Mining Optimized Gain Rules for Numeric Attributes

Sergey Brin
Stanford University
Palo Alto, CA
sergey@cs.stanford.edu

Rajeev Rastogi
Bell Laboratories
Murray Hill, NJ
rastogi@lucent.com

Kyuseok Shim
KAIST* and AITrc†
Taejon, Korea
shim@cs.kaist.ac.kr

Abstract

Association rules are useful for determining correlations between attributes of a relation and have applications in marketing, financial and retail sectors. Furthermore, *optimized association rules* are an effective way to focus on the most interesting characteristics involving certain attributes. Optimized association rules are permitted to contain uninstantiated attributes and the problem is to determine instantiations such that either the support, confidence or gain of the rule is maximized.

In this paper, we generalize the optimized gain association rule problem by permitting rules to contain disjunctions over uninstantiated numeric attributes. Our generalized association rules enable us to extract more useful information about seasonal and local patterns involving the uninstantiated attribute. For rules containing a single numeric attribute, we present an algorithm with linear complexity for computing optimized gain rules. Furthermore, we propose a bucketing technique that can result in a significant reduction in input size by coalescing contiguous values without sacrificing optimality. We also present an approximation algorithm based on *dynamic programming* for two numeric attributes. Using recent results on *binary space partitioning trees*, we show that the approximations are within a constant factor of the optimal optimized gain rules. Our experimental results with synthetic data sets for a single numeric attribute demonstrate that our algorithm scales up linearly with the attribute's domain size as well as the number of disjunctions. In addition, we show that applying our optimized rule framework to a population survey real-life data set enables us to discover interesting underlying correlations among the attributes.

Keywords: Association rules, support, confidence, gain, dynamic programming, region bucketing, binary space partitioning.

1 Introduction

Association rules, introduced in [AIS93], provide a useful mechanism for discovering correlations among the underlying data and have applications in marketing, financial and retail sectors. In its most general form, an association rule can be viewed as being defined over attributes of a relation, and has the form $C_1 \rightarrow C_2$, where C_1 and C_2 are conjunctions of conditions, and each condition is either $A_i = v_i$ or $A_i \in [l_i, u_i]$ (v_i, l_i and u_i are values from the domain of the attribute A_i). Each rule has an associated *support* and *confidence*. Let the *support* of a condition C_i be the ratio of the number of tuples satisfying C_i and the number of tuples in the relation. The support of a rule of the form $C_1 \rightarrow C_2$ is then the same as the support of $C_1 \wedge C_2$, while its confidence is the ratio of

*Korea Advanced Institute of Science and Technology
†Advanced Information Technology Research Center

the supports of conditions $C_1 \wedge C_2$ and C_1. The association rules problem is that of computing all association rules that satisfy user-specified minimum support and minimum confidence constraints, and efficient schemes for this can be found in [AS94, MTV94, PCY95, SON95, HF95, SA95, SA96].

For example, consider a relation in a telecom service provider database that contains call detail information. The attributes of the relation are date, time, src_city, src_country, dst_city, dst_country and duration. A single tuple in the relation thus captures information about the two endpoints of each call, as well as the temporal elements of the call. The association rule (src_city = NY) $\rightarrow$ (dst_country = France) would satisfy the user-specified minimum support and minimum confidence of 0.05 and 0.3, respectively, if at least 5% of total calls are from NY to France, and at least 30% of the calls that originated from NY are to France.

1.1 Optimized Association Rules

The *optimized association rules* problem, motivated by applications in marketing and advertising, was introduced in [FMMT96a]. An association rule R has the form $(A_1 \in [l_1, u_1]) \wedge C_1 \rightarrow C_2$, where A_1 is a numeric attribute, l_1 and u_1 are uninstantiated variables, and C_1 and C_2 contain only instantiated conditions (that is, the conditions do not contain uninstantiated variables). Also, the authors define the *gain* of rule R, denoted by $gain(R)$, to be the difference between the support of $(A_1 \in [l_1, u_1]) \wedge C_1 \wedge C_2$ and the support of $(A_1 \in [l_1, u_1]) \wedge C_1$ times the user-specified minimum confidence. The authors then propose algorithms for determining values for the uninstantiated variables l_1 and u_1 for each of the following cases:

- Confidence of R is maximized and the support of the condition $(A_1 \in [l_1, u_1]) \wedge C_1$ is at least the user-specified minimum support (referred to as the *optimized confidence* rule).
- Support of the condition $(A_1 \in [l_1, u_1]) \wedge C_1$ is maximized and confidence of R is at least the user-specified minimum confidence (referred to as the *optimized support* rule).
- Gain of R is maximized and confidence of R is at least the user-specified minimum confidence (referred to as the *optimized gain* rule).

Optimized association rules are useful for unraveling ranges for numeric attributes where certain trends or correlations are strong (that is, have high support, confidence or gain). For example, suppose the telecom service provider mentioned earlier was interested in offering a promotion to NY customers who make calls to France. In this case, the timing of the promotion may be critical – for its success, it would be advantageous to offer it close to a period of consecutive days in which the percentage of calls from NY that are directed to France is maximum. The framework developed in [FMMT96a] can be used to determine such periods. Consider, for example, the association rule (date $\in [l_1, u_1]$) $\wedge$ src_city = NY $\rightarrow$ dst_country = France. With a minimum confidence of 0.5, the optimized gain rule results in the period in which the calls from NY to France exceeds 50% of the total calls from NY, and furthermore, the number of these excess calls is maximum.

A limitation of the optimized association rules dealt with in [FMMT96a] is that only a single optimal interval for a single numeric attribute can be determined. However, in a number of applications, a single interval may be an inadequate description of local trends in the underlying data. For example, suppose the telecom service provider is interested in doing upto k promotions for customers in NY calling France. For this purpose, we need a mechanism to identify upto k periods during which a sizeable fraction of calls are from NY to France. If association rules were permitted to contain disjunctions of uninstantiated conditions, then we could determine the optimal k (or fewer) periods by finding optimal instantiations for the rule: (date $\in [l_1, u_1] \vee \cdots \vee$ date $\in [l_k, u_k]$) $\wedge$ src_city = NY $\rightarrow$ dst_country = France. This information can be used by the telecom service provider to determine the most suitable periods for offering discounts on international long distance calls to France. The

above framework can be further strengthened by enriching association rules to contain more than one uninstantiated attribute as is done in [FMMT96b]. Thus, optimal instantiations for the rule $(\mathsf{date} \in [l_1, u_1] \wedge \mathsf{duration} \in [\hat{l_1}, \hat{u_1}]) \vee \cdots \vee (\mathsf{date} \in [l_k, u_k] \wedge \mathsf{duration} \in [\hat{l_k}, \hat{u_k}]) \rightarrow \mathsf{dst\_country} = \mathsf{France}$ would yield valuable information about types of calls (in terms of their duration) and periods in which a substantial portion of the call volume is directed to France.

1.2 Our Contributions

In this paper, we consider the generalized optimized gain problem. Unlike [FMMT96a] and [FMMT96b], we permit rules to contain upto k disjunctions over one or two uninstantiated numeric attributes. Thus, unlike [FMMT96a] and [FMMT96b], that only compute a single optimal region, our generalized rules enable upto k optimal regions to be computed.

Furthermore, unlike [RS99], in which we only addressed the optimized support problem, in this paper, we focus on the optimized gain problem and consider both the one and two attribute cases. In addition, for rules containing a single numeric attribute, we develop an algorithm for computing the optimized gain rule whose complexity is $O(nk)$ where n is the number of values in the domain of the uninstantiated attribute (the dynamic programming algorithm for optimized support that we presented in [RS99] had complexity $O(n^2k)$). We also propose a bucketing optimization that can result in significant reductions in input size by coalescing contiguous values. For two numeric attributes, we present a dynamic programming algorithm that computes approximate association rules. Using recent results on binary space partitioning trees, we show that for the optimized gain case, the approximations are within a constant factor (of $\frac{1}{4}$) of the optimal solution. Our experimental results with synthetic datasets for a single numeric attribute demonstrate that our algorithms scale up linearly with the attribute's domain size as well as the number of disjunctions. In addition, we show that applying our optimized rule framework to a population survey real-life data set enables us to discover interesting underlying correlations among the attributes.

The remainder of the paper is organized as follows. In Section 2, we discuss related work and in Section 3, we introduce the necessary definitions and problem formulation for the optimized gain problem. We present our linear time complexity algorithm for computing the optimized gain rule for a single numeric attribute in Section 4. In Section 5, we develop a dynamic programming algorithm for two numeric attributes and show that the computed gain is within a constant factor of the optimal. We present the results of our experiments with synthetic and real-life data sets in Section 6. Finally, we offer concluding remarks in Section 7.

2 Related Work

In [RS99], we generalized the optimized association rules problem for support, described in [FMMT96a]. We allowed association rules to contain upto k disjunctions over one uninstantiated numeric attribute. For one attribute, we presented a *dynamic programming* algorithm for computing the optimized support rule and whose complexity is $O(n^2k)$, where n is the number of values in the domain of the uninstantiated attribute. In [RS98], we considered a different formulation of the optimized support problem which we showed to be NP-hard even for the case of one uninstantiated attribute. The optimized support problem described in [RS98] required the confidence over all the optimal regions, considered together, to be greater than a certain minimum threshold. Thus, the confidence of an optimal region could fall below the threshold and this was the reason for its intractability. In [RS99], we redefined the optimized support problem such that each optimal region is required to have the minimum confidence. This made the problem tractable for the one attribute case.

Schemes for clustering quantitative association rules with two uninstantiated numeric attributes in the left hand side are presented in [LSW97]. For a given support and confidence, the authors present a clustering algorithm to generate a set of non-overlapping rectangles, such that every point in each rectangle has the required confidence and support. Our schemes, on the other hand, compute an optimal set of non-overlapping rectangles with the maximum gain. Further, in our approach, we only require that each rectangle in the optimal set have minimum confidence; however, individual points in a rectangle may not have the required confidence.

Recent work on histogram construction, presented in [JKM+98], is somewhat related to our optimized rule computation problem. In [JKM+98], the authors propose a dynamic programming algorithm to compute V-optimal histograms for a single numeric attribute. The problem is to split the attribute domain into k buckets such that the *sum squared error* over the k buckets is minimum. Our algorithm for computing the optimized gain rule (for one attribute) differs from the histogram construction algorithm of [JKM+98] in a number of respects. First, our algorithm attempts to maximize the gain, which is very different from minimizing the sum squared error. Second, histogram construction typically involves identifying bucket boundaries while our optimized gain problem requires us to compute optimal regions (that may not share a common boundary). Finally, our algorithm has a linear time dependency on the size of the attribute domain – in contrast, the histogram construction algorithm of [JKM+98] has time complexity that is quadratic in the number of distinct values of the attribute under consideration.

In [BA99], the authors propose a general framework for optimized rule mining, which can be used to express our optimized gain problem as a special case. However, the generality precludes the development of efficient algorithms for computing optimized rules. Specifically, the authors use a variant of Dense-Miner from [BAG97] which essentially relies on enumerating optimized rules in order to explore the search space. Since there are an exponential number of optimized rules, the authors propose pruning strategies to reduce the search space and thus improve the efficiency of the search. However, in the worst case, the time complexity of the algorithm from [BAG97] is still exponential in the size of attribute domains. In contrast, our algorithms for computing the optimized gain rule have polynomial time complexity (linear complexity for one numeric attribute) since they exploit the specific properties of gain and one and two-dimensional spaces.

3 Problem Formulation

In this section, we define the optimized association rules problem addressed in the paper. The data is assumed to be stored in a relation defined over categorical and numeric attributes. Association rules are built from *atomic* conditions each of which has the form $A_i = v_i$ (A_i could be either categorical or numeric), and $A_i \in [l_i, u_i]$ (only if A_i is numeric). For the atomic condition $A_i \in [l_i, u_i]$, if l_i and u_i are values from the domain of A_i, the condition is referred to as *instantiated*; otherwise, if they are variables, we refer to the condition as *uninstantiated*.

Atomic conditions can be combined using operators $\wedge$ or $\vee$ to yield more complex conditions. Instantiated association rules, that we study in this paper, have the form $C_1 \rightarrow C_2$, where C_1 and C_2 are arbitrary instantiated conditions. Let the support for an instantiated condition C, denoted by $sup(C)$, be the ratio of the number of tuples satisfying the condition C and the total number of tuples in the relation. Then, for the association rule R: $C_1 \rightarrow C_2$, $sup(R)$ is defined as $sup(C_1)$ and $conf(R)$ is defined as $\frac{sup(C_1 \wedge C_2)}{sup(C_1)}$. Note that our definition of $sup(R)$ is different from the definition in [AIS93] where $sup(R)$ was defined to be $sup(C_1 \wedge C_2)$. Instead, we have adopted the definition of support used in [FMMT96a, FMMT96b, RS98, RS99]. Also, let minConf denote the user-specified minimum confidence. Then, $gain(R)$ is defined to be the difference between $sup(C_1 \wedge C_2)$ and

minConf times $sup(C_1)$. In other words, $gain(R)$ is $sup(C_1 \wedge C_2) - \mathsf{minConf} * sup(C_1) = sup(R) * (conf(R) - \mathsf{minConf})$.

The optimized association rule problem requires optimal instantiations to be computed for an uninstantiated association rule that has the form: $U \wedge C_1 \rightarrow C_2$, where U is a conjunction of one or two uninstantiated atomic conditions over distinct numeric attributes, and C_1 and C_2 are arbitrary instantiated conditions. For simplicity, we assume that the domain of an uninstantiated numeric attribute is $\{1, 2, \ldots, n\}$. Depending on the number, one or two, of uninstantiated numeric attributes, consider a one or two-dimensional space with an axis for each uninstantiated attribute, and values along each axis corresponding to increasing values from the domain of the attributes. Note that if we consider a single interval in the domain of each uninstantiated attribute, then their combination results in a *region*. For the one-dimensional case, this region $[l_1, u_1]$ is simply the interval $[l_1, u_1]$ for the attribute; for the two-dimensional case, the region $[(l_1, l_2), (u_1, u_2)]$ is the rectangle bounded along each axis by the endpoints of the intervals $[l_1, u_1]$ and $[l_2, u_2]$ along the two axis.

Suppose, for a region $R = [l_1, u_1]$, we define $conf(R)$, $sup(R)$ and $gain(R)$ to be $conf$, sup and $gain$, respectively, for the rule $A_1 \in [l_1, u_1] \wedge C_1 \rightarrow C_2$ (similarly, for $R = [(l_1, l_2), (u_1, u_2)]$, $conf(R)$, $sup(R)$ and $gain(R)$ are defined to be $conf$, sup and $gain$ for $A_1 \in [l_1, u_1] \wedge A_2 \in [l_2, u_2] \wedge C_1 \rightarrow C_2$). In addition, for a set of non-overlapping regions, $S = \{R_1, R_2, \ldots, R_j\}$, $R_i = [l_{i1}, u_{i1}]$, suppose we define $conf(S)$, $sup(S)$ and $gain(S)$ to be the $conf$, sup and $gain$, respectively of the rule $\vee_{i=1}^{j} A_1 \in [l_{i1}, u_{i1}] \wedge C_1 \rightarrow C_2$. For two dimensions, in which case each $R_i = [(l_{i1}, l_{i2}), (u_{i1}, u_{i2})]$, $conf(S)$, $sup(S)$ and $gain(S)$ are defined to be the $conf$, sup and $gain$, respectively of the rule $\vee_{i=1}^{j} (A_1 \in [l_{i1}, u_{i1}] \wedge A_2 \in [l_{i2}, u_{i2}]) \wedge C_1 \rightarrow C_2$. Then, since $R_1, \ldots, R_j$ are non-overlapping regions, the following hold for set S.

$$\begin{aligned} sup(S) &= sup(R_1) + \cdots + sup(R_j) \\ conf(S) &= \frac{sup(R_1) \cdot conf(R_1) + \cdots + sup(R_j) \cdot conf(R_j)}{sup(R_1) + \cdots + sup(R_j)} \\ gain(S) &= gain(R_1) + \cdots + gain(R_j) \end{aligned}$$

Having defined the above notation, we present below, the formulation of the optimized association rule problem for gain.

Problem Definition (Optimized Gain): Given k, determine a set S containing at most k regions such that for each region $R_i \in S$, $conf(R_i) \geq \mathsf{minConf}$ and $gain(S)$ is maximized. ∎

We refer to the set S as the optimized gain set.

Example 3.1: Consider the telecom service provider database (discussed in Section 1) containing call detail data for a one week period. Figure 1 presents the summary of the relation for the seven days – the summary information includes, for each date, the total # of calls made on the date, the # of calls from NY and the # of calls from NY to France. Also included in the summary are the support, confidence and gain, for each date v, of the rule $\mathsf{date} = v \wedge \mathsf{src\_city} = \mathsf{NY} \rightarrow \mathsf{dst\_country} = \mathsf{France}$. The total number of calls made during the week is 2000.

Suppose we are interested in discovering the interesting periods with heavy call volume from NY to France (a period is a range of consecutive days). Then, the following uninstantiated association rule can be used.

$$\mathsf{date} \in [l, u] \wedge \mathsf{src\_city} = \mathsf{NY} \rightarrow \mathsf{dst\_country} = \mathsf{France}$$

Date	1	2	3	4	5	6	7
# of total calls	340	280	230	170	220	400	360
# of calls from NY	170	140	105	95	100	200	180
# of calls from NY to France	90	65	35	18	75	50	95
support	0.085	0.07	0.052	0.047	0.05	0.1	0.09
confidence	0.53	0.46	0.33	0.19	0.75	0.25	0.53
gain($\times 10^{-3}$)	2.5	-2.5	-8.75	-14.75	12.5	-25	2.5

Figure 1: Summary of Call Detail Data for a One Week Period

In the above rule, U is date $\in [l, u]$, C_1 is src_city = NY and C_2 is dst_country = France. Let us assume that we are interested in at most 2 periods (that is, $k = 2$) with minConf = 0.50. An optimized gain set is {[5,5], [7,7]} – we require upto two periods such that the percentage of calls during each of the periods from NY that are to France is at least 50%, and the gain is maximized. Of the possible periods [1,1], [5,5] and [7,7], the gain in period [5,5] is 12.5×10^{-3}, and both [1,1] and [7,7] have gains of 2.5×10^{-3}. Thus, both {[5,5], [7,7]} and {[1,1], [5,5]} are optimized gain sets. ∎

In the remainder of the paper, we shall assume that the support, confidence and gain for every point in a region are available – these can be computed by performing a single pass over the relation. The points, along with their supports, confidences and gains, thus constitute the input to our algorithms. Thus, the input size is n for the one-dimensional case, while for the two-dimensional case, it is n^2.

4 One Numeric Attribute

In this section, we tackle the problem of computing the optimized gain set when association rules contain a single uninstantiated numeric attribute. Thus, the uninstantiated rule has the form: $(A_1 \in [l_1, u_1]) \wedge C_1 \rightarrow C_2$, where A_1 is the uninstantiated numeric attribute. We propose an algorithm with linear time complexity for computing the optimized gain set (containing upto k non-overlapping intervals) in Section 4.2. But first, in Section 4.1, we present preprocessing algorithms for collapsing certain contiguous ranges of values in the domain of the attribute into a single bucket, thus reducing the size of the input n.

4.1 Bucketing

For the one-dimensional case, each region is an interval and since the domain size is n, the number of possible intervals is $O(n^2)$. Now, suppose we could split the range $1, 2, \ldots, n$ into b buckets, where $b < n$, and map every value in A_1's domain into one of the b buckets to which it belongs. Then the new domain of A_1 becomes $\{1, 2, \ldots, b\}$ and the number of intervals to be considered becomes $O(b^2)$ – which could be much smaller, thus reducing the time and space complexity of our algorithms. Note that the reduction in space complexity also results in reduced memory requirements for our algorithms.

In the following, we present a bucketing algorithm that 1) does not compromise the optimality of the optimized set (that is, the optimized set computed on the buckets is identical to the one computed using the raw domain values), and 2) has time complexity $O(n)$. The output of the

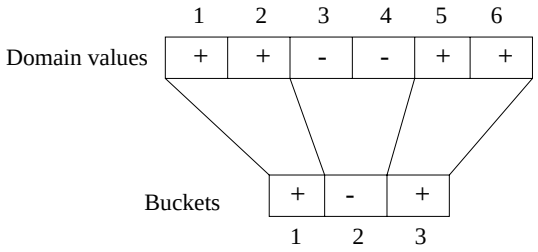


Figure 2: Example of Buckets Generated

algorithm is the b buckets with their supports, confidences and gains, and this becomes the input to the algorithm for computing the optimized gain set in Section 4.2.

For optimized gain sets, we begin by making the following simple observation – values in A_1's domain whose confidence is exactly minConf have a gain of 0 and can thus be ignored. Including these values in the optimized gain set does not affect the gain of the set and so we can assume that for every value in $\{1, 2, \ldots, n\}$, either the confidence is greater than minConf or less than minConf.

The bucketing algorithm for optimized gain collapses contiguous values whose confidence is greater than minConf into a single bucket. It also combines contiguous values each of whose confidence is less than minConf, into a single bucket. Thus, for any interval assigned to a bucket it is the case that either all values in the interval have confidence greater than minConf or all values in the interval have confidence less than minConf.

For instance, let the domain of A_1 be $\{1, 2, \ldots, 6\}$ and confidences of 1, 2, 5 and 6 be greater than minConf, while confidences of 3 and 4 be less than minConf. This is illustrated in Figure 2 with symbols + and − indicating a positive and negative gain, respectively, for domain values. Then, our bucketing scheme generates 3 buckets – the first containing values 1 and 2, the second 3 and 4, and the third containing values 5 and 6. It is straightforward to observe that assigning values to buckets can be achieved by performing a single pass over the input data and thus has linear time complexity.

In order to show that the above bucketing algorithm does not violate the optimality of the optimized set, we use the result of the following theorem.

Theorem 4.1: *Let S be an optimized gain set. Then, for any interval $[u, v]$ in S, it is the case that $conf([u-1, u-1]) <$ minConf, $conf([v+1, v+1]) <$ minConf, $conf([u, u]) >$ minConf and $conf([v, v]) >$ minConf.*

Proof: Note that $conf([u, v]) >$ minConf. As a result, if $conf([u-1, u-1]) >$ minConf, then $conf([u-1, v]) >$ minConf and since $gain([u-1, u-1]) > 0$, $gain([u-1, v]) > gain([u, v])$. Thus, the set $(S - \{[u, v]\}) \cup \{[u-1, v]\}$ has higher gain and is the optimized gain set – thus, leading to a contradiction. A similar argument can be used to show that $conf([v+1, v+1]) <$ minConf.

On the other hand, if $conf([u, u]) <$ minConf, then $gain([u, u]) < 0$ and since $conf([u, v]) >$ minConf, $conf([u+1, v]) >$ minConf. Also, $gain([u+1, v]) > gain([u, v])$, and thus the set $S - \{[u, v]\} \cup \{[u+1, v]\}$ has higher gain and is the optimized gain set – thus, leading to a contradiction. A similar argument can be used to show that $conf([v, v]) >$ minConf. ∎

From the above theorem, it follows that if $[u, v]$ is an interval in the optimized set, then values u and $u-1$ cannot both have confidences greater than or less than minConf – the same holds for values v and $v+1$. Thus, for a set of contiguous values if the confidence of each and every value is greater than (or is less than) minConf, then the optimized gain set either contains all of the values or none of them. Thus, an interval in the optimized set either contains all the values in a bucket

procedure optGain1D$(1, b, k)$
begin
1. PSet $:= \emptyset$, NSet $:= \{[1, b]\}$
2. **for** $i := 1$ **to** k {
3. Let P_q be the interval in PSet with the smallest value for $gain(min(P_q))$
4. Let N_q be the interval in NSet with the largest value for $gain(max(N_q))$
5. **if** $gain(min(P_q)) + gain(max(N_q)) < 0$ {
6. Delete P_q from PSet
7. Split P_q into three subintervals (with $min(P_q)$ as the middle interval)
8. Insert the first and third intervals to PSet and the second interval to NSet
9. }
10. **else** {
11. Delete N_q from NSet
12. Split N_q into three subintervals (with $max(N_q)$ as the middle interval)
13. Insert the first and third intervals to NSet and the second interval to PSet
14. }
15. }
16. **return** PSet
end

Figure 3: Algorithm for Computing Optimized Gain Set

or none of them – as a result, the optimized set can be computed using the buckets instead of the original values in the domain.

4.2 Algorithm for Computing Optimized Gain Set

In this subsection, we present an $O(bk)$ algorithm for the optimized gain problem for one dimension. The input to the algorithm is the b buckets generated by our bucketing scheme in Section 4.1 along with their confidences, supports and gains. The problem is to determine a set of at most k (non-overlapping) intervals such that the confidence of each interval is greater than or equal to minConf and gain of the set is maximized.

Note that due to our bucketing algorithm, buckets adjacent to a bucket with positive gain have negative gain, and vice versa. Thus, if there are at most k buckets with positive gain, then these buckets constitute the desired optimized gain set. Otherwise, procedure optGain1D, shown in Figure 3, is used to compute the optimized set. For an interval I, we denote by $max(I)$, the subinterval of I with maximum gain. Also, we denote by $min(I)$, the subinterval of I whose gain is minimum. Note that, for an interval I, $min(I)$ and $max(I)$ can be computed in time that is linear in the size of the interval. This is due to the following dynamic programming relationship for the gain of the subinterval of I with the maximum gain and ending at point u (denoted by $max(u)$): $max(u) = \max\{gain([u, u]), max(u-1) + gain([u, u])\}$. (A similar relationship can be derived for the subinterval with minimum gain).

The k desired intervals are computed by optGain1D in k iterations – the i^{th} iteration computes the i intervals with the maximum gain using the results of the $i - 1^{th}$ iteration. After the $i - 1^{th}$ iteration, PSet is the optimized gain set containing $i - 1$ intervals, while the remaining intervals not in PSet are stored in NSet. After P_q and N_q have been computed as described in steps 3–4, if $gain(min(P_q)) + gain(max(N_q)) < 0$, then it follows that the gain of $min(P_q)$ is more negative than the gain of $max(N_q)$ is positive. Thus, the best strategy for maximizing gain is to split P_q into

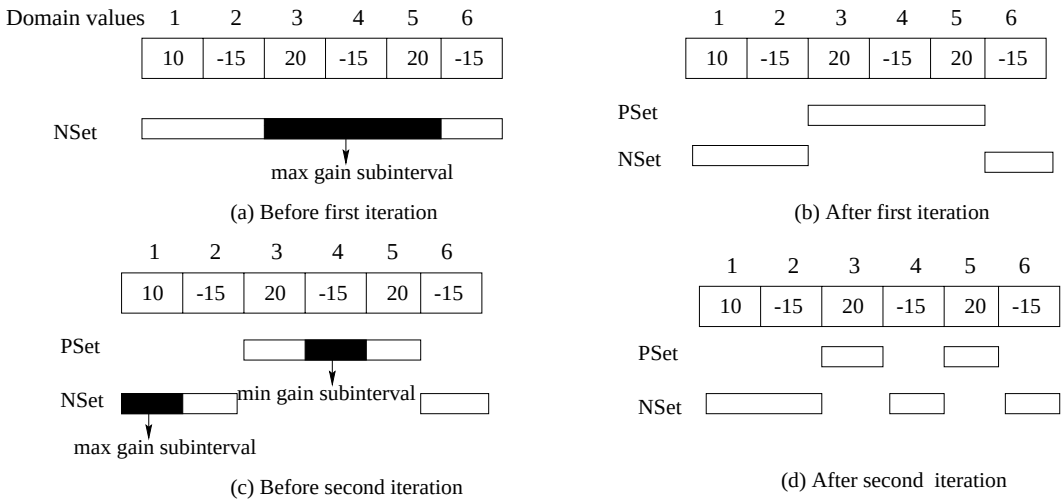


Figure 4: Execution Trace of Procedure optGain1D

two subintervals using $min(P_q)$ as the splitting interval, and include the two subintervals in the optimized gain set (steps 6–8). On the other hand, if $gain(min(P_q)) + gain(max(N_q)) \geq 0$, then the gain can be maximized by adding $max(N_q)$ to the optimized gain set (steps 11–13). Note that if PSet/NSet is empty, then we cannot compute P_q/N_q, and so $gain(min(P_q))/gain(max(N_q))$ in Step 5 is 0.

Example 4.2: Consider the 6 buckets $1, 2, \ldots, 6$ with gains 10, -15, 20, -15, 20 and -15 shown in Figure 4(a). We trace the execution of optGain1D assuming that we are interested in computing the optimized gain set containing 2 intervals.

Initially, NSet is set to {[1,6]} (see Figure 4(a)). During the first iteration of optGain1D, N_q is [1,6] since it is the only interval in NSet. Furthermore, $max(N_q) = [3, 5]$ (the dark subinterval in Figure 4(a)) and $gain(max(N_q)) = 25$. Since PSet is empty, $gain(min(P_q)) = 0$ and N_q is split into 3 intervals [1,2], [3,5] and [6,6], of which [3,5] is added to PSet, and [1,2] and [6,6] are added to NSet (after deleting [1,6] from it). The sets PSet and NSet at the end of the first iteration are depicted in Figure 4(b).

In the second iteration, $P_q = [3, 5]$ ($min(P_q) = [4, 4]$) and $N_q = [1, 2]$ ($max(N_q) = [1, 1]$) (since $gain(max([1, 2])) = 10$ is larger than $gain(max([6, 6])) = -15$). Thus, since $gain(min(P_q)) + gain(max(N_q)) = -5$, $[3, 5]$ is split into 3 intervals [3,3], [4,4] and [5,5], of which [3,3] and [5,5] are added to PSet (after deleting [3,5] from it), which is the desired optimized gain set. The dark subintervals in Figure 4(c) denote the minimum and maximum gain subintervals of P_q and N_q, respectively, and the final intervals in PSet and NSet (after the second iteration) are depicted in Figure 4(d). ∎

We can show that the above simple greedy strategy computes the i intervals with the maximum gain (in the i^{th} iteration). We first show that after the i^{th} iteration, the intervals in PSet and NSet satisfy the following conditions (let the i intervals in PSet be $P_1, \ldots, P_i$, and the remaining intervals in NSet be $N_1, \ldots, N_j$).

- **Cond 1:** Let $[u, v]$ be an interval in PSet. For all $u \leq l \leq v$, $gain([u, l]) \geq 0$ and $gain([l, v]) \geq 0$.
- **Cond 2:** Let $[u, v]$ be an interval in NSet. For all $u \leq l \leq v$, $gain([u, l]) \leq 0$ (except when $u = 1$) and $gain([l, v]) \leq 0$ (except when $v = b$).

- **Cond 3:** For all $1 \leq l \leq i$, $1 \leq m \leq j$, $gain(P_l) \geq gain(max(N_m))$.
- **Cond 4:** For all $1 \leq l \leq i$, $1 \leq m \leq j$, $gain(min(P_l)) \geq gain(N_m)$ (except for N_m that contain one of the endpoints, 1 or b).
- **Cond 5:** For all $1 \leq l, m \leq i$, $l \neq m$, $gain(min(P_l)) + gain(P_m) \geq 0$.
- **Cond 6:** For all $1 \leq l, m \leq j$, $l \neq m$, $gain(max(N_l)) + gain(N_m) \leq 0$ (except for N_m that contain one of the endpoints, 1 or b).

For an interval $[u, v]$ in PSet or NSet, conditions 1 and 2 state properties about the gain of its subintervals that contain u or v. Simply put, they state that extending or shrinking the intervals in PSet does not cause its gain to increase. Condition 3 states that the gain of PSet cannot be increased by replacing an interval in PSet by one contained in NSet, while conditions 4 and 5 state that splitting an interval in PSet and merging two other adjacent intervals in it or deleting an interval from it, cannot increase its gain either. Finally, Condition 6 covers the case in which two adjacent intervals in PSet are merged and an additional interval from NSet is added to it – Condition 6 states that these actions cannot cause PSet's gain to increase.

Lemma 4.3: *After the i^{th} iteration of procedure optGain1D, the intervals in PSet and NSet satisfy conditions 1–6.*

Proof: See Appendix. ∎

We can also show that any set of i intervals (in PSet) that satisfies all of the 6 above conditions is optimal with respect to gain.

Lemma 4.4: *Any set of i intervals satisfying conditions 1–6 is an optimized gain set.*

Proof: See Appendix. ∎

From the above two lemmas, we can conclude that at the end of the i^{th} iteration, procedure optGain1D computes the optimized gain set containing i intervals (in PSet).

Theorem 4.5: *Procedure optGain1D computes the optimized gain set.* ∎

It is straightforward to observe that the time complexity of procedure optGain1D is $O(bk)$ since it performs k iterations and in each iteration, intervals P_q and N_q can be computed in $O(b)$ steps.

5 Two Numeric Attributes

We next consider the problem of mining the optimized gain set for the case when there are two uninstantiated numeric attributes. In this case, we need to compute a set of k non-overlapping rectangles in two-dimensional space whose gain is maximum. Unfortunately, this problem in NP-hard [KMP98]. In the following subsection, we describe a dynamic programming algorithm with polynomial time complexity that computes approximations to optimized sets.

```
procedure optGain2D((i, j), (p, q), k)
begin
1.  if optSet[(i, j), (p, q), k] computed earlier
2.     return optSet[(i, j), (p, q), k]
3.  if conf([(i, j), (p, q)]) ≥ minConf
4.     optSet[(i, j), (p, q), k] := {[(i, j), (p, q)]};
5.  optSet_1 := optSet_2 := ∅
6.  if p > i {
7.     if k = 1
8.        optSet_1 = maxGainSet(optGain2D((i, j), (p − 1, q), k), optGain2D((i + 1, j), (p, q), k))
9.     else
10.       for l := i to p − 1 do
11.          for m := 1 to k − 1 do
12.             optSet_1 = maxGainSet(optSet_1, optGain2D((i, j), (l, q), m) ∪ optGain2D((l + 1, j), (p, q), k − m))
13. }
14. if q > j {
15.    if k = 1
16.       optSet_2 = maxGainSet(optGain2D((i, j), (p, q − 1), k), optGain2D((i, j + 1), (p, q), k))
17.    else
18.       for l := j to q − 1 do
19.          for m := 1 to k − 1 do
20.             optSet_2 = maxGainSet(optSet_2, optGain2D((i, j), (p, l), m) ∪ optGain2D((i, l + 1), (p, q), k − m))
21. }
22. optSet[(i, j), (p, q), k] := maxGainSet(optSet[(i, j), (p, q), k],optSet_1, optSet_2)
23. return optSet[(i, j), (p, q), k]
end
```

Figure 5: Dynamic Programming Algorithm for Computing Optimized Gain Set

5.1 Approximation Algorithm Using Dynamic Programming

The procedure optGain2D (see Figure 5) for computing approximate optimized gain sets is a dynamic programming algorithm that uses simple end to end horizontal and vertical cuts for splitting each rectangle into two subrectangles. Procedure optGain2D accepts as input parameters, the coordinates of the lower left $((i, j))$ and upper right $((p, q))$ points of the rectangle for which the optimized set is to be computed. These two points completely define the rectangle. The final parameter is the bound on the number of rectangles that the optimized set can contain. The array optSet$[(i, j), (p, q), k]$ is used to store the optimized set with size at most k for the rectangle, thus preventing recomputations of the optimized set for the rectangle. The confidence, support and gain for each rectangle is precomputed – this can be done in $O(n^4)$ steps which is proportional to the total number of rectangles possible.

In optGain2D, the rectangle $[(i, j), (p, q)]$ is first split into two subrectangles using vertical cuts (Steps 6–13), and later horizontal cuts are employed (Steps 14–21). For $k > 1$, vertical cuts between i and $i+1$, $i+1$ and $i+2$, ..., $p-1$ and p are used to divide rectangle $[(i, j), (p, q)]$ into subrectangles $[(i, j), (l, q)]$ and $[(l + 1, j), (p, q)]$ for all $i \leq l \leq p - 1$. For every pair of subrectangles generated above, optimized sets of size k_1 and k_2 are computed by recursively invoking optGain2D for all k_1, k_2 such that $k_1 + k_2 = k$. An optimization can be employed in case $k = 1$ (Step 7), and instead of considering every vertical cut, it suffices to only consider the vertical cuts at the ends since the single optimized rectangle must be contained in either $[(i, j), (p - 1, q)]$ or $[(i + 1, j), (p, q)]$. After

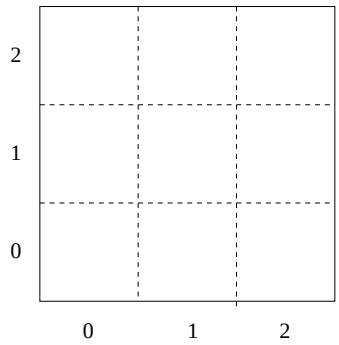


Figure 6: Rectangle [(0,0),(2,2)]

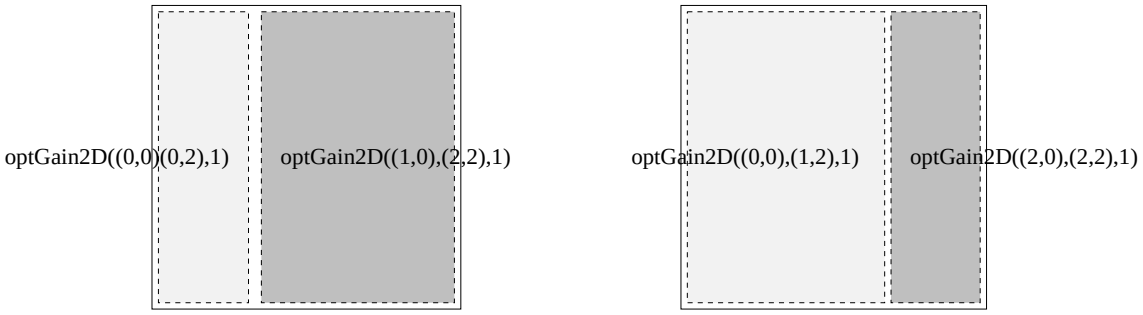


Figure 7: Vertical Cuts for optGain2D((0,0),(2,2),2)

similarly generating pairs of subrectangles using horizontal cuts, the optimized set for the original rectangle is set to the union of the optimized sets for the pair with the maximum gain (function maxGainSet returns the set with the maximum gain from among its inputs).

Example 5.1: Consider the two-dimensional rectangle [(0,0),(2,2)] in Figure 6 for two numeric attributes, each with domain $\{0, 1, 2\}$. We trace the execution of optGain2D for computing the optimized gain set containing 2 non-overlapping rectangles.

Consider the first invocation of optGain2D. In the body of the procedure, the variable l is varied from 0 to 1 for both vertical and horizontal cuts (Steps 10 and 18). Further, the only value for variable m is 1 since k is 2 in the first invocation. Thus, in Steps 10–12 and 18–20, the rectangle [(0,0),(2,2)] is cut at two points in each of the vertical and horizontal directions, and optGain2D is called recursively for the two subrectangles due to each cut. These subrectangle pairs for the horizontal and vertical cuts are illustrated in Figures 7 and 8, respectively.

Continuing further with the execution of the first recursive call of optGain2D with rectangle [(0,0),(0,2)] and $k = 1$, observe that the boundary points of the rectangle along the horizontal axis are the same. As a result, since $k = 1$, in Step 16, optGain2D recursively invokes itself with two subrectangles, each of whose size is one unit smaller along the vertical axis. These subrectangles are depicted in Figure 9. The process of recursively splitting rectangles is repeated for the newly generated subrectangles as well as rectangles due to previous cuts.

∎

The number of points input to our dynamic programming algorithm for the two-dimensional case is $N = n^2$ since n is the size of the domain of each of the two uninstantiated numeric attributes.

Theorem 5.2: *The time complexity of Procedure optGain2D is* $O(N^{2.5}k^2)$.

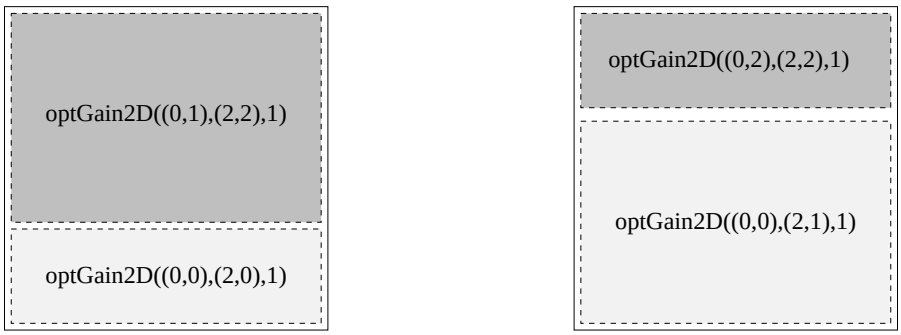


Figure 8: Horizontal Cuts for optGain2D((0,0),(2,2),2)

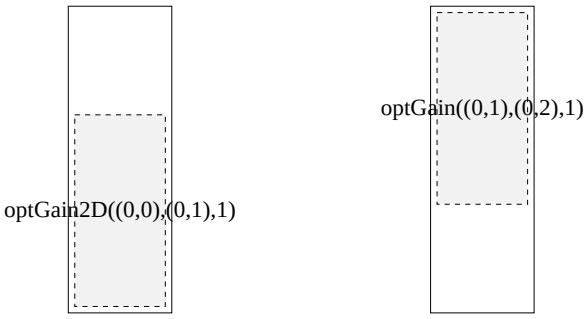


Figure 9: Smaller Rectangles Recursively Considered by optGain2D((0,0),(0,2),1)

Proof: The complexity of procedure optGain2D is simply the number of times procedure optGain2D is invoked multiplied by a constant. The reason for this is that steps in optGain2D that do not have constant overhead (e.g., the for loops in Steps 12 and 13) result in recursive calls to optGain2D. Thus, the overhead of these steps is accounted for in the count of the number of calls to optGain2D. Consider an arbitrary rectangle $[(i,j),(p,q)]$, and consider an arbitrary $1 \leq l \leq k$. We show that optGain2D is invoked with the above parameters at most $4nk$ times. Thus, since the number of rectangles is at most n^4 and l can take k possible values, the complexity of the algorithm is $O(n^5k^2)$.

We now show that optGain2D with a given set of parameters can be invoked at most $4nk$ times. The first observation is that optGain2D with rectangle $[(i,j),(p,q)]$ and l is invoked only from a different invocation of optGain2D with a rectangle that on being cut vertically or horizontally yields rectangle $[(i,j),(p,q)]$ and with a value for k that lies between l and k. The number of such rectangles is at most $4n$ – each of these rectangles can be obtained from $[(i,j),(p,q)]$ by stretching it in one of four directions, and there are only n possibilities for stretching a rectangle in any direction. Thus, each invocation of optGain2D can result from $4nk$ different invocations of optGain2D. Furthermore, since the body of optGain2D for a given set of input parameters is executed only once, optGain2D with a given input is invoked at most $4nk$ times. ∎

5.2 Optimality Results

Procedure optGain2D's approach of splitting each rectangle into two subrectangles and then combining the optimized sets for each subrectangle may not yield the optimized set for the original rectangle. This point is further illustrated in Figure 10(a) that shows a rectangle and the optimized set of rectangles for it. It is obvious that there is no way to split the rectangle into two subrectangles

Figure 10: Binary Space Partitionable Rectangles

such that each rectangle in the optimized set is completely contained in one of the subrectangles. Thus, a dynamic programming approach that considers all possible splits of the rectangle into two subrectangles (using horizontal and vertical end-to-end cuts) and then combines the optimized sets for the subrectangles may not result in the optimized set for the original rectangle being computed.

In the following, we first identify restrictions under which optGain2D yields optimized sets. We then show bounds on how far the computed approximation for the general case can deviate from the optimal solution.

Let us define a set of rectangles to be *binary space partitionable* if it is possible to recursively partition the plane such that no rectangle is cut, and each partition contains at most one rectangle. The set of rectangles in Figure 10(b) is binary space partitionable (the bold lines are a partitioning of the rectangles) – however, the set in Figure 10(a) is not.

If we are willing to restrict the optimized set to only binary space partitionable rectangles, then we can show that procedure optGain2D computes the optimized set. Note that any set of 3 or fewer rectangles in a plane is always binary space partitionable. Thus, for $k \leq 3$, optGain2D computes the optimized gain set.

Theorem 5.3: *Procedure optGain2D computes the optimized set of binary space partitionable rectangles.*

Proof: The proof is by induction on the size of the rectangles that optGain2D is invoked with.
Basis: For all $1 \leq l \leq k$, optGain2D can be trivially shown to compute the optimized binary space partitionable set for the unit rectangle $[(i,i),(i,i)]$ (if confidence of the rectangle is at least minConf, then the optimized set is the rectangle itself).
Induction: We next show that for any $1 \leq l \leq k$ and rectangle $[(i,j),(p,q)]$, the algorithm computes the optimized binary space partitionable set (assuming that for all its subrectangles, for all $1 \leq l \leq k$, the algorithm computes the optimized binary space partitionable set). We need to consider three cases. The first is when the optimized set is the rectangle $[(i,j),(p,q)]$ itself. In this case, $conf([(i,j),(p,q)]) \geq$ minConf and optSet is correctly set to $\{[(i,j),(p,q)]\}$ by optGain2D. In case $l = 1$, then since $conf([(i,j),(p,q)]) <$ minConf, the optimized rectangle must be contained in one of the 4 largest subrectangles in $[(i,j),(p,q)]$, and thus the optimized set of size 1 for these subrectangles whose support is maximum is the optimized set for $[(i,j),(p,q)]$. Finally, if $l > 1$, then since we are interested in computing the optimized binary space partitionable set, there must exist a horizontal or vertical cut that cleanly partitions the optimized rectangles, one of the two subrectangles due to the cut containing at most $1 \leq r < l$ and the other containing $l - r$ rectangles of the optimized set. Thus, since in optGain2D, all possible cuts are considered and optSet is then set to the union of the optimized sets for the subrectangles such that the resulting gain is

maximum, due to the induction hypothesis, it follows that this is the optimized gain set for the rectangle $[(i,j),(p,q)]$. ∎

We next use this result in order to show that in the general case, the approximate optimized gain set computed by procedure optGain2D is within a factor of $\frac{1}{4}$ of the optimized gain set. The proof also uses a result from [AF92], in which it is shown that for any set of rectangles in a plane, there exists a *binary space partitioning* (that is, a recursive partitioning) of the plane such that each rectangle is cut into at most four subrectangles and each partition contains at most one subrectangle.

Theorem 5.4: *Procedure optGain2D computes an optimized gain set whose gain is greater than or equal to $\frac{1}{4}$ times the gain of the optimized gain set.*

Proof: From the result in [AF92], it follows that it is possible to partition each rectangle in the optimized set into four subrectangles such that the set of subrectangles is binary space partitionable. Furthermore, for each rectangle, consider its subrectangle with the highest gain. The gain of each such subrectangle is at least $\frac{1}{4}$ times the gain for the original rectangle. Thus, the set of these subrectangles is binary space partitionable and has $\frac{1}{4}$ of the gain of the optimized set. As a result, due to Theorem 5.3 above, it follows that the optimized set computed by optGain2D has gain that is at least $\frac{1}{4}$ times the gain of the optimized set. ∎

6 Experimental Results

In this section, we study the performance of our algorithms for computing optimized gain sets for the one-dimensional and two-dimensional cases. In particular, we show that our algorithm is highly scaleable for one dimension. For instance, we can tackle attribute domains with sizes as high as one million in a few minutes. For two dimensions, however, the high time and space complexities of our dynamic programming algorithm make it less suitable for large domain sizes and large number of disjunctions. We also present results of our experiments with a real-life population survey data set where optimized gain sets enable us to discover interesting correlations among attributes.

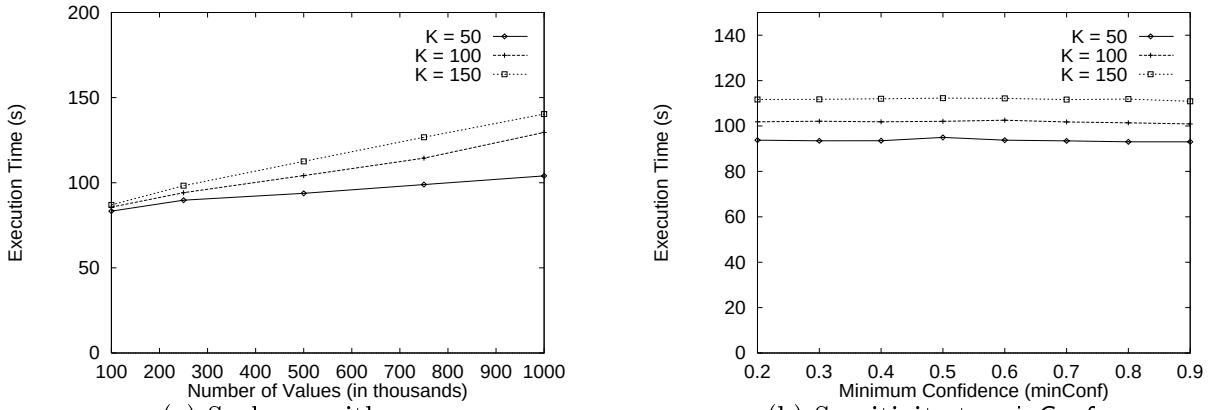
In our experiments, the data file is read only once at the beginning in order to compute the gain for every point. The time for this, in most cases, constitutes a tiny fraction of the total execution time of our algorithms. Thus, we do not include the time spent on reading the data file in our results. Furthermore, note that the performance of our algorithms does not depend on the number of tuples in the data file – it is more sensitive to the size of the attribute's domain n and the number of intervals k. We fixed the number of tuples in the data file to be 10 million in all our experiments. Our experiments were performed on a Sun Ultra-2/200 machine with 512 MB of RAM and running Solaris 2.5.

6.1 Performance Results on Synthetic Datasets

The association rule that we experimented with, has the form $U \wedge C_1 \rightarrow C_2$ where U contains 1 or 2 uninstantiated attributes (see Section 3) whose domains consist of integers ranging from 1 to n. Every instantiation of $U \wedge C_1 \rightarrow C_2$ is (that is, point in m-dimensional space) is assigned a randomly generated confidence between 0 and 1 with uniform distribution. Each value in m-dimensional space is also assigned a randomly generated support between 0 and $\frac{2}{n^m}$ with uniform distribution; thus, the average support for a value is $\frac{1}{n^m}$.

n	500	1000	2000	5000	7500	10000	50000	100000
b	195	354	738	1829	2759	3711	16321	32266

Table 1: Values of b for different domain sizes



(a) Scale-up with n (b) Sensitivity to minConf

Figure 11: Performance Results for One-Dimensional Data

6.1.1 One-Dimensional Data

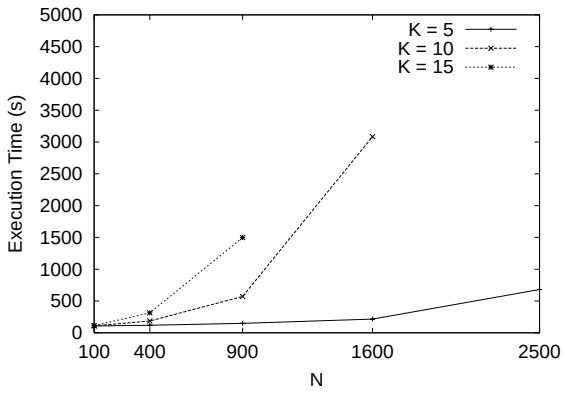
Bucketing: We begin by studying the reduction in input size due to the bucketing optimization. Table 1 illustrates the number of buckets for domain sizes ranging from 500 to 100,000 when minConf is set to 0.5. From the table, it follows that bucketing can result in reductions to input size as high as 65%.

Scale-up with n: The graph in Figure 11(a) plots the execution times of our algorithm for computing optimized gain sets as the domain size is increased from 100,000 to 1 million for a minConf value of 0.5. Note that for this experiment, we turned off the bucketing optimization – so the running times would be even smaller if we were to employ bucketing to reduce the input size. The experiments validate our earlier analytical results on the $O(bk)$ time complexity of procedure optGain1D. As can be seen from the figure, our optimized gain set algorithm scales linearly with the domain size as well as k.

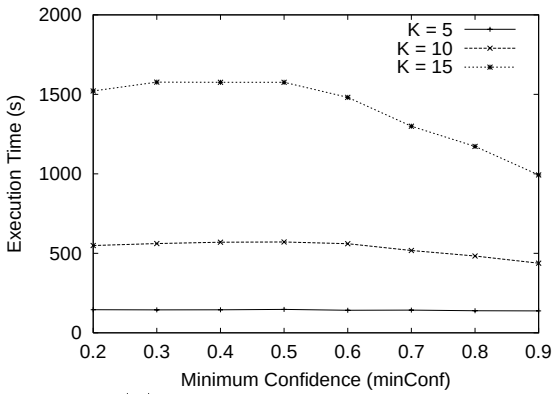
Sensitivity to minConf: Figure 11(b) depicts the running times for our algorithm for a range of confidence values and a domain size of 500,000. From the graphs, it follows that the performance of procedure optGain1D is not affected by values for minConf.

6.1.2 Two-Dimensional Data

Scale-up with n: The graph in Figure 12(a) plots the execution times of our dynamic programming algorithm for computing optimized gain sets as the domain sizes n x n are increased from 10 x 10 to 50 x 50 in increments of 10. The value of minConf is set to 0.5 and in the figure, each point is represented along the x-axis with the value of $N = n^2$ (e.g., 10 x 10 is represented as 100). Note that for this experiment, we do not perform bucketing since this optimization is applicable only to one-dimensional data. The running times corroborate our earlier analysis of the time complexity of $O(N^{2.5}k^2)$ for procedure optGain2D. Note that due to the high space complexity of $O(N^2k)$ for our dynamic programming algorithm, we could not measure the execution times for large values



(a) Scale-up with N



(b) Sensitivity to minConf

Figure 12: Performance Results for Two-Dimensional Data

of N and k. For these large parameter settings, the system returned an "out-of-memory" error message.

Sensitivity to minConf: Figure 12(b) depicts the running times of our algorithm for a range of confidence values and a domain size of 30 x 30. From the graph, it follows that the performance of procedure optGain2D improves with increasing values for minConf. The reason for this is that at high confidence values, fewer points have positive gain, and thus optSet, in most cases, is either empty or contains a small number of rectangles. As a result, operations on optSet like union (Steps 12 and 20) and assignment (Steps 8, 12, 16, 20 and 22) are much faster, and consequently the execution time of the procedure is lower for high confidence values.

6.2 Experiences with Real-life Datasets

In order to gauge the efficacy of our optimized rule framework for discovering interesting patterns, we conducted experiments with a real-life data set consisting of the current population survey (CPS) data for the year 1995[1]. The CPS is a monthly survey of about 50,000 households and is the primary source of information on the labor force characteristics of the U.S. population. The CPS data consists of a variety of attributes that include age (A_AGE), group health insurance coverage (COV_HI), hours of work (HRS_WK), household income (HTOTVAL), temporary work experience for a few days (WTEMP), and unemployment compensation benefits received Y/N-Person (UC_YN). The number of records in the 1995 survey is 149642.

Suppose we are interested in finding the age groups that contain a large concentration of temporary workers. This may be of interest to a job placement/recruiting company that is actively seeking temporary workers. Knowledge of the demographics of temporary workers can help the company target its advertizements better and locate candidate temporary workers more effectively. This correlation between age and temporary worker status can be computed in our optimized rule framework using the following rule: A_AGE $\in [l, u]) \rightarrow$ WTEMP = YES. The optimized gain regions found with minConf = 0.01 (1%) for this rule is presented in Table 2. (The domain of the A_AGE attribute varies from 0 to 90 years.) From the table, it follows that there is a high concentration of temporary workers among young adults (age between 15 and 23) and seniors (age between 62 and 69). Thus, advertizing on television programs or web sites that are popular among

[1]This data can be downloaded from http://www.bls.census.gov/cps.

k	Optimized Gain Regions	Total Gain
1	[15, 23]	177.01
2	[15, 23], [62, 69]	181.74
3	[15, 23], [62, 69], [76, 76]	186.04
4	[15, 23], [59, 59], [62, 69], [76, 76]	188.06
5	[15, 23], [59, 59], [62, 63], [67, 69], [76, 76]	189.22
6	[15, 23], [59, 59], [62, 63], [65, 65], [67, 69], [76, 76]	189.44

Table 2: Optimized Gain Regions for **age** $\in [l, u]) \rightarrow$ WTEMP = YES

k	Optimized Gain Regions	Total Gain
1	[61, 90]	692.32
2	[15, 21], [61, 90]	1117.56
3	[15, 21], [59, 59] [61, 90]	1119.48
4	[15, 21], [52, 52], [59, 59] [61, 90]	1119.96
5	[15, 21], [52, 52], [59, 59] [61, 90]	1119.96

Table 3: Optimized Gain Regions for **age** $\in [l, u]) \rightarrow$ UC_YN = NO

these age groups would be an effective strategy to reach candidate temporary workers. Also, observe that the improvement in the total gain slows for increasing k until it reaches a point where increasing the number of gain regions does not affect the total gain.

We also ran our algorithm to find optimized gain regions for the rule A_AGE $\in [l, u]) \rightarrow$ UC_YN = NO with minConf = 0.95 (95%). The results of this experiment are presented in Table 3 and can be used by the government, for example, to design appropriate training programs for unemployed individuals based on their age. The time to compute the optimized set for both real-life experiments was less than 1 second.

7 Concluding Remarks

In this paper, we generalized the optimized gain association rule problem by permitting rules to contain upto k disjunctions over one or two uninstantiated numeric attributes. For one attribute, we presented an $O(nk)$ algorithm for computing the optimized gain rule, where n is the number of values in the domain of the uninstantiated attribute. We also presented a bucketing optimization that coalesces contiguous values – all of which have confidence either greater than the minimum specified confidence or less than the minimum confidence. For two attributes, we presented a *dynamic programming* algorithm that computes approximate gain rules – we showed that the approximations are within a constant factor of the optimized rule using recent results on binary space partitioning. For a single numeric attribute, our experimental results with synthetic data sets demonstrate the effectiveness of our bucketing optimization and the linear scale-up for our algorithm for computing optimized gain sets. For two numeric attributes, however, the high time and space complexities of our dynamic programming based approximation algorithm make it less suitable for large domain sizes and large number of disjunctions. Finally, our experiments with the real-life population survey data set indicate that our optimized gain rules can indeed be used to unravel interesting correlations among attributes.

Acknowledgements: Without the support of Yesook Shim, it would have been impossible to

complete this work. The work of Kyuseok Shim was partially supported by the Korea Science and Engineering Foundation (KOSEF) through the Advanced Information Technology Research Center (AITrc).

References

[AF92] F. D. Amore and P. G. Franciosa. On the optimal binary plane partition for sets of isothetic rectangles. *Information Processing Letters*, 44(5):255–259, December 1992.

[AIS93] Rakesh Agrawal, Tomasz Imielinski, and Arun Swami. Mining association rules between sets of items in large databases. In *Proc. of the ACM SIGMOD Conference on Management of Data*, pages 207–216, Washington, D.C., May 1993.

[AS94] Rakesh Agrawal and Ramakrishnan Srikant. Fast algorithms for mining association rules. In *Proc. of the VLDB Conference*, Santiago, Chile, September 1994.

[BA99] Roberto J. Bayardo and Rakesh Agrawal. Mining the most interesting rules. In *Proc. of the ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, San Diego, 1999.

[BAG97] Roberto J. Bayardo, Rakesh Agrawal, and Dimitrios Gunopulos. Constraint-based rule mining in large, dense databases. In *Int'l Conference on Data Engineering*, 1997.

[FMMT96a] Takeshi Fukuda, Yasuhiko Morimoto, Shinichi Morishita, and Takesh Tokuyama. Mining optimized association rules for numeric attributes. In *Proc. of the ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, June 1996.

[FMMT96b] Takeshi Fukuda, Yasuhiko Morimoto, Shinichi Morishita, and Takesh Tokuyama. Data mining using two-dimensional optimized association rules: Scheme, algorithms, and visualization. In *Proc. of the ACM SIGMOD Conference on Management of Data*, June 1996.

[HF95] J. Han and Y. Fu. Discovery of multiple-level association rules from large databases. In *Proc. of the VLDB Conference*, Zurich, Switzerland, September 1995.

[JKM+98] H. V. Jagadish, N. Koudas, S. Muthukrishnan, V. Poosala, K. Sevcik, and T. Suel. Optimal histograms with quality guarantees. In *Proc. of the VLDB Conference*, New York, August 1998.

[KMP98] S. Khanna, S. Muthukrishnan, and M. Paterson. On approximating rectangle tiling and packing. In *Proc. 9th Annual Symposium on Discrete Algorithms (SODA)*, pages 384–393, 1998.

[LSW97] Brian Lent, Arun Swami, and Jennifer Widom. Clustering association rules. In *Int'l Conference on Data Engineering*, Brmingham, U.K., April 1997.

[MTV94] Heikki Mannila, Hannu Toivonen, and A. Inkeri Verkamo. Efficient algorithms for discovering association rules. In *KDD-94: AAAI Workshop on Knowledge Discovery in Databases*, pages 181–192, Seattle, Washington, July 1994.

[PCY95] Jong Soo Park, Ming-Syan Chen, and Philip S. Yu. An effective hash based algorithm for mining association rules. In *Proc. of the ACM-SIGMOD Conference on Management of Data*, San Jose, California, May 1995.

[RS98] R. Rastogi and K. Shim. Mining optimized association rules for categorical and numeric attributes. In *Int'l Conference on Data Engineering*, Orlando, 1998.

[RS99] R. Rastogi and K. Shim. Mining optimized support rules for numeric attributes. In *Int'l Conference on Data Engineering*, Sydney, Austrailia, 1999. To appear.

[SA95] Ramakrishnan Srikant and Rakesh Agrawal. Mining generalized association rules. In *Proc. of the VLDB Conference*, Zurich, Switzerland, September 1995.

[SA96] Ramakrishnan Srikant and Rakesh Agrawal. Mining quantitative association rules in large relational tables. In *Proc. of the ACM SIGMOD Conference on Management of Data*, June 1996.

[SON95] A. Savasere, E. Omiecinski, and S. Navathe. An efficient algorithm for mining association rules in large databases. In *Proc. of the VLDB Conference*, Zurich, Switzerland, September 1995.

-Appendix-

Proof of Lemma 4.3: We use induction to show that after iteration i, the intervals in PSet and NSet satisfy conditions 1–6.

Basis ($i = 0$): The 6 conditions trivially hold initially before the first iteration begins.

Induction Step: Let us assume that the $i-1$ intervals in PSet satisfy conditions 1–6 after the $i-1^{th}$ iteration completes. Let $P_1, \ldots, P_{i-1}$ be the $i-1$ intervals in PSet and $N_1, \ldots, N_j$ be the intervals in NSet after the $i-1^{th}$ iteration. During each iteration either P_q is split into 2 subintervals using $min(P_q)$ which are then added to PSet or $max(N_q)$ is added to PSet. Both actions result in 3 subintervals that satisfy conditions 1 and 2. For instance, let interval $P_q = [u, v]$ be split into 3 intervals and let $[s, t]$ be $min(P_q)$. For all $u \leq l \leq s-1$, it must be the case that gain$([l, s-1]) \geq 0$ since otherwise $[s, t]$ would not be the interval with the minimum gain. Similarly, it can be shown that for all $s \leq l \leq t$, it must be the case that $gain([s, l]) \leq 0$ since otherwise $[s, t]$ would not be the minimum gain subinterval in $[u, v]$. The other cases can be shown using a similar argument.

Next we show that splitting interval $P_q = [u, v]$ in Plist into 3 subintervals with $[s, t] = min([u, v])$ as the middle interval, preserves the remaining four conditions.

- **Cond 3:** We need to show that $gain([u, s-1])$ and $gain([t+1, v]) \geq gain(max([s, t]))$, $gain([u, s-1])$ and $gain([t+1, v]) \geq gain(max(N_m))$, and $gain(P_l) \geq gain(max([s, t]))$. We can show that $gain([s, t]) + gain(max([s, t])) \leq 0$ since otherwise the subinterval of $[s, t]$ preceding $max([s, t])$ would have smaller gain than $[s, t]$ (every subinterval of $[s, t]$ with endpoint at s has gain ≤ 0 and so the sum of the gains of $max([s, t])$ and the subinterval of $[s, t]$ preceding $max([s, t])$ is ≤ 0). Since every subinterval of $[u, v]$ with u as an endpoint has gain ≥ 0, it follows that $gain([u, s-1]) + gain([s, t]) \geq 0$. Thus, it follows that $gain([u, s-1]) \geq gain(max([s, t]))$. Similarly, we can show that $gain([t+1, v]) \geq gain(max([s, t]))$.

  Note that since $P_q = [u, v]$ is chosen for splitting, it must be the case that $gain(max(N_m)) + gain([s, t]) \leq 0$. Also, since $gain([u, s-1]) + gain([s, t]) \geq 0$, we can deduce that $gain([u, s-1]) \geq gain(max(N_m))$. Using an identical argument, it can be shown that $gain([t+1, v]) \geq gain(max(N_m))$.

  Finally, for every P_l, $gain(P_l) + gain([s, t]) \geq 0$ (due to Condition 5). Combining this with $gain([s, t]) + gain(max([s, t])) \leq 0$, we obtain that $gain(P_l) \geq gain(max([s, t]))$.

- **Cond 4:** The preservation of this condition follows because $gain(min(P_l)) \geq gain([s, t])$ since $[s, t]$ is chosen such that it has the minimum gain from among all the $min(P_l)$. The same argument can be used to show that $gain(min([u, s-1]))$ and $gain(min([t+1, v])) \geq gain([s, t])$. Also, since $gain([s, t]) \geq gain(N_m)$ (due to Condition 4) and $[s, t]$ is the subinterval of P_q with the minimum gain, it follows that $gain(min([u, s-1]))$ and $gain(min([t+1, v])) \geq gain(N_m)$.

- **Cond 5:** $gain(min[u, s-1]) + gain(P_m) \geq 0$ since $gain([s, t]) + gain(P_m) \geq 0$ (due to Condition 5) and $gain(min([u, s-1])) \geq gain([s, t])$ ($[s, t]$ is the subinterval with the minimum gain in $[u, v]$). Similarly, we can show that $gain(min([t+1, v])) + gain(P_m) \geq 0$. Also, we need to show that $gain(min(P_l)) + gain([u, s-1]) \geq 0$. This follows since $gain([u, s-1]) + gain([s, t]) \geq 0$ and $gain(min(P_l)) \geq gain([s, t])$. Similarly, it can be shown that $gain(min(P_l)) + gain([t+1, v]) \geq 0$.

- **Cond 6:** Since $[s, t]$ is used to split P_q, it is the case that $gain(max(N_m)) + gain([s, t]) \leq 0$. We next need to show that $gain(max([s, t])) + gain(N_m) \leq 0$. Due to Condition 4, $gain([s, t]) \geq gain(N_m)$ and as shown in the proof for Condition 3 above, $gain([s, t]) + gain(max([s, t])) \leq 0$. Combining the two, we obtain $gain(max([s, t])) + gain(N_m) \leq 0$.

Next we show that splitting interval $N_q = [u, v]$ in Nlist into 3 intervals with $[s, t] = max([u, v])$ as the middle interval, preserves the remaining four conditions.

- **Cond 3:** Due to Condition 3, $gain(P_l) \geq gain([s, t])$ and $gain([s, t]) \geq gain(max([u, s-1]))$ since $[s, t]$ is the subinterval with maximum gain in $[u, v]$. Thus, $gain(P_l) \geq gain(max([u, s-1]))$ and we can also similarly show that $gain(P_l) \geq gain(max([t+1, v]))$. Also, since $[s, t]$ has the maximum gain from among $max(N_m)$, it follows that $gain([s, t]) \geq gain(max(N_m))$.

- **Cond 4:** We first show that $gain(min([s, t])) + gain([s, t]) \geq 0$ since if $gain(min([s, t])) + gain([s, t]) < 0$, then the subinterval of $[s, t]$ preceding $min([s, t])$ would have gain greater than $[s, t]$ (since every subinterval of $[s, t]$ begining at s has gain greater than or equal to 0, it follows that the sum of the gains of $min([s, t])$ and the subinterval preceding $min([s, t])$ in $[s, t]$ is ≥ 0). Also since every subinterval of N_q begining at u has gain less than or equal to 0 (assuming $u \neq 1$), we get $gain([u, s-1]) + gain([s, t]) \leq 0$. Combining the two, we get $gain(min([s, t])) \geq gain([u, s-1])$ (except when $[u, s-1]$ contains an endpoint). Using an identical argument, we can also show that $gain(min([s, t])) \geq gain([t+1, v])$ (except when $[t+1, v]$ contains an endpoint).

  Also, due to Condition 6, $gain([s, t]) + gain(N_m) \leq 0$ which when combined with $gain(min([s, t])) + gain([s, t]) \geq 0$ implies that $gain(min([s, t])) \geq gain(N_m)$ (except when N_m contains an endpoint). Finally, since $[s, t]$ is used to split interval N_q, it follows that $gain(min(P_l)) + gain([s, t]) \geq 0$. Furthermore, if $u \neq 1$, we have $gain([u, s-1]) + gain([s, t]) \leq 0$. Combining the two, we get $gain(min(P_l)) \geq gain([u, s-1])$ (except when $[u, s-1]$ contains an endpoint). Similarly, we can show that $gain(min(P_l)) \geq gain([t+1, v])$ (except when $[t+1, v]$ contains an endpoint).

- **Cond 5:** Since N_q is the interval chosen for splitting, $gain(min(P_l)) + gain([s, t]) \geq 0$. Due to Condition 3, $gain(P_l) \geq gain([s, t])$ and as shown in the proof of Condition 4 above, $gain(min([s, t])) + gain([s, t]) \geq 0$ from which we can deduce that $gain(min([s, t])) + gain(P_l) \geq 0$.

- **Cond 6:** Since N_q is the interval chosen for splitting, $gain(max(N_l)) \leq gain([s, t])$. Also, since $gain([u, s-1]) + gain([s, t]) \leq 0$ (except when $u = 1$), we get $gain(max(N_l)) + gain([u, s-1]) \leq 0$ (except when $[u, s-1]$ contains an endpoint). Similarly, we can show that $gain(max(N_l)) + gain([t+1, v]) \leq 0$ (except when $[t+1, v]$ contains an endpoint).

  Again, since $[s, t]$ is used to split N_q, $gain(max([u, s-1])) \leq gain([s, t])$. Also, due to Condition 6, $gain([s, t]) + gain(N_m) \leq 0$ thus yielding $gain(max([u, s-1])) + gain(N_m) \leq 0$ (except when N_m contains an endpoint). Using a similar argument it can be shown that $gain(max([t+1, v])) + gain(N_m) \leq 0$ (except when N_m contains an endpoint). ∎

Proof of Lemma 4.4: Let $P_1, P_2, \ldots, P_i$ be i intervals in PSet satisfying conditions 1–6. In order to show that this is an optimized gain set, we show that the gain of every other set of i intervals is no larger than that of $P_1, P_2, \ldots, P_i$. Consider any set of i intervals $\text{PSet}' = P'_1, P'_2, \ldots, P'_i$. We transform these set of intervals in a series of steps to $P_1, P_2, \ldots, P_i$. Each step ensures that the gain of the successive set of intervals is at least as high as the preceeding set. As a result, it follows that the gain of $P_1, P_2, \ldots, P_i$ is at least as high as any other set of i intervals and thus $\{P_1, \ldots, P_i\}$ is an optimized gain set.

The steps involved in the transformation are as follows:

1. For an interval $P'_j = [u, v]$ that intersects one of the P_ls do the following: if $u \in P_l$ for some $P_l = [s, t]$, then if $[s, u-1]$ does not intersect any other P'_ms, modify P'_j to be $[s, v]$ (that is, delete $[u, v]$ from PSet$'$ and add $[s, v]$ to PSet$'$). Note that due to condition 1, $gain([s, u-1]) \geq 0$ and so $gain([s, v]) \geq gain([u, v])$. Similarly, if $v \in P_l$, for some $P_l = [s, t]$, then if $[v+1, t]$ does not intersect any other P'_ms, modify P'_j to be $[u, t]$.

   On the other hand, if for an interval $P'_j = [u, v]$ that intersects one of the P_ls, if $u \notin P_l$ for all P_l, then let m be the max value such that $[u, m]$ does not intersect with any of the P_ls. Modify P'_j to be $[m+1, v]$. Note that due to condition 2, $gain([u, m]) \leq 0$ and so $gain([m+1, v]) \geq gain([u, v])$. Similarly, if $v \notin P_l$ for all P_l, then let m be the min value such that $[m, v]$ does not intersect with any of the P_ls. Modify P'_j to be $[u, m-1]$.

   Thus, at the end of Step 1, each interval P'_j in PSet$'$ (some of which may have been modified) either does not intersect any P_l or if it does intersect an interval P_l, then each endpoint of P'_j lies in some interval P_m, and each endpoint of P_l lies in some interval P'_m. Also, note that if two intervals P_l and P'_j overlap and intersect with no other intervals, then at the end of Step 1, $P'_j = P_l$.

2. In this step we transform all P'_js in PSet$'$ that intersect multiple P_ls. Consider a $P'_j = [u, v]$ that intersects multiple P_ls (that is, spans an $N_m = [s, t]$). Thus, since there are k intervals, we need to consider two possible cases: 1) there is a P'_m that does not intersect with any P_l, and 2) some P_l intersects with multiple P'_ms. For case 1, due to condition 6, it follows that $gain(P'_m) + gain([s, t]) \leq 0$, and thus deleting P'_m from PSet$'$ and splitting P'_j into $[u, s-1]$ and $[t+1, v]$ (that is, deleting $[u, v]$ from PSet$'$, and adding $[u, s-1]$ and $[t+1, v]$ to it) does not cause the resulting gain of PSet$'$ to decrease. For case 2, due to Condition 4, it follows that merging any two adjacent intervals that intersect with P_l and splitting P'_j into $[u, s-1]$ and $[t+1, v]$ does not cause the gain of PSet$'$ to reduce. This procedure can be repeated to get rid of all P'_js in PSet$'$ that overlap with multiple P_ls. At the end of Step 2, each P'_j in PSet$'$ overlaps with at most one P_l. As a result, for every P_l that overlaps with multiple P'_js or every P'_j that does not overlap with any P_l, there exists a P_m that overlaps with no P'_js.

3. Finally, consider the P_ms that do not intersect with any of the P'_js in PSet$'$. We need to consider two possible cases: 1) there is a P'_j that does not intersect with any P_l, or 2) some P_l intersects with multiple P'_js. For case 1, due to Condition 3, it follows that $gain(P_m) \geq gain(P'_j)$, and thus deleting P'_j and adding P_m to PSet$'$ does not cause the overall gain of PSet$'$ to decrease. For case 2, due to Condition 5, it follows that merging any two adjacent intervals P'_j that intersect with P_l and adding P_m to PSet$'$ does not cause PSet$'$'s gain to reduce. This procedure can be repeated until every P_l intersects with exactly one P'_j, thus making them identical. ∎