings, e.g., used in [23], and other data models, e.g., XML-based [10, 24] ones, remain future work.

2.1 Current Advertisement-based Approaches

Currently, advertisements have been proposed in conjunction with some pub/sub systems. All of these systems only support conjunctive subscriptions. Advertisements are defined as conjunctions, or, even less expressive, they only specify the message type published later on. An example for this type-based approach is HERMES [23]. Pub/sub systems supporting conjunctive advertisements include A-MEDIAS [16], PADRES [17], REBECA [18], SIENA [8, 9], and the proposal in [15]. The proposed algorithms to compute the overlapping relationship, if given at all, are specialized to the restricted conjunctive forms of advertisements and subscriptions. These given algorithms cannot be applied to more expressive subscriptions and advertisements than conjunctive ones.

To only base advertisements upon the published event type is clearly less expressive than allowing publishers to further restrict their potentially sent messages by either arbitrary Boolean or conjunctive combinations of predicates. Thus, the mechanisms offered by HERMES [23] do not minimize the amount of forwarded subscriptions (and thus the computational load in brokers) to the same extend as the other types of advertisements. However, the overlapping relationship is more efficiently to calculate in this case.

It is well-known that we can convert arbitrary Boolean expressions, e.g., advertisements, to disjunctive normal forms (DNFs). Thus, if publishers would register several conjunctive advertisements, they can specify the same potential messages as in an arbitrary Boolean advertisement. However, these canonical forms are exponential in size in the worst case [19]. For arbitrary Boolean subscriptions, it has been shown that their direct support decreases the memory requirements for storing and indexing in various practical settings [2]. We will apply a similar argumentation to arbitrary Boolean advertisements in Sect. 3.4. Additionally, a system has to support several advertisements per publisher for this conjunctive approach.

Furthermore, the calculation of the overlapping relationship for arbitrary Boolean advertisements works more efficiently compared to a canonical form. This is because a Boolean algorithm needs to evaluate subexpressions, occurring multiply in the converted form, only once. Conjunctive algorithms, however, treat all advertisements independently of each other and thus create a higher system load. We further elaborate on this subject in Sect. 3.4, and practically compare conjunctive and Boolean approaches in Sect. 6.3.

2.2 Current Advertisement-based Optimizations

In the existing literature on content-based pub/sub systems, one can hardly find any optimizations that are based on the registered advertisements. Instead, routing optimizations that have been proposed for subscriptions are suggested to be applied to advertisements as well. These optimizations are subscription covering [9] and subscription merging [18]. Both subscription covering and merging (in particular the perfect merging approach) have, however, strong assumptions on the registered subscriptions, and they require similarities and relationships among these subscriptions to lead to any optimization effect (we refer to [7] for a detailed analysis and description of these drawbacks of covering and merging).

Imperfect merging does not show this strong dependency on the subscriptions and advertisements registered with the system. Additionally, it may have a higher optimization potential than perfect merging [25]. One can further improve imperfect merging for subscriptions by incorporating knowledge from advertisements, as proposed in [17]. However, we are not aware of any existing approaches that are tailored to advertisement optimizations.

These facts describe the general problem of existing advertisement optimizations: They are either independent of their application, i.e., the optimizations do not exploit whether they are applied to subscriptions or advertisements. Or, the optimizations have been specifically developed for subscriptions and cannot be successfully applied to advertisements.

As a result, meaningful evaluations of advertisement optimizations can hardly be found in the existing literature on pub/sub systems: SIENA [9] supports subscription and advertisement covering in its routing protocols. However, this work does not answer the question of the influence of advertisement covering on any system parameter. The same does hold for HERMES [23] that, however, only supports little expressive type-based advertisements.

Some other analyses of pub/sub systems consider the existence of advertisements and evaluate the influence of optimizations based on subscriptions on the routing load: REBECA [18] only analyzes the application of subscription covering and subscription merging in combination with advertisement-based subscription routing. Also the PADRES project [17], presenting a novel computation approach for covering and merging, does not consider the optimization of advertisements in its evaluation. In this paper, we present an evaluation investigating the influence of the novel advertisement pruning optimization in Sect. 6.

2.3 Review of Subscription Pruning

Our later proposed advertisement-based optimization builds on the foundations of the subscription pruning routing optimization. We thus briefly review subscription pruning here:

We have introduced and evaluated subscription pruning in [7] (we refer to this work for the full details about the optimization). Subscription pruning basically removes parts of subscription trees and is suitable to be applied for all kinds of Boolean subscriptions. The decisions about the removals of subtrees are based on heuristics [4].

One objective of subscription pruning is to decrease the memory usage for storing subscriptions by reducing the subscription complexity, i.e., by pruning parts of subscription trees. In doing so, the event filtering process also experiences efficiency gains, another objective of pruning, due to this reduced complexity. Subscription pruning, however, creates more general subscriptions and thus introduces false positives to the system, which is increasing the internal network load. These false positives are, although, not forwarded to subscribers because the local broker of each subscriber does not perform subscription pruning for its local subscriptions. Thus, subscription pruning does not affect correct event delivery.

As already mentioned, the decisions about pruning operations are based on a heuristic that is estimating the effects of prunings on the network load. This estimation is performed by taking into account the increases in selectivity induced by pruning operations, referred to as selectivity degradation. Ordering the possible pruning operations by these degradations then allows the system to firstly perform those prunings that only have little effects on the network load before executing prunings that strongly increase this load.

The estimation of selectivities is based on incoming event messages. The selectivity of predicates is directly known by counting the number of matching messages. For newly registered predicates, we can apply estimation methods similar to [14]. For arbitrary Boolean subscriptions, the selectivities are estimated based on the operators used in subscription trees [7].

3 Semantics of Events, and Arbitrary Boolean Advertisements and Subscriptions

After having presented the state-of-the-art, we now introduce our notions and definitions of event messages (Sect. 3.1), subscriptions (Sect. 3.2), and advertisements (Sect. 3.3). We need to properly define these concepts and their exact semantics because the later developed algorithms to determine the overlapping relationship and the advertisement optimization strongly depend on these definitions. To conclude this section, we show the advantages of supporting arbitrary Boolean advertisements in pub/sub services in Sect. 3.4.

3.1 Definition and Semantics of Event Messages

For event messages, we assume a well-known definition based on event types and attribute-value pairs. An event type T specifies a set of attributes a_i, $\{a_1 \ldots a_n\}$, the corresponding attribute domains $dom(a_i)$, and the supported filter functions for these attributes (to be used in advertisements and subscriptions). An event message e itself consists of an event type and a set of attribute-value pairs: $e = (T, \{(a_1, v_1), \ldots, (a_n, v_n)\})$. Event messages contain exactly one attribute-value pair for each attribute of their type. In our application scenario of online book auctions, an example event message e_1 is (for brevity, we only use a restricted set of four attributes):

$$e_1 = (\text{book}, \{(\text{title}, \mathit{Harry\ Potter}), (\text{ending}, 6h), (\text{condition}, \mathit{new}), (\text{price}, 21.00)\})$$

This event message specifies the event type “book” and describes that a new copy of a book is offered with the title “Harry Potter” and a current price of NZ$21.00. The auction for this particular item is ending within six hours.

3.2 Definition and Semantics of Arbitrary Boolean Subscriptions

A subscription S is issued by subscribers to specify their interest in event messages. Each subscription consists of an arbitrary Boolean filter expression[2] and an event type. Each variable of the filter expression is called a predicate p and is represented by an attribute-function-operand triple $p = (a, f, o)$. A predicate might refer to any of the attributes specified by the event type of the subscription.

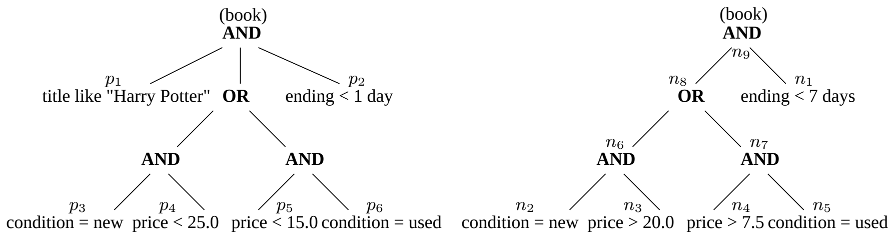
Each function f has two inputs and evaluates to a Boolean value. Its first input is an attribute value (as given by an attribute-value pair of an event), and its second input is an operand o (as given by a predicate). These operands are not restricted to values of attribute domains; for example, they may also specify a set of values. The exact definition of the permitted operands depends on the specification of function f.

We can represent the Boolean filter expression of a subscription by a tree structure [6]. Negations in filter expressions are shifted down to the leaf nodes using De Morgan’s laws. We have given an example subscription S_1 for our online book auction scenario in Fig. 1(a) (for clarity, we have specified the event type above the root—it does, although, not belong to the tree structure). It describes the interest in books whose title contains the phrase “Harry Potter”, and there is less than one day left for the auction. The subscriber wants to pay less than NZ$25.0 for new book copies and less than NZ$15.0 for already used copies. We have named the predicates of S_1 as p_1 to p_6.

An event e fulfills a subscription S if and only if S specifies the same event type as e, and the Boolean filter expression of S evaluates to *true* on event e. For this evaluation, each variable of the filter expression, i.e., a predicate $p = (a, f, o)$, gets assigned the result of the following operation: Evaluate function f with the value given in the attribute-value pair of e (referring to the same attribute) as first input and the operand o of predicate p as second input. If event e fulfills subscription S, e is referred to as an event that is matching S. Our example subscription S_1 (Fig. 1(a)) is fulfilled by event e_1, which has been given in Sect. 3.1.

Subscriptions do not need to contain predicates referring to all of the attributes specified by the event type of the subscription. Furthermore, subscriptions might contain several predicates referring to the same attribute. The semantics in this case is given

[2] We restrict our further specifications to the conjunctive, disjunctive, and negation operators. We can represent all other Boolean operators by the help of these three supported ones.



(a) Example subscription S_1 (b) Example advertisement A_1

Fig. 1. Example subscription S_1 and advertisement A_1, using our online book auction scenario; we have named the predicates of S_1 by p_1 to p_6 and the nodes of A_1 by n_1 to n_9

by the Boolean operators in the filter expression. For attributes not referred to by predicates, subscribers do not restrict the attribute value in an event message, i.e., they accept all values. Whether the incoming event is matching solely depends on the predicates used in the filter expression. This semantics is different from conjunctive approaches, e.g., [18], where all attributes have to be referred to by exactly one predicate of a filter expression. Other approaches, e.g., [13], explicitly insert *don't-care* predicates for attributes mentioned in the type of a subscription S but not in S itself. Our approach does not require this preprocessing, strongly increasing the subscription complexity.

3.3 Definition and Semantics of Arbitrary Boolean Advertisements

An advertisements A is the counterpart of a subscription and is issued by publishers. Advertisements describe the event messages publishers will send later on and need to be registered with the pub/sub system before actual messages are published. Similarly to subscriptions, advertisements consist of an event message type and an arbitrary Boolean filter expression. The filter expression is defined as in subscriptions, i.e., it contains predicates p_i linked by Boolean operators. The semantics of an advertisement A is as follows: The issuing publisher will send messages of the type given in A. For each message e that will be sent, the Boolean filter expression of A evaluates to *true* on e.

Again, there might be several predicates in the filter expression referring to one attribute. And some attributes might not be referred to by any predicate of a filter expression. For the former case, the semantics depends on the Boolean operator, e.g., a disjunctive operator of two predicates referring to one attribute a, $p_i = (a, f_i, o_i)$ or $p_j = (a, f_j, o_j)$, means that either f_i or f_j will evaluate to *true* for each message (also both functions could evaluate to *true* if they do not exclude each other). For the later case (attributes that are not referred to by predicates), publishers will send any values for these attributes, i.e., they do not restrict the values they will send in their messages.

We give an example of an advertisement A_1 in Fig. 1(b) (again, we specify the event type, which does not belong to the tree structure, above the root node). This advertisement A_1 specifies that the publisher will send messages describing auctions about used books that cost more than NZ\$7.50 and about new books having a price of more than NZ\$20.00. Furthermore, the auctions of this publisher last no longer than seven days. Advertisement A_1 does not restrict the title of books, i.e., its publisher might send messages of any book title. We have named all nodes of the tree structure of A_1 as n_1 to n_9. The filter expression of advertisement A_1 evaluates to *true* on event e_1 (cf. Sect. 3.1). That is, event e_1 could be sent by a publisher that has registered A_1.

3.4 Advantages of Supporting Arbitrary Boolean Advertisements

As already argued in Sect. 2, arbitrary Boolean advertisements allow publishers to more accurately specify their potentially sent event messages compared to conjunctive advertisements. Even if assuming the conversion of arbitrary Boolean advertisements to DNFs and the registration of several conjunctive advertisements instead of a Boolean one, the direct support of Boolean advertisements does still have advantages in respect to both system efficiency and scalability compared current approaches:

To determine the overlapping relationship in current conjunctive proposals, advertisements are treated individually. Each conversion to DNFs creates several advertisements out of an arbitrary Boolean one. These conjunctive advertisements share common subexpressions. However, there is no optimization in respect to this property, i.e., for these subexpressions the required calculations are multiply performed[3]. Using the arbitrary Boolean expression, (most) subexpressions occur only once. That is, a Boolean algorithm performs the calculations for these subexpressions exactly once. This beneficial effect outbalances the higher computational load for analyzing a Boolean expression compared to a conjunctive one. This advantage of using arbitrary Boolean advertisements to determine the overlapping relationship increases with a growing size of the equivalent DNF. In Sect. 6.3, we show these efficiency benefits when using arbitrary Boolean advertisements.

Next to the efficiency benefits when calculating the overlapping relationship, arbitrary Boolean advertisements require less memory for storage and indexation than the converted conjunctive ones. These memory requirements directly influence the scalability properties of the broker components of a distributed pub/sub service [2, 20]. This behavior results out of the requirement to apply pure main memory algorithms out of efficiency reasons.

The work in [2] presents an extensive evaluation of the memory requirements of arbitrary Boolean filtering algorithms and conjunctive filtering approaches. It is shown that even if subscriptions contain only one disjunctive operator, the utilization of arbitrary Boolean filtering algorithms requires less memory. Advertisements should be indexed and handled in the same way as subscriptions to allow for the efficient determination of the overlapping relationship [18]. These properties influence the memory requirements as follows:

Let us exemplarily pick the counting algorithm [26] as conjunctive approach, and use the results and notions from the analysis in [2]: To calculate the overlapping relationship, for storing and indexing conjunctive advertisements we require a hit vector, an advertisement predicate count vector (which is the counterpart to the subscription predicate count vector), a predicate advertisement association table (being the counterpart

[3] The work in [17] shares common subexpressions of conjunctive filters in subscription indexes; however, the work does not present solutions to compute the overlapping relationship among subscriptions and advertisements.

to the predicate subscription association table), and an advertisement predicate association table (counterpart to subscription predicate association table). To index and store Boolean advertisements, we require the advertisement trees (counterpart to subscription trees), an advertisement location table (counterpart to subscription location table), a predicate advertisement association table (counterpart to predicate subscription association table), a hit vector, and a minimum predicate count vector. That is, we require exactly the same structures as for the filtering of subscriptions. And these structures require the same memory as found in the analysis in [2]. Thus, its results symmetrically hold for advertisements: Directly utilizing arbitrary Boolean advertisements requires less memory for storage and indexation than the usage of the converted conjunctive advertisements.

4 Calculation of the Overlapping Relationship

We have developed an algorithm allowing for the computation of the overlapping relationships among arbitrary Boolean advertisements and subscriptions. Without loss of generality, we consider the determination of all overlapping subscriptions in the following descriptions. Finding the overlapping advertisements for a subscription works analogously (due to the symmetric definition of advertisements and subscriptions).

In the following subsection (Sect. 4.1), we present the general idea of our calculation approach. Then, we outline the important and required concept of violating predicates in Sect. 4.2. In Sect. 4.3, we gradually develop our algorithm for the determination of overlapping subscriptions for conjunctive, for disjunctive, and finally for arbitrary Boolean advertisements. A practical implementation of our approach is then described in Sect. 4.4.

4.1 General Idea of the Calculation of Overlappings

A first solution to the problem of determining the overlapping relationship could work similarly to the approaches for conjunctive advertisements and subscriptions, e.g., as sketched in [18]. These solutions, in turn, are related to the event filtering algorithms for conjunctive subscriptions, e.g., introduced in [26]: Let us firstly determine all predicates of subscriptions that overlap the predicates of the advertisement. Secondly, we can derive whether a subscription is overlapping based on the overlapping information about its predicates: For conjunctive subscriptions, this is a counting of the number of overlapping predicates. And for arbitrary Boolean subscriptions, we can base this decision on the minimally required number of fulfilled predicates [2] to determine the candidate subscriptions and on the evaluation of the subscription trees of these candidates.

However, a closer at the semantics of arbitrary Boolean subscriptions and advertisements reveals the inapplicability of such an approach: An advertisement A does not need to contain predicates for all attributes of its type. Thus, a subscription S might overlap advertisement A even if not all predicates of S overlap a predicate of A. That is, a calculation based on overlapping predicates leads to a wrong answer if an attribute is not used in A. Hence, the calculation of the overlapping subscriptions based on their overlapping predicates does not work correctly for arbitrary Boolean advertisements.

Approaching the overlapping problem from the opposite direction, although, leads to the correct result: Only the attributes referred to by the predicates of advertisements are restricted in conforming event messages. And a subscription S only restricts an attribute value if S contains predicates referring to this attribute. Thus, we should compute all those predicates of subscriptions that are *not overlapped* by an advertisement. Attributes a_i referred to in a subscription but not in an advertisement A do not cause problems because the publisher does not restrict the values of a_i. Thus, these attributes will be fulfilled by all event messages conforming to A.

In the following, we refer to non-overlapping predicates as *violating predicates*. We describe their calculation in the next subsection. In the further, remaining subsections, we then introduce how to calculate the overlapping subscriptions purely based on these violating predicates.

4.2 Definition of Violating Predicates

We have given a leaf node n_i of an advertisement tree that contains the predicate $p_i = (a_i, f_i, o_i)$. The violating predicates $P_{vio}(n_i)$ of n_i are all those predicates $p_j = (a_j, f_j, o_j)$ (used in subscriptions) that refer to the same attribute of the same event type, and there exists no attribute value that will lead to a *true* result when applied to both functions given in the predicates. That is, it has to hold:

$$a_i = a_j \wedge \nexists v \in dom(a_i)(f_i(v, o_i) = f_j(v, o_j) = \textit{true}) \ .$$

We can compute the violating predicates based on the one-dimensional indexes utilized for event filtering. The calculation depends on the functions used in both predicates. For the functions EQUALS, NOTEQUALS, GREATERTHAN, and LESSTHAN, we exemplarily give the rules for the determination of violating predicates in Table 1. Here, we assume the attribute domain as totally ordered set (e.g., integers). The first column contains the function f_i used in a predicate p_i of a leaf node n_i of the advertisement; the second column shows the function f_j of a predicate p_j of subscriptions; the last column describes the calculation of the violating predicates $P_{vio}(n_i)$. Also for other domains and operators, e.g., strings in combination with prefix, suffix, and substring operators, we can calculate the violating predicates based on look ups in the utilized predicate indexes.

To compute the violating predicates of each leaf node n_i of an advertisement, we walk through all the indexes belonging to the attribute referred to by the predicate stored in n_i. Then, we calculate the violating predicates for each of these indexes (as described in Table 1) and finally unite our results to get the set $P_{vio}(n_i)$.

Example 1 (Determination of the violating predicates for predicates). For our example advertisement A_1 (cf. Fig. 1(b)), the violating predicates of the leaf nodes n_1 to n_5 are as follows. Here, we assume that subscription S_1 (cf. Fig. 1(a)) has been registered:

$$P_{vio}(n_1) = \varnothing, P_{vio}(n_2) = \{p_6\}, P_{vio}(n_3) = \{p_5\}, P_{vio}(n_4) = \varnothing, P_{vio}(n_5) = \{p_3\} \ .$$

Table 1. Overview of the calculation of violating predicates based on the used functions (f_i– function of the predicate $p_i = (a_i, f_i, o_i)$ used in the leaf node n_i of the advertisement, f_j– function of the predicate p_j used in subscriptions)

f_i	f_j	Calculation of $P_{vio}(n_i)$
EQUALS	EQUALS	All predicates specifying another value than o_i
EQUALS	NOTEQUALS	All predicates specifying the same value o_i
EQUALS	GREATERTHAN	All predicates specifying a greater value than o_i
EQUALS	LESSTHAN	All predicates specifying a less or equal value than o_i
NOTEQUALS	EQUALS	All predicates specifying the same value o_i
NOTEQUALS	NOTEQUALS	All predicates specifying another value than o_i
NOTEQUALS	GREATERTHAN	If o_i is the maximal value, the predicates specifying the predecessor of o_i; otherwise none
NOTEQUALS	LESSTHAN	If o_i is the minimal value, the predicates specifying the successor of o_i; otherwise none
GREATERTHAN	EQUALS	All predicates specifying a less or equal value than o_i
GREATERTHAN	NOTEQUALS	If o_i is the predecessor of the maximal value, the predicates specifying the maximal value; otherwise none
GREATERTHAN	GREATERTHAN	There are no violating predicates
GREATERTHAN	LESSTHAN	All predicates specifying a less or equal value than o_i
LESSTHAN	EQUALS	All predicates specifying a greater/equal value than o_i
LESSTHAN	NOTEQUALS	If o_i is the successor of the minimal value, the predicates specifying the minimal value; otherwise none
LESSTHAN	GREATERTHAN	All predicates specifying a greater or equal value than the predecessor of o_i
LESSTHAN	LESSTHAN	There are no violating predicates

4.3 Overlapping Subscriptions Based on Violating Predicates

In this section, we subsequently present the calculation of the violating predicates for conjunctive, disjunctive, and finally for arbitrary Boolean advertisements. Then, we show how to determine all overlapping subscriptions based on these violating predicates.

Conjunctive Advertisements. For a conjunctive advertisement A_c with root node n_c, the set of violating predicates $P_{vio}(n_c)$ is the union of the sets of predicates violating each of the child nodes (i.e., predicates) of n_c. The reason for this definition is the fact that for all event messages conforming to the conjunctive advertisement A_c, the functions given in all predicates in the leaf nodes evaluate to *true*. Thus, we need to combine the violating predicates for all k children (n_1 to n_k) of n_c:

$$P_{vio}(n_c) = \bigcup_{i=1\ldots k} P_{vio}(n_i) \enspace .$$

Disjunctive Advertisements. For a disjunctive advertisement A_d with root node n_d, we could build the intersection of all violating predicates of the children of n_d. This approach results in a set of predicates violating all predicates of the disjunction. However, the other violating predicates (those only violating one or several predicates) are neglected in this calculation. Because each disjunctive advertisement A_d contains several descriptions of event messages—of which one or several have to hold for each message—the violating predicates of A_d are in fact expressed by several predicate sets.

This characteristic contradicts our previous notion of violating predicates as a set of predicates. Violating predicates should instead be defined as a set containing sets of predicates. Each of these sets describes one of the options expressed by the disjunctive advertisement A_d with root node n_d, having k children (n_1 to n_k):

$$P_{vio}(n_d) = \{P_{vio}(n_i) | i = 1 \ldots k\} \enspace .$$

Arbitrary Boolean Advertisements. Arbitrary Boolean advertisements might contain both disjunctive and conjunctive operators. Thus, their violating predicates must also be defined as sets containing sets of predicates. We will base the calculation of the violating predicates on the operators in the nodes n of advertisement trees. In the following, we refer to this refined notion of violating predicates in combination with any node n of advertisement trees as $P'_{vio}(n)$. All elements in $P'_{vio}(n)$ describe a set of violating predicates induced by n.

For leaf nodes n_l, we look up our indexes as described in Sect. 4.2. Then, we embed the computed violating predicates of n_l in a set to obtain our refined notion:

$$P'_{vio}(n_l) = \{P_{vio}(n_l)\} \enspace .$$

For a conjunctive node n_c having k children, n_1 to n_k, we unite each set of violating predicates of each child with each set of violating predicates of all other children:

$$P'_{vio}(n_c) = \{ \bigcup_{i=1\ldots k} s | s \in P'_{vio}(n_i)\} \enspace .$$

For a disjunctive node n_d having k children, n_1 to n_k, we unite the sets of all children:

$$P'_{vio}(n_d) = \bigcup_{i=1\ldots k} P'_{vio}(n_i) \enspace .$$

Recursively calculating the violating predicates of the root node of the advertisement tree of A then allows us to determine the overlapping subscriptions for A. We describe this method in the following paragraph, following this example:

Example 2 (Determination of violating predicates). Again assuming the registration of subscription S_1 (Fig. 1(a)), the violating predicates for the leaf nodes of advertisement

A_1 (Fig. 1(b)) are straightforwardly derived from our calculations in Example 1:

$$\begin{aligned}
P'_{vio}(n_1) &= \{P_{vio}(n_1)\} = \{\varnothing\},\\
P'_{vio}(n_2) &= \{P_{vio}(n_2)\} = \{\{p_6\}\},\\
P'_{vio}(n_3) &= \{P_{vio}(n_3)\} = \{\{p_5\}\},\\
P'_{vio}(n_4) &= \{P_{vio}(n_4)\} = \{\varnothing\},\\
P'_{vio}(n_5) &= \{P_{vio}(n_5)\} = \{\{p_3\}\} \ .
\end{aligned}$$

For the conjunctive nodes n_6 and n_7 of A_1, the violating predicates are as follows:

$$P'_{vio}(n_6) = \{\{p_5, p_6\}\}, P'_{vio}(n_7) = \{\{p_3\}\} \ .$$

And for the disjunctive node n_8, we derive the following violating predicates:

$$P'_{vio}(n_8) = \{\{p_5, p_6\}, \{p_3\}\} \ .$$

Finally, for the conjunctive root node n_9 the violating predicates are the same as for n_8 in this example (because it holds $P'_{vio}(n_1) = \{\varnothing\}$):

$$P'_{vio}(n_9) = P'_{vio}(n_8) = \{\{p_5, p_6\}, \{p_3\}\} \ .$$

Determination of Overlapping Subscriptions. We now describe how to determine the overlapping subscriptions based on the calculated violating predicates. We require two steps for this determination: In the first step, we determine all *candidate overlapping subscriptions*. This set excludes all those subscriptions that will definitely not overlap the given advertisement. The calculation of each candidate overlapping subscription S is purely based on the number of violating predicates per subscription, $|P_{vio}(S)|$. In the second step, we then evaluate all candidates to determine whether their subscription tree might still evaluate to *true* if all violating predicates evaluate to *false*.

We can base the calculation of the candidate overlapping subscriptions on the minimally required number of fulfilled predicates $|P_{min}(S)|$ per subscription S [2, 3]. This subscription-specific property is required in the event filtering step and thus already known for all indexed subscriptions. The value $|P_{min}(S)|$ specifies the minimal number of predicates evaluating to *true* if a subscription S is fulfilled by an incoming event message, e.g., for a conjunctive subscription, $|P_{min}(S)|$ is the total number of predicates, $|P(S)|$. For each candidate overlapping subscription S, the following property has to hold:

$$|P(S)| \geq |P_{vio}(S)| + |P_{min}(S)| \ .$$

This inequality describes that a subscription tree can still evaluate to *true* even if all violating predicates do not overlap an incoming advertisement (i.e., evaluate to *false*).

To determine whether a candidate is an overlapping subscription, we evaluate its subscription tree. We do not need to evaluate the functions of predicates because we already know whether predicates overlap the advertisement: For violating predicates, we assume the result of the function of the predicate as *false*. All other predicates are assumed to evaluate to *true*. If the whole subscription tree might still result in *true*, the subscription overlaps the advertisement. The reason for this is that violating predicates are never fulfilled by an event message e conforming to the advertisement. The other predicates, however, might be fulfilled, depending on the values used in e.

This calculation is performed for all elements in $P'_{vio}(n)$, i.e., all predicate sets, with n specifying the root of the advertisement tree. We illustrate this in an example:

Example 3 (Determination of overlapping subscriptions). For the root node of advertisement A_1 (Fig. 1(b)), we have already determined the set of violating predicates (cf. Example 2):

$$P'_{vio}(n_9) = \{\{p_5, p_6\}, \{p_3\}\} \ .$$

For our subscription S_1 (Fig. 1(a)), it holds:

$$|P(S_1)| = 6, \ |P_{min}(S_1)| = 4 \ .$$

Thus, subscription S_1 is a candidate subscription for both elements in $P'_{vio}(n_9)$ because it holds $6 \geq 2 + 4$ (for set $\{p_5, p_6\}$) and $6 \geq 1 + 4$ (for set $\{p_3\}$). S_1 is in fact an overlapping subscription because its subscription tree can result in *true* if p_5 and p_6, or p_3 is *false* (it is sufficient if this is the case for either one of these predicate sets).

4.4 Practical Implementation of the Calculation Approach

We have integrated our described approach of determining the overlapping relationship in our prototype of a distributed pub/sub system, implemented in C/C++.

For the determination of the violating predicates of an advertisement, we use the existing implementation of a predicate bit vector, already used as fulfilled predicate vector in the filtering process [2]. For leaf nodes, we determine the violating predicates using the one-dimensional index structures and store them in the predicate bit vector. Currently, we use the STL map class for the realization of our one-dimensional indexes and support the type integer. For the calculation, disjunctive nodes store the bit vectors for all their children in an array. Conjunctive nodes determine the violating predicates for their first child and then subsequently for the other children, including the required union operation with the previous results (cf. Sect. 4.3). For pure conjunctive advertisements and conjunctive nodes involving only leaf nodes as children, we have implemented an optimization that is only using one predicate bit vector.

For each calculated predicate bit vector, we then determine the candidate overlapping subscriptions. This step uses the hit vector implementation, also required in the filtering process. For each predicate, i.e., a bit that is set in the vector, we determine all subscriptions containing this predicate (using the existing predicate subscription association table) and increase a counter in the hit vector. Having performed this step, this hit vector represents the values $|P_{vio}(S_i)|$ for all subscriptions S_i.

We then calculate the candidate subscriptions using the now known information about the number of violating predicates $|P_{vio}(S_i)|$ (just calculated), about the minimally required number of fulfilled predicates $|P_{min}(S_i)|$, and about the total number of predicates $|P(S_i)|$ (the later two are known for all indexed subscriptions).

For these candidates, we can then access their subscription trees (using the subscription location table required in the filtering process) and evaluate these trees based on the violating predicates stored in the predicate bit vector (which allows the algorithm to easily access the state of each predicate). If a tree can result in *true* with the given violating predicates, it belongs to an overlapping subscription. This decision is based on the used operators in the nodes of the tree.

5 Advertisement Pruning: An Optimization for Advertisements

After having the means to calculate the overlappings, we now proceed with developing an optimization approach for arbitrary Boolean advertisements. The general idea of advertisement pruning is as follows: The pruning algorithm is trying to decrease the complexity of advertisements, in respect to the space complexity for storage and the time complexity for processing, without, in case of advertisements, strongly increasing the amount of overlappings. In this section, we describe how to achieve these goals. We show the inapplicability of the existing subscription pruning heuristics (cf. Sect. 2.3) for advertisements in Sect. 5.1. In the remaining subsections, we then propose how to tackle the advertisement pruning problem (Sect. 5.2 to 5.4) and how to find the most suitable pruning operation among all advertisements (Sect. 5.5).

5.1 Inapplicability of the Subscription Pruning Heuristics

One could image that the heuristic to decide on the next subscription pruning operation can be directly mapped onto the advertisement pruning problem. However, subscription and advertisement pruning follow different optimization goals, and the existing heuristic is thus not the applicable:

The main focus of subscription pruning is to decrease the selectivity of subscriptions as little as possible (cf. Sect. 2.3), i.e., with each pruning operation we want to generalize subscriptions to the least extend. The determination of the selectivities of subscriptions is, obviously, based on the incoming event messages, which should match as little as possible additional subscriptions due to pruning. When applying advertisement pruning, we want to increase the amount of overlappings among subscriptions and advertisements as little as possible. This is, at a first glance, independent of the incoming event messages. Although, we can partially utilize the selectivities deduced by event messages, as shown in the following subsections.

What is required to prune advertisements is rather a correlation between subscriptions and advertisements because their overlapping relationships are altered by the pruning process (optimally, as minimal as possible). Additionally, we do not need to evaluate event messages against advertisements. Even the sole consideration of the predicates of advertisements while filtering would unneccesarily decrease the event throughput of the system. Thus, we cannot determine the selectivities of the predicates of advertisements and, consequently, not of advertisements themselves. But also when assuming these selectivities to be given, simply minimizing their degradation (cf. Sect. 2.3) when pruning does have no relation to the registered subscriptions and thus to the overlapping relationships.

Hence, advertisement pruning does require a completely different heuristic estimation approach than subscription pruning. We successively propose a possible heuristic in the following subsections. In Sect. 6, we then evaluate it in detail.

5.2 Discovering the Influences on the Overlappings

In this section, we subsequently identify the factors affecting the overlappings among subscriptions and advertisements. We then incorporate these factors into a heuristic estimating these influences in Sect. 5.3.

The algorithm to determine the overlappings uses the concept of violating predicates for its calculations (cf. Sect. 4). The number of violating predicates is then used to compute candidate subscriptions/advertisements. The less candidates exist, the more efficient the calculation of overlappings.

Hence, an advertisement pruning operation should increase the amount of violating predicates. Such an increase is, however, impossible because a pruning operation always removes some predicates and thus the corresponding violating predicates. Hence, a pruning operation needs to target at removing as little violating predicates as possible to, in turn, enlarge the number of candidates as little as possible (Influence 1).

Although, if only considering the number of candidates, we could still strongly increase the overlappings. The worst case is that before a pruning operation none of the candidates represents an overlapping, but afterwards all candidates are overlapping. Hence, we also need to take into account what violating predicates are removed due to prunings (Influence 2).

Ideally, the removed violating predicates do not influence the fact whether a subscription and an advertisement overlap. Or, to put it the other way round, the remaining violating predicates should still disqualify subscriptions and advertisements from overlapping. Hence, pruning operations must not remove predicates from advertisements that have violating predicates preventing subscriptions from overlapping. Making rational assumptions about the usage of predicates, we can base this decision on the selectivities of predicates. We elaborate on this proposal in the next subsection.

5.3 Characterizing an Arbitrary Boolean Advertisement

Having identified the influences of pruning operations on overlappings, we now develop a heuristic incorporating these factors. The first step in doing so is to quantify an *overlapping rank* for advertisements. By the help of this rank, we can then estimate the effects of pruning operations (cf. Sect. 5.4). This, finally, allows for the determination of the best among all possible pruning options (Sect. 5.5).

The overlapping rank combines the number of violating predicates of an advertisement (*quantitative overlapping rank*) with the influence of these predicates on the number of overlappings (*qualitative overlapping rank*). That is, it incorporates both of the previously identified influences (Sect. 5.2). The overlapping rank can be successively calculated for an advertisement tree based on the utilized operators, as shown in the following paragraphs.

Leaf nodes. For a leaf node n_l of an advertisement tree, the quantitative overlapping rank includes the number of violating predicates of the predicate stored in n_l and the number of subscriptions using these predicates.

For the qualitative overlapping rank, we have to make assumptions about the usage of predicates in subscriptions: Generally, subscriptions contain a number of predicates having different selectivities. For example, predicates on the *Title* or the *Author* attribute show a high selectivity (restrictive predicates), whereas predicates on the *Condition* attribute or a generally low price have a low selectivity (general predicates). In practice, there is the tendency that highly selective predicates stronger determine whether a subscription tree might be fulfilled (in respect to both matching messages and overlapping advertisements). Thus, to state it the other way round, the more selective a predicate in subscriptions, the more important its state of fulfilment.

Putting together these observations with the importance of violating predicates when evaluating candidates, we should try to remove little selective violating predicates rather than highly selective ones. The qualitative overlapping rank should thus incorporate the selectivity of violating predicates: The higher selective a violating predicate, the higher the corresponding qualitative overlapping rank.

We define the overlapping rank $rank(n_l)$ of a leaf node n_l as follows. It is always in between 0 and 1.

$$rank(n_l) = \frac{1}{|predSubAssoc|} \sum_{p_i \in P_{vio}(n_l)} \frac{predSubAssoc(p_i)}{\sqrt{matches(p_i)+1}} \ .$$

The expression in the right multiplication factor sums up the overlapping rank for all violating predicates p_i of the leaf node n_l. The elements in the sum contain the quantitative part $predSubAssoc(p_i)$, describing the number of predicate subscription associations of a violating predicate p_i, i.e., how many subscriptions contain p_i. The qualitative part is given by the denominator, stating the selectivity of p_i. The number of matchings per predicate (referred to as $matches(p_i)$) is known from the selectivity estimation for subscriptions[4].

The left coefficient of $rank(n_l)$ ensures that the overlapping rank is always between 0 and 1. The value $|predSubAssoc|$ describes the total number of predicate subscription associations. The case $rank(n_l) = 1$ occurs if a leaf node n_l has all registered predicates (from subscriptions) as violating predicates with a selectivity of zero each.

This definition of the overlapping rank assigns higher values to leaf nodes (including predicates) having a large number of violating predicates. These violating predicates are weighed according to their selectivity, i.e., their importance to disqualify a candidate from being a real overlapping.

For the Boolean operators in an advertisement tree, we only estimate the overlapping rank, as shown in the following paragraphs. This estimation $rank^{\approx}(n_i)$ for a node n_i contains three values, the *minimal possible overlapping rank*, the *average overlapping rank*, and the *maximal possible overlapping rank*:

$$rank^{\approx}(n_i) = (rank^{min}(n_i), rank^{avg}(n_i), rank^{max}(n_i)) \ .$$

[4] It can be estimated for newly registered predicates, as outlined in [7].

For a leaf node n_l, these three estimations have the same value:

$$rank^{min}(n_l) = rank^{avg}(n_l) = rank^{max}(n_l) = rank(n_l) \ .$$

We describe the calculation of the overlapping rank for leaf nodes in the following example:

Example 4 (Determination of the overlapping rank for leaf nodes). Let us again merely assume the registration of our example advertisement A_1 (cf. Fig. 1(b)) and our example subscription S_1 (cf. Fig. 1(a)). It thus holds $|predSubAssoc| = 6$, as well as $predSubAssoc(p_i) = 1$ for $i = 1 \ldots 6$. Additionally, we assume the following values for the number of matchings for some predicates of S_1:

$$matches(p_3) = 500, matches(p_5) = 1500, matches(p_6) = 2000 \ .$$

For the five leaf nodes n_1 to n_5 of A_1 then holds (we have determined the violating predicates in Example 1):

$$\begin{aligned}
rank(n_1) &= \frac{1}{6} \times 0 = 0 \\
rank(n_2) &= \frac{1}{6} \times (\frac{1}{\sqrt{2000+1}}) \approx 0.00373 \\
rank(n_3) &= \frac{1}{6} \times (\frac{1}{\sqrt{1500+1}}) \approx 0.00430 \\
rank(n_4) &= \frac{1}{6} \times 0 = 0 \\
rank(n_5) &= \frac{1}{6} \times (\frac{1}{\sqrt{500+1}}) \approx 0.00745 \ .
\end{aligned}$$

These results describe that nodes n_1 and n_4 have the least significance in determining candidate subscriptions and their state of overlapping, i.e., n_1 and n_4 do not have any violating predicates in this example. This is followed by nodes n_2 and n_3. The most significant indicator for candidate subscriptions and their overlappings is node n_5. These results align with our assumptions about predicates: The higher the selectivities of violating predicates, the more important they are for restricting the overlappings. In this case, the violating predicate of node n_5 (predicate p_3) is the most selective one.

Conjunctive nodes. For a conjunctive node n_c, we only estimate the overlapping rank. This is required due to the general lack of information about the relationships among violating predicates. This approach, more importantly, allows for a time and space efficient calculation of the required overlapping rank. This estimated overlapping includes both concepts of a rank, the qualitative and the quantitative part.

The minimal possible overlapping rank $rank^{min}(n_c)$ occurs if all violating predicates are shared among the children of the conjunctive node n_c. It is thus the maximal rank of all children.

The average rank $rank^{avg}(n_c)$ estimates a mean value for the overlapping ranks of the children. It describes the expected mean if assuming independent child nodes and an equiprobable distribution of the violating predicates of these children.

Finally, the maximal possible rank $rank^{max}(n_c)$ occurs if the violating predicates of child nodes exclude each other. It is thus at most the sum of the ranks of all children but further restricted to not increase over 1.

This leads to the following equations for the three components. The calculations are always based on the estimation of the child nodes. For brevity, here we only consider the case of two children, n_1 and n_2:

$$\begin{aligned}
rank^{min}(n_c) &= \max(rank^{min}(n_1), rank^{min}(n_2)) \ . \\
rank^{avg}(n_c) &= rank^{avg}(n_1) + rank^{avg}(n_2) - (rank^{avg}(n_1) \times rank^{avg}(n_2)) \ . \\
rank^{max}(n_c) &= \min(1.0, rank^{max}(n_1) + rank^{max}(n_2)) \ .
\end{aligned}$$

We describe the calculation of the overlapping rank for conjunctive nodes in the following example:

Example 5 (Determination of the overlapping rank for conjunctive nodes). Let us assume the setting that is given in Example 4. The calculation of the overlapping rank for the two conjunctive nodes n_6 and n_7 of advertisement A_1 is then as follows:

$$\begin{aligned}
rank^{min}(n_6) &= \max(0.00373, 0.00430) = 0.00430, \\
rank^{avg}(n_6) &= 0.00373 + 0.00430 - (0.00373 \times 0.00430) \approx 0.00801, \\
rank^{max}(n_6) &= \min(1.0, 0.00373 + 0.00430) = 0.00803, \\
rank^{min}(n_7) &= \max(0, 0.00745) = 0.00745, \\
rank^{avg}(n_7) &= 0 + 0.00745 - (0 \times 0.00745) = 0.00745, \\
rank^{max}(n_7) &= \min(1.0, 0 + 0.00745) = 0.00745 \ .
\end{aligned}$$

Disjunctive nodes. For the other kind of inner nodes, disjunctions, we also apply an estimation approach to determine the overlapping rank. It again contains the qualitative and the quantitative part because it is based on the combined rank of the children of the disjunctive node.

The minimal possible rank is described by the rank of the child node of the disjunctive node n_d that has the smallest overlapping rank. The reason is that at least this rank will occur, regardless of which part of the disjunction will lead to the overlapping.

Similar to conjunctive nodes, the average rank of a disjunction n_d treats child nodes independently of each other and assumes an equiprobable distribution of the violating predicates of children.

The maximal overlapping rank describes the situation that all children of the disjunction hold simultaneously and that their violating predicates exclude each other.

This leads to the following equations. Again, we only consider the case of two children, n_1 and n_2, here:

$$\begin{aligned}
rank^{min}(n_d) &= \min(rank^{min}(n_1), rank^{min}(n_2)) \ . \\
rank^{avg}(n_c) &= rank^{avg}(n_1) + rank^{avg}(n_2) - (rank^{avg}(n_1) \times rank^{avg}(n_2)) \ . \\
rank^{max}(n_d) &= \min(1.0, rank^{max}(n_1) + rank^{max}(n_2)) \ .
\end{aligned}$$

We describe the calculation of the overlapping rank for disjunctive nodes in the following example:

Example 6 (Determination of the overlapping rank for disjunctive nodes). Let us again assume the setting that is given in Examples 4 and 5. The calculation of the overlapping rank for the disjunctive node n_8 of advertisement A_1 is then as follows:

$$\begin{aligned} rank^{min}(n_8) &= \min(0.00430, 0.00745) = 0.00430, \\ rank^{avg}(n_8) &= 0.00801 + 0.00745 - (0.00801 \times 0.00745) \approx 0.01540, \\ rank^{max}(n_8) &= \min(1.0, 0.00803 + 0.00745) = 0.01548 \ . \end{aligned}$$

This finally leads to the overlapping rank for the root node n_9 of A_1:

$$\begin{aligned} rank^{min}(n_9) &= \max(0.00430, 0) = 0.00430, \\ rank^{avg}(n_9) &= 0.01540 + 0 - (0.01540 \times 0) = 0.01540, \\ rank^{max}(n_9) &= \min(1.0, 0.01548 + 0) = 0.01548 \ . \end{aligned}$$

Having the means to estimate the overlapping rank for arbitrary Boolean advertisements, we can now determine the effect of a pruning operation, as shown in the following subsection.

5.4 Estimating the Influences of a Pruning on the Overlappings

In the last subsection, we have defined the overlapping rank, heuristically estimating a measure for the overlappings between an advertisement and the registered subscriptions. By the help of this heuristic measure, we can now estimate the influence of an advertisement pruning operation.

The question to be answered is as follows: Given a set of registered advertisements, what is the order of the pruning operations to perform. That is, we firstly have to determine the best pruning of each advertisement. Secondly, we need to be able to compare pruning operations of different advertisements to each other.

As identified previously, each performed pruning operation should minimally influence the amount of overlapping subscriptions for the pruned advertisement. Because our overlapping rank estimates this relationship among advertisements and subscriptions, a pruning should minimally change, i.e., decrease, the overlapping rank.

To describe the influence of prunings, we should use a relative rather than an absolute measure. This helps to weigh a change, e.g., of 0.1, higher for an existing small overlapping rank than for a large one. That is, if there is only a small amount of violating predicates, the influence of removing some of them on the overlappings is higher than for removing the same number of predicates from an overall large number of violating predicates.

We refer to this influence of a pruning of advertisement A_i to A_j as *overlapping rank degradation, $\Delta rank(A_i, A_j)$*. It is defined as follows (the rank of an advertisement A_k always equals the rank of its root node n_k, e.g., $rank^{min}(A_k) = rank^{min}(n_k)$):

$$\Delta rank(A_i, A_j) = \max(\frac{rank^{min}(A_i) - rank^{min}(A_j)}{rank^{min}(A_i)},$$

$$\frac{rank^{avg}(A_i) - rank^{avg}(A_j)}{rank^{avg}(A_i)},$$
$$\frac{rank^{max}(A_i) - rank^{max}(A_j)}{rank^{max}(A_i)}) \ .$$

This definition weighs the change in the overlapping rank relatively to the existing rank before pruning and thus fulfils our requirements. The final step is now just to relate all prunings with each other to determine the order of prunings, as shown in the following subsection. Beforehand, we give an example of calculating the overlapping rank degradation:

Example 7 (Calculation of the overlapping rank degradation). We again assume the setting that is given in Examples 4 to 6. The original advertisement A_1 with root node n_9 has led to the following estimated overlapping rank (cf. Example 6):

$$rank^{\approx}(A_1) = (0.0043, 0.0154, 0.01548) \ .$$

Lets us assume the removal of node n_1 of A_1, leading to A_2. The root node of A_2 is n_8, describing the same subtree as in A_1. It thus holds $rank^{\approx}(A_2) = rank^{\approx}(n_8)$ but also $rank^{\approx}(n_8) = rank^{\approx}(n_9)$ (cf. Example 6). This leads to:

$$\Delta rank(A_1, A_2) = \max(0, 0, 0) = 0 \ .$$

Removing node n_5, which is resulting in A_3, leads to the following:

$$rank^{\approx}(A_3) = rank^{\approx}(n_8) = rank^{\approx}(n_9) = (0, 0.00801, 0.00803) \ .$$

This results in an overlapping rank degradation $\Delta rank(A_1, A_3)$ of:

$$\Delta rank(A_1, A_3) = \max(1.0, 0.48, 0.48) = 1.0 \ .$$

Another pruning option, the removal of n_3 that is resulting in A_4, leads to this estimation:

$$rank^{\approx}(A_4) = rank^{\approx}(n_8) = rank^{\approx}(n_9) = (0.00373, 0.01152, 0.01118) \ .$$

The overlapping rank degradation $\Delta rank(A_1, A_4)$ is then:

$$\Delta rank(A_1, A_4) = \max(0.13, 0.25, 0.28) = 0.28 \ .$$

Hence, if only assuming these three pruning options, the pruning of n_1 does not have any influence on the overlappings. This is followed by the pruning of n_3 and n_5.

5.5 Determining the Best Pruning and Pruning in Practice

Our measure for the overlapping rank degradation allows us to find the best pruning operation for each advertisement A: We just need to calculate all possible pruning operations and compare their degradations with each other. The pruning that is leading to the least degradation should be performed for A.

Analogous to subscription pruning, the only valid pruning operation is to remove a child of a conjunctive node in an advertisement tree because it creates a more general advertisement, potentially increasing the overlappings. Removing the child of a disjunction, conversely, results in a more restrictive advertisement. This pruning operation would thus reduce the amount of overlappings and introduce false negatives, opposing the pruning idea. For the last option, removing the root node, the semantics is that all subscriptions would overlap the completely removed advertisement, opposing the overall application of advertisements.

Practically, whenever an advertisement A_i is registered we calculate the best possible pruning operation leading to A_j, i.e., the pruning that is resulting in the least degradation $\Delta rank(A_i, A_j)$. We then insert the tuple $(\Delta rank(A_i, A_j), A_i)$ in the *advertisement degradation queue*. This queue implements a priority queue that is ordered in an ascending way by the first element of the tuple, the overlapping rank degradation.

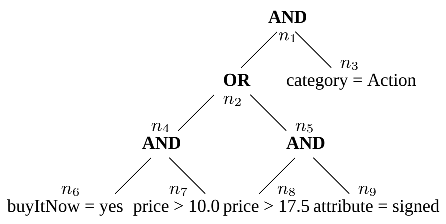
To perform a pruning, we just need to extract the top element of the advertisement degradation queue. This operation works in a constant time due to our decision about the data structure. We then perform the best pruning of the corresponding advertisement A_i, resulting in A_j, and re-insert the new best pruning of A_j into the queue. According to the goal of the optimization, e.g., to decrease the memory usage or to increase the time efficiency to calculate the overlappings, we successively perform the prunings that are stated by the top elements of the advertisement degradation queue. The pruning stops when the required optimization has been achieved, e.g., the memory usage has been reduced by a certain percentage.

The most important question emerging from our advertisement pruning proposal is the influence of prunings on the memory usage (to index and store advertisements), the efficiency to calculate overlappings (among subscriptions and advertisements), and the number of overlappings (determining the forwarding of subscriptions in the network[5]). We extensively evaluate these properties of advertisement pruning in the next section.

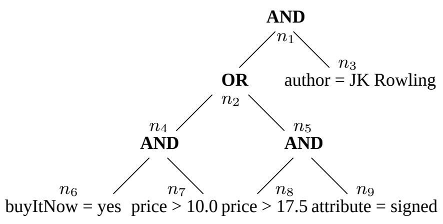
6 Experimental Analysis

In this section, we describe our evaluation of an extensive set of practical experiments we have undertaken to analyze both the algorithm to determine the overlappings and the advertisement pruning optimization. In Sect. 6.1, we characterize our experimental setup to allow for a classification of our results and the repeatability of our experiments.

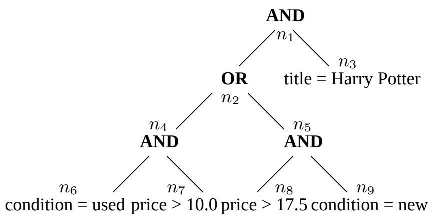
The efficiency properties of our algorithm to determine the overlappings are then analyzed in Sect. 6.2. We proceed in Sect. 6.3 by comparing the time efficiency of our calculation approach to that of a conjunctive solution. The influence of the advertisement pruning optimization on time efficiency, space efficiency, and amount of overlappings is analyzed in Sect. 6.4. Section 6.5 goes a step further and finally evaluates these three parameters when combining advertisement and subscription pruning.



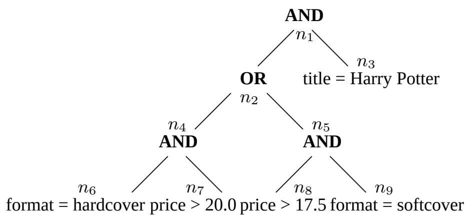
(a) Advertisement Class 1



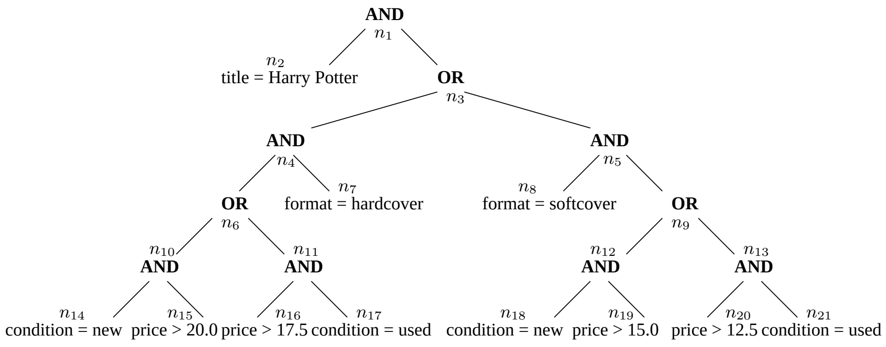
(b) Advertisement Class 2



(c) Advertisement Class 3



(d) Advertisement Class 4



(e) Advertisement Class 5

Fig. 2. Examples of advertisements of Advertisement Classes 1 to 5

6.1 Experimental Setup

We have identified eight classes of advertisements and three classes of subscriptions for our experiments. These classes of subscriptions and advertisements describe reasonable interests (subscriptions) and potential event messages (advertisements) in our online book auction scenario [5]. These classes are as follows:

Advertisement Class 1. A publisher offers books of a certain category. These books are Buy-It-Now items that are further specified by a minimal price, or signed book copies that are also stating a minimal price. We have illustrated an example advertisement of Class 1 in Fig. 2(a). It specifies category "Action", and prices of NZ$10.0 and NZ$17.5.

[5] This, in turn, also determines the memory usage for storing subscriptions and the filter efficiency.

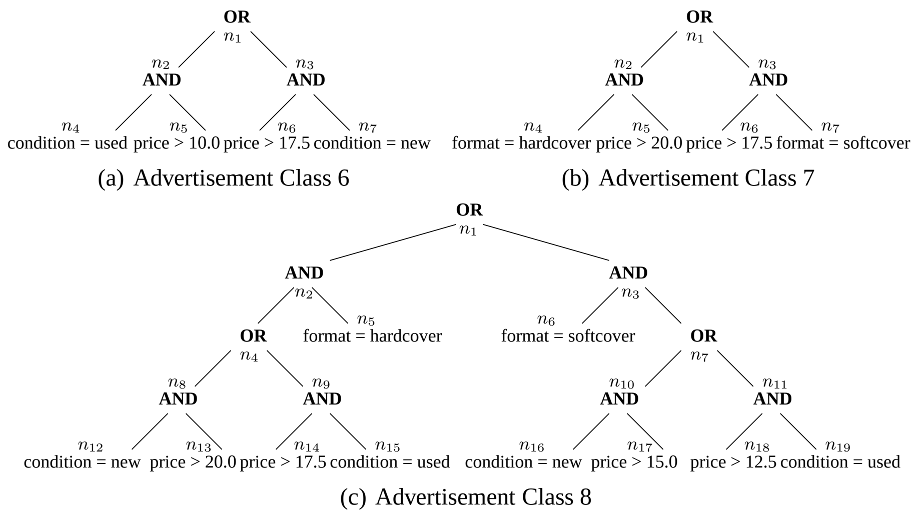


Fig. 3. Examples of advertisements of Advertisement Classes 6 to 8

Advertisement Class 2. This advertisement class is similar to Class 1 but specifies authors instead of categories. An example is given in Fig. 2(b), stating author “JK Rowling”, and prices of NZ\$10.0 and NZ\$17.5.

Advertisement Class 3. A publishers offers books of a particular book title. There are different minimal prices for used and new copies of the book. In the example in Fig. 2(c), we have chosen “Harry Potter” as title and, again, prices of at least NZ\$10.0 and NZ\$17.5.

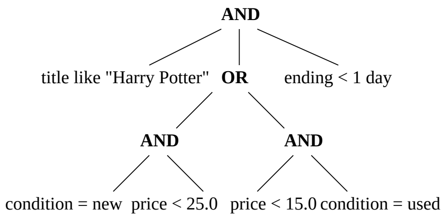
Advertisement Class 4. This class is similar to Advertisement Class 3 but specifies different prices for hardcover and softcover copies of the book. Figure 2(d) shows an example of this class stating prices of NZ\$20.0 and NZ\$17.5 for books entitled “Harry Potter”.

Advertisement Class 5. This class combines Advertisement Classes 3 and 4. That is, the publishers offers different prices for the four combinations of hardcover and softcover, as well as new and used book copies. Our example in Fig. 2(e) again refers to the title “Harry Potter” and specifies prices of at least NZ\$12.5, NZ\$15.0, NZ\$17.5, and NZ\$20.0.

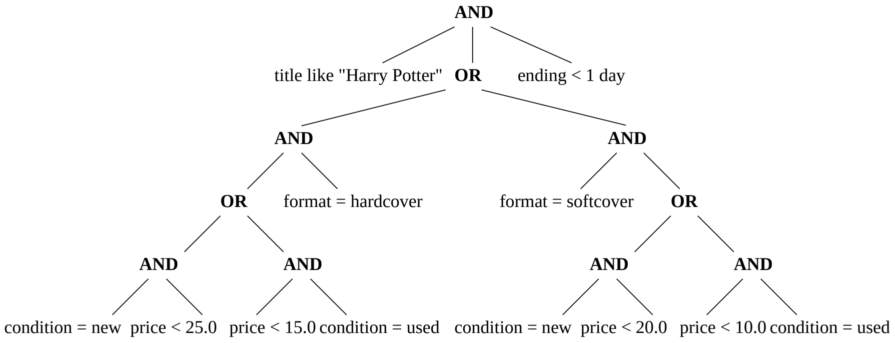
Advertisement Class 6. The publisher offers used and new books, stating a different minimal price (similar to Advertisement Class 3 except of the title). In the example in Fig. 3(a), these prices are NZ\$10.0 and NZ\$17.5.

Advertisement Class 7. This class describes a publisher offering softcover and hardcover books of different minimal prices (similar to Advertisement Class 4 except of the title). In Fig. 3(b), these prices have been chosen with NZ\$17.5 and NZ\$20.0.

Advertisement Class 8. Similar to Advertisement Class 5, the publishers specifies different minimal prices for the four combinations of used and new book copies, as well as hardcover and softcover books. There are no further restrictions on the



(a) Subscription Class 1)



(b) Subscription Class 2

Fig. 4. Examples of subscriptions of Subscription Classes 1 and 2

books. The example in Fig. 3(c) specifies prices of at least NZ$12.5, NZ$15.0, NZ$17.5, and NZ$20.0.

Subscription Class 1. A subscriber is interested in a particular book title and specifies different maximal prices for used and new copies of books. Notifications should only be sent if the auction ends in less than one day. We have illustrated an example subscription of this class in Fig. 4(a). It specifies that the book title should contain the phrase “Harry Potter”, and that the book copy should cost less than NZ$15.0 or NZ$20.0.

Subscription Class 2. In addition to Subscription Class 1, the subscriber specifies different prices for the four combinations of used and new books, as well as hard-

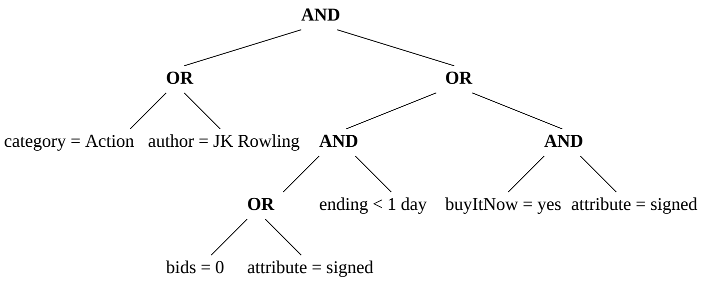


Fig. 5. Example of a subscription of Subscription Class 3

Table 2. Properties of the advertisement (abbreviated by A1 to A8) and subscription classes (abbreviated by S1 to S3)

Advertisement/subscription class	A1	A2	A3	A4	A5	A6	A7	A8	S1	S2	S3
Number of predicates	5	5	5	5	11	4	4	10	6	12	7
Number of conjunctive operators	3	3	3	3	7	2	2	6	3	10	3
Number of disjunctive operators	1	1	1	1	3	1	1	2	1	2	3
Number of conjunctions	2	2	2	2	4	2	2	4	2	4	6
Number of predicates per conjunction	3	3	3	3	4	2	2	3	4	5	3

covers and softcovers. Our example in Fig. 4(b) again expresses interest in books entitled “Harry Potter” that are offered for at most NZ$10.0, NZ$15.0, NZ$20.0, or NZ$25.0.

Subscription Class 3. A collector is interested in books from a certain author but also in books of a particular category. The books should either be signed and Buy-It-Now items, or end within one day if they are signed or have got zero bids so far. The example in Fig. 5 states the category with “Action” and the author with “JK Rowling”.

These advertisement and subscription classes represent a wide range of interests and potential event messages, and are considered as representative for our application scenario.

In the advertisement examples (Fig. 2 and 3), we have named all nodes n_i of advertisement trees (we can thus refer to them later on). In our experiments, all predicates or, more precisely, the values of all predicates of these subscriptions and advertisements are chosen randomly using a uniform distribution of reasonable values (derived from the practical analysis in [5]).

We give an overview of the properties of the advertisement and subscription classes in Table 2. This includes the number of predicates (Row 1), conjunctive operators (Row 2), and disjunctive operators (Row 3). We also show the number of conjunctive advertisements/subscriptions (Row 4) and the number of predicates per conjunctive advertisement/subscription [2] (Row 5) if performing a conversion to DNFs.

All of our experiments have been run on a machine equipped with 512 MB of main memory and a processor speed of 1.8 GHz. We have derived all results described in the following by analyzing settings involving a large number of advertisements and subscriptions, as described later on in detail. These experiments have led to negligible variances and we thus only show the derived mean values in the following tables and figures.

Table 3. Results of our evaluation of the efficiency properties of the individual subscription classes (abbreviated by S1 to S3) and the setting involving all three of them (abbreviated by S1–S3)

Parameter	S1	S2	S3	S1–S3
Time in msec (First)	$9.8 \ldots 134.8$	$4.2 \ldots 41.5$	$0.2 \ldots 5.4$	$4.9 \ldots 62.9$
Time in msec (All)	$22.2 \ldots 320.9$	$35.7 \ldots 510.4$	$16.3 \ldots 253.4$	$25.6 \ldots 374.4$
Proportion of candidates	1.29	1.85	1.36	1.5
Proportion of overlappings	0.04	0.35	1.0	0.46
$\lvert P'_{vio}(S) \rvert$	2	4	4	$\frac{10}{3}$

6.2 Efficiency of the Calculation of the Overlapping Relationship

In this section, we analyze the time efficiency of our approach to calculate the overlappings. We have evaluated all three subscription classes individually; additionally, we have analyzed a setting involving a uniform distribution of these three classes.

We consider two scenarios: In Scenario All, we determine the average time efficiency to calculate all advertisements overlapping a given subscription. Scenario First only determines whether an overlapping advertisement exists. That is, in Scenario First the algorithm stops its calculations as soon as an overlapping advertisement is found. This calculation is practically required in the registration process of subscriptions to determine whether the subscription needs to be forwarded to neighbor brokers.

In our experiments, we register an increasing number of advertisements (uniform distribution among the eight classes) with the system and analyze the time efficiency properties. We give an overview of our results in Table 3. In its columns, we present the behavior of the different classes of subscriptions; rows show the analyzed parameters.

In Row 1 of Table 3, we present the ranges of time required to determine whether an overlapping advertisement exists. We have derived these mean values by testing at least 25,000 subscriptions. This range describes the results for 20,000 up to 300,000 registered advertisements of all eight classes. The actual time per subscription increases linearly with the number of advertisements. For example, for an incoming subscription of Class 3 the algorithm requires on average 5.4 msec to determine whether it overlaps at least one of the 300,000 registered advertisements. These times stated in Row 1 of the table include all calculations that are required to determine the overlappings except of the evaluation of all candidate advertisements. The algorithm already stops its computation if one overlapping advertisement is found.

For Subscription Class 3, we can determine the existence of overlapping advertisements in the most efficient way. This is due to the fact that for this class all advertisements overlap an incoming subscription (cf. Row 4 of Table 3), i.e., the proportion of overlapping advertisements is 1.0. Hence, the calculations finish after evaluating only one or a small number of candidates. Generally, the time efficiency for this operation degrades with a decreasing number of overlapping advertisements. For Subscription

Class 1, only having overlappings for 4% of the incoming subscriptions, this operation is thus least efficient.

In Row 2 of Table 3, we show the time efficiency to determine all overlapping advertisements (again, the mean value for 25,000 subscriptions). The time per subscription does again grow linearly with the number of registered advertisements. The time values shown here include the evaluation of all candidate overlapping advertisements. The efficiency of this operation generally depends on the number of candidates required to evaluate (cf. to the value in Row 3[6]). Typically, the more candidates, the less efficient the calculation of all overlappings. However, for Subscription Class 3 (proportional number of candidates of 1.36) this operation is more efficient than for Subscription Class 1 (proportional number of candidates of 1.25). This is because the violating predicates of Subscription Class 3 occur in far less advertisements than those of Subscription Class 1; hence, the counting of the violating predicates per subscription (cf. Sect. 4.4) is performed more efficiently for Subscription Class 3—there are less counters to increase. As expected, for Subscription Class 2, having the highest number of candidates, this operation is less efficient than for the other classes.

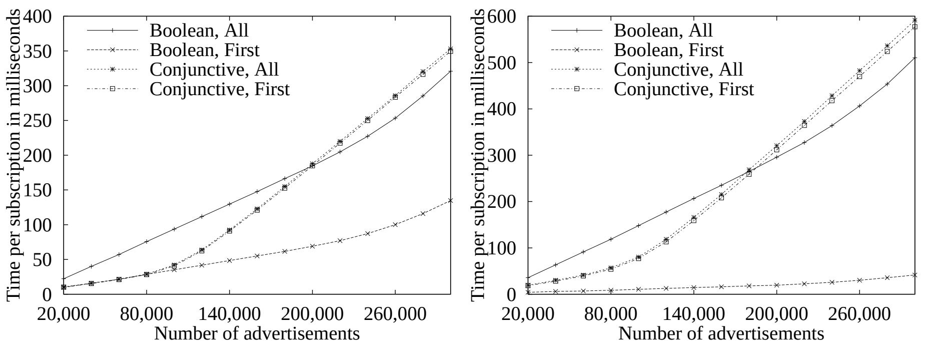
6.3 Comparison to Conjunctive Calculation Approaches

In Sect. 3.4, we have used the results of a comparison of arbitrary Boolean and conjunctive subscriptions to show the benefits in memory usage when applying Boolean advertisements. We now investigate the efficiency properties of calculating the overlapping relationship for arbitrary Boolean in comparison to conjunctive advertisements and subscriptions.

In our experiments, we have again increased the number of advertisements up to 300,000, and tested the efficiency for calculating the overlappings for all three subscription classes individually as well as for the combined setting. We show the results in Fig. 6 for Subscription Classes 1 and 2, and in Fig. 7 for Subscription Class 3 and the combined setting. We have again derived these results by analyzing 25,000 different subscriptions and taking the mean values. The abscissae represent the number of advertisements; the ordinates state the average time per subscription in milliseconds. We show four curves per diagram, describing the two scenarios (All and First, cf. Sect. 6.2) using the original arbitrary Boolean subscriptions and advertisements, and the converted conjunctive subscriptions and advertisements, respectively.

In both the conjunctive and the arbitrary Boolean case, the calculations required to determine the overlappings increase linearly with the number of registered advertisements. This behavior is due to our pattern of creating advertisements and subscriptions: On average, doubling the number of advertisements creates double the amount of overlapping advertisements. However, Fig. 6 and 7(b) appear to show another behavior of the efficiency properties, in particular for the conjunctive algorithm. We have thoroughly analyzed this development and clearly identified the counting of predicate

[6] Note that the proportional number of candidate advertisements (shown in Row 3) is greater than 1.0. This is due to the facts that the violating predicates $P'_{vio}(S)$ for each subscription S are a set, and advertisements might need to be evaluated for several elements in this set $P'_{vio}(S)$. For completeness, we show the average size of set $P'_{vio}(S)$ in Row 5 of Table 3.



(a) Subscription Class 1

(b) Subscription Class 2

Fig. 6. Comparison of the efficiency of Boolean and conjunctive approach using Subscription Classes 1 and 2

occurrences in combination with the processor cache as its reason. We can observe the same effect in the Boolean algorithm; however, it appears at a much larger number of advertisements.

The reason for this behavior is as follows: As already stated, the amount of calculations increases linearly with a growing number of registered advertisements. The calculation algorithm (cf. Sect. 4.4) uses a counter per advertisement to sum up the number of violating predicates. The more advertisements we register, the more counters do not fit into the processor cache and lead to cache misses if increased. Thus, the overall calculation time increases superlinearly even if the calculations themselves only increase linearly. In the conjunctive algorithm, a large number of cache misses happens at a much smaller number of registered advertisements due to the required conversion to conjunctions. Subscription Class 3 (Fig. 7(a)) is less influenced by this behavior because (a) the number of violating predicates, leading to increasing the counters, is much smaller, (b) the violating predicates only occur in a subset of all registered advertisements, and (c) the violating predicates are not distributed among the converted conjunctive subscriptions.

This effect of accessing and incrementing the counters leads to an increasing gradient in the curves representing the calculation times. For example, in Fig. 6(a) the maximal gradient is reached at approximately 150,000 advertisements in the conjunctive case. Further increasing the number of advertisements then linearly increases the calculation time.

Comparing the Boolean to the conjunctive algorithm leads to the following observations: For all subscription classes, the determination whether overlappings exist (Scenario First) is much more efficiently to calculate using the Boolean algorithm. The computation is 3, 14, and 13 times more efficient for 300,000 registered advertisements. This calculation is always more efficient for Subscription Classes 2 and 3 (Fig. 6(b) and Fig. 7(a)). For Subscription Class 1 (cf. Fig. 6(a)), the conjunctive and the arbitrary Boolean approach do not show efficiency differences for less than 80,000 advertise-

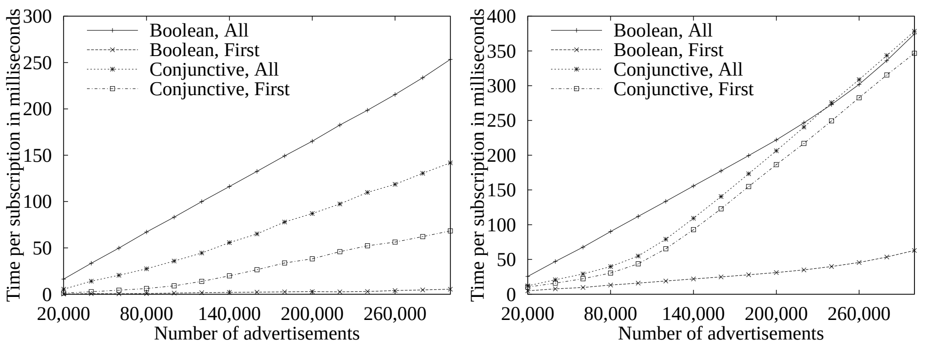


Fig. 7. Comparison of the efficiency of Boolean and conjunctive approach using Subscription Class 3 and Subscription Classes 1 to 3

ments; from approximately this point on, the Boolean solution has increasing efficiency benefits compared to the Boolean one.

For Scenario All, the Boolean algorithm is more efficient for Subscription Classes 1 and 2 (cf. Fig. 6(a) and 6(b)) in case of high advertisement numbers. Less registered advertisements favor the conjunctive solution. The reason for this changing is the cache behavior, as explained before. The conjunctive solution is always more efficient for Subscription Class 3 (Fig. 7(a)) for up to the maximal number of tested advertisements. This is also due to the cache behavior in combination with the properties of the violating predicates for this class.

The difference in efficiency between Scenario First and Scenario All depends on the number of overlapping advertisements, shown in Row 4 of Table 3 for the Boolean approach. Thus, this difference is the least for Subscription Class 1; it is followed by Subscription Class 2 and the setting containing all three classes; the difference is the most for Subscription Class 3.

Concluding these experiments, we can state that a Boolean solution is always more suitable to determine whether overlappings exist at all. This calculation is the practically required one of the two analyzed scenarios. If we need to calculate all overlappings, the more efficient approach depends on the properties of advertisements and subscriptions.

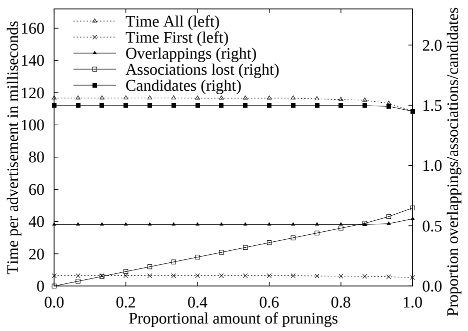
6.4 Analysis of Advertisement Pruning

We have analyzed the influence of advertisement pruning on all eight advertisement classes individually and on a combination of these classes in order to understand the effects of pruning in a wide range of settings. In this series of experiments, we have registered 100,000 subscriptions and 50,000 advertisements. We graphically present the results for the individual classes in Fig. 8 to 11; Figure 12 shows the effects on the combined setting involving all eight classes. We have always registered a combination of subscriptions of all three subscription classes in our experiments.

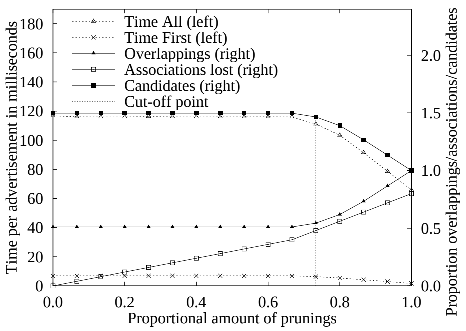
We show the proportion of advertisement pruning operations on the abscissae of all figures. They range from the value of 0, describing the situation without pruning, to the value of 1, stating that all possible pruning operations have been performed, i.e., another pruning would remove a complete advertisement. The left ordinate shows the time efficiency, i.e., the average time to calculate the overlappings per advertisement. We show two curves, one curve describing the time that is required to determine all overlapping subscriptions (Scenario All) and one curve testing whether at least one overlapping subscription exists (Scenario One). The right ordinate states the proportion of overlappings that exists, the proportion of removed predicate advertisement associations (our measure for memory usage), and the proportion of candidate subscriptions for a given amount of prunings (abscissa). These properties are represented by three individual curves.

The ideal result for an optimization is an increasing time efficiency (less time at the left ordinate), a decreasing memory usage (increasing proportion of removed predicate advertisement associations at the right ordinate), and a constant amount of overlappings (stable proportion of overlappings at the right ordinate) with an increasing amount of performed pruning operations (abscissa). This ideal behavior does, apparently, not hold in practice. There, we rather target at only slightly increasing the overlappings, and simultaneously decreasing both the memory requirements and the time to calculate the overlappings.

In some of our settings, we experience a cut-off point. At this point, the behavior of advertisement pruning changes from a worthwhile optimization to less valuable one, i.e, the amount of overlappings increases rather strongly. We have given these cut-off points in the figures representing these settings (Fig. 8(b), 10(b), 11(b), and 12). For the eight advertisement classes, our evaluation of advertisement pruning is described in the following paragraphs.



(a) Advertisement Class 1



(b) Advertisement Class 2

Fig. 8. Influence of advertisement pruning on Advertisement Classes 1 and 2

Advertisement Class 1. For Advertisement Class 1 (Fig. 8(a)), the pruning optimization decreases the memory requirements by 65% after performing all prunings (removal of proportionally 0.65 predicate advertisement associations, shown at the right ordinate). Simultaneously, determining whether any overlapping exists (Scenario One) is performed 18% faster (the time is reduced from 6.4 to 5.2 msec, illustrated at the left ordinate); the calculation of all overlappings (Scenario All) works 7% more efficient (i.e., the time is decreased from 117 to 109 msec per advertisement, as shown at the left ordinate). These beneficial results are obtained by only increasing the amount of overlappings from 51% to 56% (right ordinate) of all registered subscriptions.

The reason for this advantageous behavior is the initial pruning of predicates on the attributes *Buy It Now* (node n_6, cf. Fig. 2(a)) and *Special Attribute* (node n_9), having merely significance in subscriptions of Class 3. But even for this class, the removal of these predicates does not change the overlapping relationships. The predicates on the attribute *Category* (node n_3) are pruned later on, followed by some disjunctive nodes (node n_2), which only have predicates on *Price* (Nodes n_7 and n_8) as child nodes anymore. This pruning of predicates on *Price* then creates the little increase in overlappings, as illustrated in Fig. 8(a) from approximately 90% of all pruning operations onwards.

Advertisement Class 2. There exists a cut-off point at approximately 73% of all pruning operations (abscissa) for Advertisement Class 2 (Fig. 8(b)). At this point, the pruning algorithm has decreased the memory requirements for advertisements by 48%. The time efficiency has increased by 5% to calculate all overlappings (Scenario All) and by 8% to determine whether any overlapping exists (Scenario One). The total amount of overlappings has grown from 51% to 54% of all registered subscriptions. When performing more pruning operations than stated by the cut-off point, the amount of overlappings increases up to 100%, i.e., all subscriptions overlap all advertisements.

We can find the reason for this behavior in the advertisement and subscription structures: Up to the cut-off point, the algorithm only prunes predicates on the attributes *Buy It Now* (node n_6, cf. Fig. 2(b)) and *Special Attribute* (node n_9), which do not influence the overlappings (cf. Advertisement Class 1). All of these predicates have been pruned at the cut-off point. Later on, the algorithm prunes the disjunctive nodes (node n_2). Hence, the only remaining predicate in advertisements is on the attribute *Author* (node n_3). All registered subscriptions are then overlapping these advertisements due to the subscription structure.

Advertisement Class 3. We can decrease the memory requirements for advertisements by 80% when performing all possible advertisement pruning operations for Advertisement Class 3 (Fig. 9(a)). Additionally, the efficiency properties improve strongly: The algorithm works 55% faster for Scenario All; the computations for Scenario One are performed 24% more efficiently. Also the amount of overlappings develops promising and increases by only 1%.

The reason for the stronger improvement in Scenario One compared to Scenario All is the development of candidate subscriptions: The number of candidates increases when pruning. All of these candidates need to be evaluated to determine all overlappings. Hence, the reduction in advertisement complexity (due to prunings) does not have the same strong effect as in case of merely determining the existence of overlappings. After removing the disjunctive node of advertisements (node n_2, cf. Fig. 2(c)),

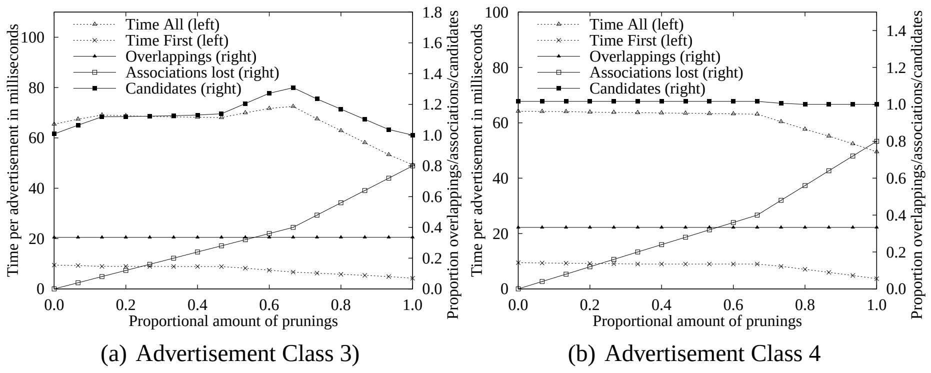


Fig. 9. Influence of advertisement pruning on Advertisement Classes 3 and 4

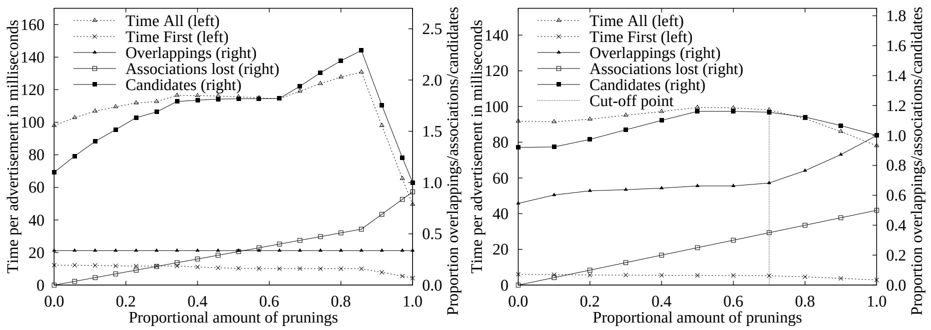
the amount of candidates decreases again. This is due the the definition of violating predicates (cf. Sect. 4.2) for disjunctive nodes, which determine the candidate subscriptions. Also the memory requirements decrease more strongly from this point onwards because the algorithm removes two associations per pruning operation.

The reason for the advantageous slight increase in the overlappings is again found in the structure of advertisements and subscriptions: Subscription Class 3 is not influenced by any pruning operations because it specifies predicates on other attributes only. Subscription Classes 1 and 2 use the same attributes as Advertisement Class 3. Hence, pruning operations should affect the overlappings but only do so insignificantly: The most selective indicator for overlappings is the *Title* attribute (node n_3). Subscriptions and advertisements have to specify the same title in order to lead to a potential overlapping. The overlapping is further depending on the prices (Nodes n_7 and n_8) according to the condition (Nodes n_6 and n_9). For most subscriptions either the prices for used or for new book copies lead to an overlapping. Thus, pruning any of these predicates in advertisements does not change the overlapping relationship in most cases.

Advertisement Class 4. Advertisement Class 4 (Fig. 9(b)) has a similar structure as Advertisement Class 3. Thus, we experience a similar behavior when performing pruning operations: The memory requirements for advertisements decrease by 80% when performing all possible prunings. This strong improvement creates an additional amount of overlappings of less than 1% of all registered subscriptions (always around 33%). The efficiency to calculate all overlapping subscriptions is increased by 23% (Scenario All); the determination whether any overlapping exists works 61% faster (Scenario First).

The reason for this beneficial behavior when pruning is again found in the structure of subscriptions and advertisements: Pruning the predicates on *Format* (Nodes n_6 and n_9, cf. Fig. 2(d)) does not affect the amount of overlappings to a large extend. This is due to the *Title* attribute (node n_3) mandatorily requiring to specify the same value in an advertisement and a subscription to lead to an overlapping. If this is given, in most cases either the respective prices for hardcover or softcover items overlap (Subscription

Class 2), or one of these prices overlaps any of the specifications in subscriptions (Subscription Class 1). When pruning the disjunctive node (node n_2) in advertisements, we again experience a stronger decrease in memory usage than when pruning leaf nodes (bend in curve). The efficiency also increases more strongly from this point onwards.



(a) Advertisement Class 5 (b) Advertisement Class 6

Fig. 10. Influence of advertisement pruning on Advertisement Classes 5 and 6

Advertisement Class 5. Advertisement Class 5 (Fig. 10(a)) leads to the best results when applying advertisement pruning: The memory requirements have been decreased by 91% after performing all possible pruning operations. At this point, the computations in Scenario All can be performed 49% faster than without applying advertisement pruning. For Scenario First, the improvement is even 65%. The pruning algorithm achieves these advances with increasing the amount of overlappings by only 1%.

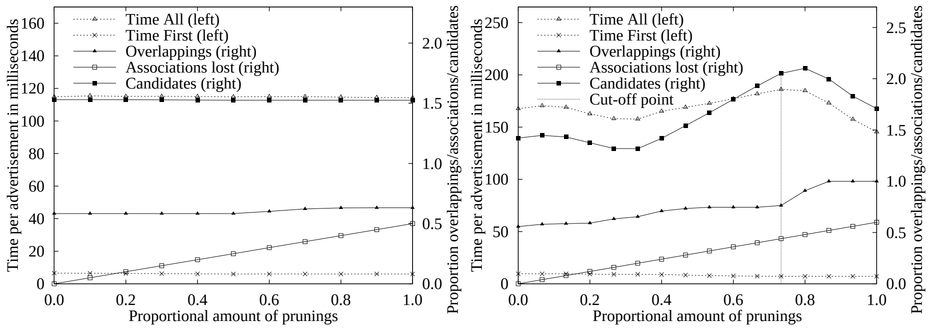
However, the application of pruning firstly degrades the efficiency to determine all overlappings. The reason is again found in the number of candidates, which show a similar increase as the overall time to determine the overlappings in Scenario All. This increase in candidates is due to the performed pruning operations: In the beginning, the pruning algorithm chooses to prune predicates on the attribute *Format* (Nodes n_7 and n_9, cf. Fig. 2(e)). This results in more subscriptions of Class 2 being considered as candidates. Then, mainly predicates specifying low prices are pruned (also some predicates stating new books). This results in an increase in candidates of Subscription Classes 1 and 2. However, at some point the amount of candidates only increases very slowly anymore because mainly predicates specifying new books (Nodes n_{14} and n_{18}) are pruned (which decreases the number of violating predicates much less than pruning predicates on *Price*). If most of these predicates on *Condition* have been pruned, the algorithm again decides to prune on *Price*. This leads to the newly developing larger gradient of the curve in Fig. 10(a). Finally, the disjunctive nodes (node n_3) of advertisements are pruned, resulting in a dropping amount of candidates (only one predicate remains per advertisement (node n_2)).

Advertisement Class 6. For Advertisement Class 6 (Fig. 10(b)), we again experience a cut-off point. This time, it occurs after having performed approximately 70% of all possible pruning operations. At this point, the memory requirements could be reduced by 35%; the overlappings are increased from 55% to 68% of all registered subscriptions. The algorithm can determine whether any overlappings exist (Scenario First) 13% more efficiently at the cut-off point. However, all overlapping subscriptions (Scenario All) require a 6% higher computation time compared to the situation without pruning.

When performing all prunings, each subscription overlaps every advertisement, but the computation time is faster than without pruning. The reason for this behavior is that after more than approximately 70% of all prunings, mostly one predicate on *Condition* and one predicate on *Price* remain in advertisements. Because subscriptions of Classes 1 and 2 always specify the interest in both possible conditions, all subscriptions finally overlap. The overlappings with Subscription Class 3 are not influenced by any prunings.

The first increase in the computation time in Scenario All is again due to the increase in candidates and the required evaluation of these candidates. After the cut-off point, the amount of candidates decreases and thus also the computation time.

Altogether, more overlappings exist for Advertisement Class 6 than for Class 3 because Class 6 does not additionally restrict the titles of books, i.e., Advertisement Class 6 is more general than Advertisement Class 3.



(a) Advertisement Class 7 (b) Advertisement Class 8

Fig. 11. Influence of advertisement pruning on Advertisement Classes 7 and 8

Advertisement Class 7. There are again strong benefits of advertisement pruning when using Advertisement Class 7 (Fig. 11(a)): After performing all pruning operations, the memory usage is half of the original one, and the overall increase in overlapping subscriptions is only 5%. Additionally, the efficiency in Scenario First is increased by 8%; the time to compute all overlappings is improved by 1%.

The amount of overlappings is nearly unaffected by pruning operations for Advertisement Class 8 because only predicates on the attribute *Format* (Nodes n_4 and n_7, cf. Fig. 3(b)) are considered to be pruned (similarly to Advertisement Class 4). Firstly, the algorithm prunes hardcover specifications (node n_4), which does not affect the overlappings at all (hardcover books are generally more expensive than softcover books). When pruning softcover predicates (node n_7), the amount of overlappings increases slightly. In fact, it only increases if the hardcover part of an advertisement does not overlap the subscription but the lower price from the softcover overlaps the subscription.

Advertisement Class 8. We again experience a cut-off point for advertisements of Advertisement Class 8 (Fig. 11(b)). This point occurs at approximately 73% of all pruning operations. The pruning algorithm can decrease the memory usage at this point by 44%. The total number of overlappings is increased by 21%, and the computations in Scenario First work 25% faster. However, the efficiency to compute all overlappings (Scenario All) degrades by 11%.

This efficiency degradation in Scenario All is again caused by the increase in candidate subscriptions. Similar to Advertisement Class 5, the pruning algorithm firstly removes predicates on *Format* (Nodes n_5 and n_6, cf. Fig. 3(c)), increasing the candidates of Subscription Class 2. However, in contrast to Advertisement Class 5 there is no predicate on *Title* in Advertisement Class 8. This property also increases the amount of overlappings and finally leads to the situation of all advertisements overlapping all subscriptions (when pruning more than specified by the cut-off point).

Advertisement Classes 1–8. The influence of advertisement pruning in the setting including a combination of all eight advertisement classes is not merely the mean of the results of the individual classes. It is also influenced by the quality of our heuristic when applied to different advertisement structures simultaneously.

In Fig. 12, we have illustrated the cut-off point that is occurring after performing approximately 77% of all pruning operations. At this point, the pruning algorithm has decreased the memory usage by 49%; the overall amount of overlappings has increased by only 5% at this cut-off point. The efficiency properties in Scenario First are improved by 24%; Scenario All, however, leads to a decrease by 3%. When performing slightly more prunings than stated by the cut-off point, also the efficiency in Scenario All does improve compared to the unpruned situation (3% efficiency increase with a further 5% increase in overlappings). Although, we have plotted the cut-off point at 77% because of the bend that is existing in the overlapping curve in Fig. 12.

Our combined setting shows that our pruning heuristic does also work if registering advertisements involving different structures. The promising results at the cut-off point—and in particular its position—show that our heuristic chooses pruning operations for advertisements of different classes and correctly judges their influence on the overlappings. In fact, the pruning algorithm firstly performs those prunings that do not strongly increase the overlappings (pruning operations before the cut-off points of the individual classes). The pruning operations increasing the overlappings (after the cut-off points of the individual classes) are only performed if no other suitable pruning opportunities exist anymore, which is leading to the cut-off point in the combined setting.

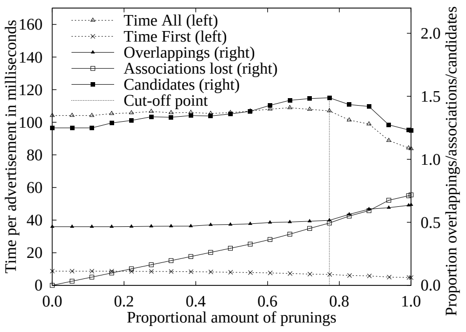


Fig. 12. Influence of advertisement pruning on Advertisement Classes 1–8

Summary of the Results. We have illustrated an overview of the previously described behaviors in Table 4. In the columns, we show our different settings. Rows contain the measured parameters: Row 1 states the amount of prunings at the cut-off point proportional to the overall number of possible pruning operations. The decrease in memory at the cut-off point is presented in Row 2 (proportional to the overall memory usage for advertisements). Row 3 indicates the proportional change in the average computation time per advertisement for Scenario All (an advantageous decrease is indicated by "↓" and an increase by "↑"). The improvement in Scenario First is shown in Row 4. Row 5 contains the proportional increase in overlappings at the cut-off point. The total increase in overlappings is finally given in Row 6.

Our results show that advertisement pruning is a valuable optimization: At the cut-off point, the memory usage has been decreased between 44% and 91%. The efficiency of the practically required operation to determine whether an overlapping exists is, at the same time, improved by 8% to 65%. The total increase in overlappings at the cut-off point is between 1% and 24%. For the general setting, involving all advertisement classes, the decrease in memory usage is 49%, the efficiency improvement 24%, and the increase in overlappings only 5%.

6.5 Combining Advertisement and Subscription Pruning

Our evaluation in the previous section has only considered the influence of advertisement pruning so far. Subscription pruning, which advertisement pruning is derived from (cf. Sect. 5), has been analyzed in [7]. These independent evaluations have shown that the two approaches are valuable optimizations if applied individually. What is still missing is to analyze the effects of utilizing both subscription and advertisement pruning (i.e., subscription-based and advertisement-based optimizations) at the same time. We will do so in this section. We have again registered 100,000 subscriptions and 50,000 advertisements in this series of experiments.

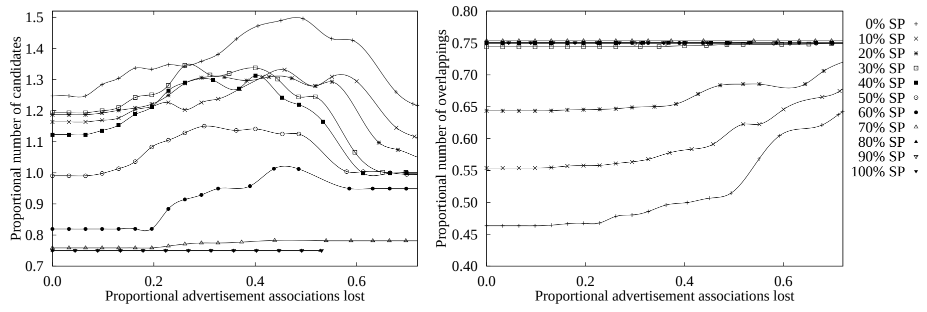
For our evaluation, we consider the general setting containing subscriptions of all three classes and advertisements of all eight classes. We analyze the influence of advertisement pruning after having performed different amounts of subscription pruning

Table 4. Overview of the influence of advertisement pruning at the cut-off point (the eight advertisement classes are abbreviated by A1 to A8)

Advertisement class	A1	A2	A3	A4	A5	A6	A7	A8	A1–8
Prop. cut-off point	1.0	0.73	1.0	1.0	1.0	0.7	1.0	0.73	0.77
Prop. decrease in memory	0.65	0.48	0.8	0.8	0.91	0.35	0.5	0.44	0.49
Prop. change in time (All)	↓0.07	↓0.05	↓0.24	↓0.23	↓0.49	↑0.06	↓0.01	↑0.11	↑0.03
Prop. decrease in time (First)	0.18	0.08	0.55	0.61	0.65	0.13	0.08	0.25	0.24
Prop. increase in overlappings	0.09	0.06	0.01	0.01	0.01	0.25	0.08	0.37	0.11
Total increase in overlappings	0.05	0.03	0.01	0.01	0.01	0.24	0.05	0.21	0.05

operations. These amounts vary from the situation of no subscription pruning (0% SP) to the point where all possible subscription prunings have been performed (100% SP). We vary the proportion of subscription pruning in steps of 10%, each displayed in a separate curve.

In contrast to our figures in the previous subsection, the abscissae of our graphs now state the proportion of reduced memory requirements, i.e., the amount of removed predicate advertisement associations (previously also shown as a curve). The maximal possible reduction varies according to the amount of subscription prunings, e.g., 72% for no subscription pruning (0% SP) and 53% for all possible subscription prunings (100% SP).



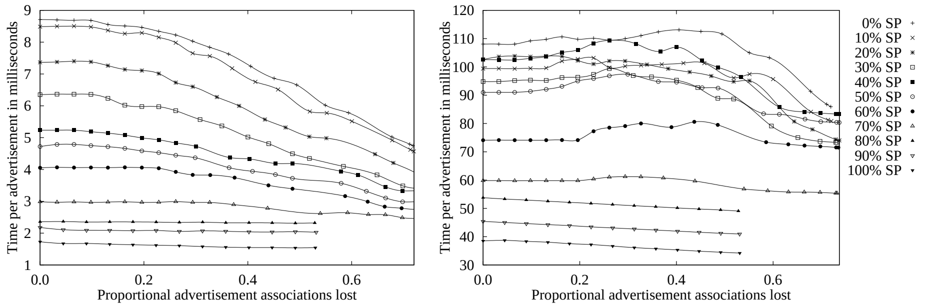
(a) Number of Candidates (b) Number of Overlappings

Fig. 13. Amount of candidates and amount of overlappings when applying both subscription and advertisement pruning

In Fig. 13(a), we have illustrated the proportional number of candidate subscriptions, strongly influencing the time efficiency in Scenario All (determination of all overlappings). Generally, the more subscription pruning operations one performs (represented by the different curves), the less candidates exist. When only some subscription prunings have been executed, this behavior is not that stable yet and does still depend on the advertisement prunings as well. But after approximately 50% of all subscription prunings, the amount of candidates clearly decreases more and more. From 80% of all subscription prunings onwards, the number of candidates remains stable at 75% of all registered subscriptions.

The reason for this decrease in candidates is the removal of predicates in subscriptions, leading to more general subscriptions. Obviously, more general subscriptions do overlap more advertisements than more restrictive subscriptions. Hence, the amount of candidates decreases because subscriptions are mostly evaluated only once.

The just pointed out increase in overlappings when performing subscription pruning is illustrated in Fig. 13(b) (proportional amount of overlappings). Without subscription pruning, the proportional number of overlappings is approximately 46%. Having performed all subscription prunings, approximately 75% of all subscriptions overlap an advertisement on average (which is the same as the number of candidates from 80% of all subscription prunings onwards, i.e., at this point all candidates are, in fact, an overlap). This is a rather strong increase, showing that the subscription pruning heuristic does not focus on the influence of prunings on overlappings. Advertisement pruning, however, only slightly increases the overlappings: When reducing the amount of predicate advertisement associations by 72%, the amount of overlappings increases by 18% (0% subscription pruning). But reducing the amount of predicate subscription associations by only 30% results in an increase in overlappings of even 29% (0% advertisement pruning). A sound property of subscription pruning in combination with advertisement pruning is, however, the stabilization of the number of overlappings at 30% of all subscription prunings. Having performed these prunings, neither advertisement nor subscription pruning influences the overlappings anymore. Thus, one could perform all possible advertisement prunings and all possible subscription prunings without further increasing the overlappings.



(a) Efficiency in Scenario First (b) Efficiency in Scenario All

Fig. 14. Average time per advertisement to determine whether any overlappings exist (Scenario First) and to determine all overlappings (Scenario All)

The influence of subscription pruning on the time efficiency in Scenario First (determination whether an overlapping exists) is shown in Fig. 14(a). Here, we can clearly observe the improving influence of subscription pruning on the efficiency properties: The more subscription prunings one performs, the more efficient the determination whether overlappings exist. Additionally, the more subscription prunings have been executed, the less influence does the application of advertisement pruning have: Without any subscription pruning, advertisement pruning improves the efficiency in Scenario First by 46%. Having performed all possible subscription prunings, the improvement is 11%. Altogether, this set of results shows that both subscription and advertisement pruning improve the efficiency to determine the existence of overlappings.

The time efficiency to calculate all overlappings is finally shown in Fig. 14(b). Similar to the development of the candidates (Fig. 13(a)), subscription pruning generally improves the efficiency properties of calculating the overlappings after a stabilization phase: Having performed more than 50% of all subscription prunings, Fig. 14(b) clearly shows a steadily improving time efficiency. Without any advertisement pruning, the time to calculate all overlapping subscriptions has decreased from approximately 108 to 38 msec per advertisement. Generally, the time efficiency to calculate overlappings develops similarly as the candidate subscriptions due the interdependence of these two parameters. So, up to 70% of all subscription prunings, advertisement pruning shows an uneven development of the calculation times (similar to the development of the candidates). When performing more than 70% of all subscription prunings, the time efficiency when additionally applying advertisement pruning improves linearly. This is due to the stable amount of candidates (cf. Fig. 13(a)) in combination with the decreasing complexity of advertisements.

Summarizing these results, we can state that subscription pruning leads to an optimization when applied in combination with advertisement pruning. However, subscription pruning does not focus on the influence of prunings on the overlappings (it focusses on minimally increasing the network load for event routing). Thus, it increases the amount of overlappings stronger than advertisement pruning when reducing the memory usage to the same amount. The increase in overlappings stops after having performed a certain number of subscription pruning operations (approximately 30% of all possible prunings). After this point, subscription pruning still increases the efficiency properties very significantly and, simultaneously, reduces the memory requirements to index and store subscription.

7 Conclusions and Future Work

Current pub/sub systems define subscriptions and advertisements as conjunctive filter expressions. An advantage of such conjunctive solutions is the opportunity to ignore the internal structure of subscriptions while filtering. However, there also exist advantages when directly supporting arbitrary Boolean subscriptions, as it has been shown recently.

In this paper, we have presented the first advertisement-based pub/sub system that is internally supporting both arbitrary Boolean subscriptions and advertisements. In Sect. 3.1 to 3.3, we have firstly defined the exact semantics of event messages, arbitrary Boolean subscriptions, and arbitrary Boolean advertisements. This step has become necessary because the introduction of arbitrary Boolean filter expressions imposes challenges that do not occur for restricted conjunctive forms. These challenges range from the problems occurring due to the usage of the same attribute in several or no predicates, to the definition of an overlapping relationship between advertisements and subscriptions. In Sect. 3.4, we could then show that arbitrary Boolean advertisements require less memory for storage and indexation than conjunctive ones.

The greatest challenge when supporting arbitrary Boolean advertisements is to develop an algorithm to determine the overlapping relationships. We have proposed such an algorithm in Sect. 4. We have started by showing that current calculation approaches do not work for the Boolean case followed by proposing the general computation idea of our algorithm in Sect. 4.1. This outline has led to the concept of violating predicates, described in Sect. 4.2. We have then elaborated on how to base the calculation of overlappings on the violating predicates in Sect. 4.3; Section 4.4 has finally described a practical implementation of our algorithm.

In this paper, we have also presented a novel optimization approach, advertisement pruning, that is tailored to advertisements. Advertisement pruning is based on the subscription pruning routing optimization (reviewed in Sect. 2.3) but directly bases its pruning decisions on the influence of these prunings on the overlapping properties among subscriptions and advertisements. This makes advertisement pruning to the first advertisement-tailored optimization for pub/sub systems

We have successively developed this optimization approach in Sect. 5: After having identified the effects of prunings on the overlappings in Sect. 5.2, we have incorporated these influences into the measure of an overlapping rank in Sect. 5.3. This rank then allows us to estimate the influence of pruning operations, as described in Sect. 5.4. In Sect. 5.5, we have shown how this, finally, gives us the means to determine the best advertisement pruning to perform, i.e., the pruning operation that increases the overlappings the least.

The final part of this paper, Sect. 6, has focused on the extensive experimental evaluation of our approaches. In Sect. 6.1, we have described our experimental setup, and have identified eight advertisement classes and three subscription classes, characteristic for an online auction application scenario. These definitions allow for the classification of our results and the repeatability of our experiments.

We have investigated the general properties of our approach for calculating the overlappings in Sect. 6.2 and have found a linear increase of the calculation times with an increasing number of advertisements. In Sect. 6.3, we have additionally compared our approach to conjunctive solutions. We could show that our algorithm strongly outperforms the conjunctive approach in determining whether any overlappings exist. This behavior is mainly due to the exponential explosion of the problem size when converting arbitrary Boolean to conjunctive expressions. If it is required to calculate all overlappings, the more efficient computation solution (Boolean or conjunctive) depends on the structure of advertisements and subscriptions.

In Sect. 6.4, we have then evaluated the advertisement pruning optimization for the different advertisement classes (identified in Sect. 6.1) individually as well as for a combined setting. We could show that advertisement pruning is a valuable optimization that, at the same time, is (i) strongly reducing the memory requirements to store advertisements (49% less memory in the combined setting), (ii) moderately increasing the efficiency to determine whether overlappings exist (24% more efficient in the combined setting), and (iii) only slightly increasing the amount of overlappings (5% in the combined setting).

The last part of our evaluation (Sect. 6.5) has focused on the effects of simultaneously applying both subscription and advertisement pruning. We could show that subscription pruning also leads to an optimization if it is applied in addition to advertisement pruning. However, subscription pruning increases the amount of overlappings more strongly than advertisement pruning. This behavior is clearly due to the different focus of subscription pruning and shows that our estimation of the influences of advertisement prunings has been effectively encapsulated in the introduced, advertisement-tailored optimization.

Our results show that the support of arbitrary Boolean subscriptions and advertisements does lead to improvements of pub/sub systems in respect to both time efficiency and space efficiency. Additionally, our results prove that optimizations for advertisements can significantly reduce the memory usage of pub/sub systems while increasing their efficiency. In the future, we plan to analyze the algorithm to calculate the overlappings and the advertisement pruning optimization in combination with other application scenarios to show their general applicability. Furthermore, we plan to fully support arbitrary Boolean advertisements and their optimization in our prototype of a distributed pub/sub system. This finally allows for the analysis of their effects in the distributed system.

References

1. M. Antollini, M. Cilia, and A. Buchmann. Implementing a High Level Pub/Sub Layer for Enterprise Information Systems. In *Proceedings of the 8th International Conference on Enterprise Information Systems*, Paphos, Cyprus, May 23–27 2006. Springer-Verlag.
2. S. Bittner and A. Hinze. A Detailed Investigation of Memory Requirements for Publish/Subscribe Filtering Algorithms. In *Proceedings of the 13th International Conference on Cooperative Information Systems (CoopIS 2005)*, pages 148–165, Agia Napa, Cyprus, October 31–November 4 2005. Springer-Verlag.
3. S. Bittner and A. Hinze. On the Benefits of Non-Canonical Filtering in Publish/Subscribe Systems. In *Proceedings of the 25th IEEE International Conference on Distributed Computing Systems Workshops (ICDCSW ’05)*, pages 451–457, Columbus, USA, June 6–10 2005. IEEE Computer Society.
4. S. Bittner and A. Hinze. Dimension-Based Subscription Pruning for Publish/Subscribe Systems. In *Proceedings of the 26th IEEE International Conference on Distributed Computing Systems Workshops (ICDCSW ’06)*, Lisbon, Portugal, July 4–7 2006. IEEE Computer Society.
5. S. Bittner and A. Hinze. Event Distributions in Online Book Auctions. Technical Report 03/2006, Computer Science Department, The University of Waikato, February 2006.
6. S. Bittner and A. Hinze. Pruning Subscriptions in Distributed Publish/Subscribe Systems. In *Proceedings of the Twenty-Ninth Australasian Computer Science Conference (ACSC 2006)*, Hobart, Australia, January 16–19 2006. ACS.
7. S. Bittner and A. Hinze. Subscription Tree Pruning: A Structure-Independent Routing Optimization for General-Purpose Publish/Subscribe Systems. Technical Report 01/2006, Computer Science Department, The University of Waikato, January 2006.

8. A. Carzaniga, D. S. Rosenblum, and A. L. Wolf. Interfaces and Algorithms for a Wide-Area Event Notification Service. Technical Report CU-CS-888-99, Department of Computer Science, University of Colorado, October 1999. revised May 2000.
9. A. Carzaniga, D. S. Rosenblum, and A. L. Wolf. Design and Evaluation of a Wide-Area Event Notification Service. *ACM Transactions on Computer Systems (TOCS)*, 19(3):332–383, 2001.
10. R. Chand and P. A. Felber. A Scalable Protocol for Content-Based Routing in Overlay Networks. In *Proceedings of the Second IEEE International Symposium on Network Computing and Applications (NCA 2003)*, pages 123–130, Cambridge, USA, April 16–18 2003. IEEE Computer Society.
11. P.-A. Chirita, S. Idreos, M. Koubarakis, and W. Nejdl. Designing Semantic Publish/Subscribe Networks Using Super-Peers. In *Semantic Web and Peer-to-Peer*, pages 159–181. Springer-Verlag, 2006.
12. P. T. Eugster, R. Guerraoui, and J. Sventek. Distributed Asynchronous Collections: Abstractions for Publish/Subscribe Interaction. In *Proceedings of the 14th European Conference on Object-Oriented Programming (ECOOP '2000)*, pages 252–276, Cannes, France, June 12–16 2000. Springer-Verlag.
13. J. Gough and G. Smith. Efficient Recognition of Events in a Distributed System. In *Proceedings of the 18th Australasian Computer Science Conference (ACSC-18)*, Adelaide, Australia, February 1–3 1995. ACS.
14. M. Guimarães and L. Rodrigues. A Genetic Algorithm for Multicast Mapping in Publish-Subscribe Systems. In *Proceedings of the Second IEEE International Symposium on Network Computing and Applications (NCA 2003)*, pages 67–74, Cambridge, USA, April 16–18 2003. IEEE Computer Society.
15. D. Heimbigner. Expressive and Efficient Peer-to-Peer Queries. In *Proceedings of the 38th Hawaii International Conference on System Sciences (HICSS-38)*, Big Island, USA, January 3–6 2005. IEEE Computer Society.
16. A. Hinze. *A-MEDIAS: Concept and Design of an Adaptive Integrating Event Notification Service*. PhD thesis, Freie Universität Berlin, Institute of Computer Science, July 2003.
17. G. Li, S. Hou, and H.-A. Jacobsen. A Unified Approach to Routing, Covering and Merging in Publish/Subscribe Systems based on Modified Binary Decision Diagrams. In *Proceedings of the 25th IEEE International Conference on Distributed Computing Systems (ICDCS '05)*, pages 447–457, Columbus, USA, June 6–10 2005. IEEE Computer Society.
18. G. Mühl. *Large-Scale Content-Based Publish/Subscribe Systems*. PhD thesis, Technische Universität Darmstadt, September 2002.
19. G. Mühl, L. Fiege, and A. Buchmann. Filter Similarities in Content-Based Publish/Subscribe Systems. In *Proceedings of the International Conference on Architecture of Computing Systems (ARCS '02)*, pages 224–238, Karlsruhe, Germany, April 8–12 2002. Springer-Verlag.
20. F. Peng and S. S. Chawathe. XPath Queries on Streaming Data. In *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data (SIGMOD 2003)*, pages 431–442, San Diego, USA, June 9–12 2003. ACM Press.
21. M. Petrovic, H. Liu, and H.-A. Jacobsen. G-ToPSS: Fast Filtering of Graph-based Metadata. In *Proceedings of the 14th International Conference on World Wide Web (WWW 2005)*, pages 539–547, Chiba, Japan, May 10–14 2005. ACM Press.
22. G. P. Picco, G. Cugola, and A. L. Murphy. Efficient Content-Based Event Dispatching in the Presence of Topological Reconfiguration. In *Proceedings of the 23rd IEEE International Conference on Distributed Computing Systems (ICDCS '03)*, pages 234–243, Rhode Island, USA, May 19–22 2003. IEEE Computer Society.
23. P. R. Pietzuch. *Hermes: A Scalable Event-Based Middleware*. PhD thesis, University of Cambrigde, Queens' College, February 2004.

24. T. Sivaharan, G. Blair, and G. Coulson. GREEN: A Configurable and Re-configurable Publish-Subscribe Middleware for Pervasive Computing. In *Proceedings of the 7th International Symposium on Distributed Objects and Application (DOA 2005)*, pages 732–749, Agia Napa, Cyprus, October 31–November 4 2005. Springer-Verlag.
25. Y.-M. Wang, L. Qiu, C. Verbowski, D. Achlioptas, G. Das, and P. Larson. Summary-based Routing for Content-based Event Distribution Networks. *ACM SIGCOMM Computer Communication Review*, 34(5):59–74, 2004.
26. T. W. Yan and H. García-Molina. Index Structures for Selective Dissemination of Information Under the Boolean Model. *ACM Transactions on Database Systems (TODS)*, 19(2):332–364, 1994.