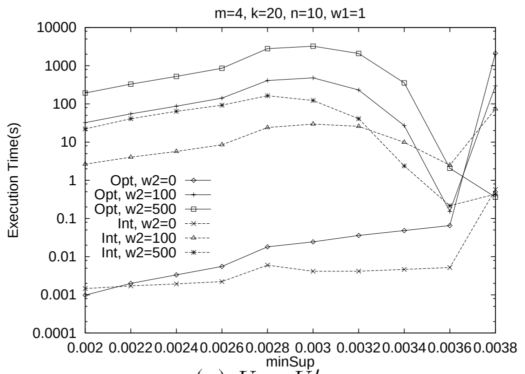
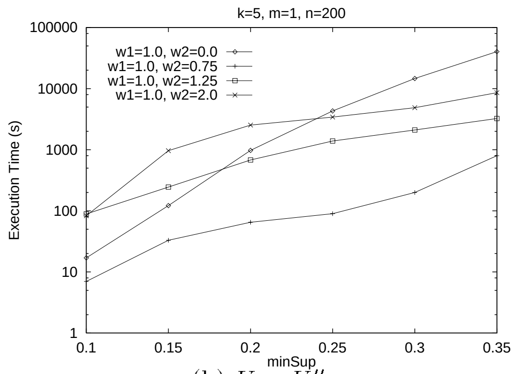


(a) $U = U'$



(b) $U = U''$

Figure 4: Sensitivity to w_1 and w_2 (optimized confidence sets)

paths that cannot result in the optimized rule. For numeric attributes, we developed an algorithm for pruning instantiated rules prior to performing the search for the optimized confidence rule. Finally, we reported the results of our experiments that demonstrate the practicality of the proposed algorithms.

Acknowledgments

We would like to thank Phil Gibbons and Jeff Vitter for their valuable insights that led to Theorem 4.2 and its proof. We would also like to thank Narain Gehani, Hank Korth and Avi Silberschatz for their encouragement and for their comments on earlier drafts of this paper. Without the support of Yesook Shim, it would have been impossible to complete this work.

References

[1] R. Agrawal, T. Imielinski, and A. 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.

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

[3] U. M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy. *Advances in Knowledge Discovery and Data Mining.* AAAI Press/The MIT Press, Menlo Park, CA, 1996.

[4] T. Fukuda, Y. Morimoto, S. Morishita, and T. 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.

[5] T. Fukuda, Y. Morimoto, S. Morishita, and T. Tokuyama. Mining optimized association rules for numeric attributes. In *Proc. of the ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, June 1996.

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

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

[8] G. Piatetsky-Shapiro. Discovery, analysis, and presentation of strong rules. In G. Piatetsky-Shapiro and W. Frawley, editors, *Knowledge Discovery in Databases*, pages 229–248. AAAI/MIT Press, Menlo Park, CA, 1991.

[9] R. Rastogi and K. Shim. Mining optimized association rule for categorical and numeric attributes. Technical Report 0112370-970217-03, Bell Laboratories, Murray Hill, 1997.

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

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

in Section 5 and the detailed algorithm and experimental result can be found in [9]. In our experiment, the best execution time for the algorithm without prior pruning is greater than the best execution time for the prior pruning algorithm. As a result, in subsequent experiments, we only use the prior pruning algorithm.

6.2 Sensitivity to w_1 and w_2

Optimized Confidence Sets: For optimized confidence sets, we fix w_1 at 1 and vary w_2. When $U = U'$, the support of each instantiation is very low (the average support of an instantiation is $\frac{1}{n^m}$). In contrast, when $U = U''$, instantiations with larger rectangles have larger supports – as a matter of fact, the instantiation with the largest rectangle has support 1. Thus, for $U = U'$ (1) only low values for minSup are meaningful, and (2) only large values for w_2 impact the performance of our search algorithms.

We first describe the results for $U = U'$. Figure 4-(a) presents the execution times for both the depth first as well as the graph search algorithms for values of minSup between 0.002 and 0.0038. For most minSup values, both algorithms perform the best when $w_2 = 0$ – that is, when instantiations are sorted by confidence only. When minSup is very high (e.g., 0.00038), the algorithms with w_2 of 500 (that is, the highest value for w_2 that we selected) perform as well or better. The reason for this is that for a majority of the support values, the optimized confidence set comprises mainly of instantiations with high confidences. When instArray is sorted by confidence, these instantiations are considered first by the algorithms, allowing a rapid convergence to the optimized confidence set. As we increase the value of w_2, instantiations with higher confidences are pushed to the end of instArray by instantiations with larger supports (and possibly smaller confidences). Thus, the algorithms need to enumerate more sets before the optimized set is reached, and this makes them perform poorly. Higher values of w_2, however, are better for large minSup values when the optimized confidence set contains instantiations with both high supports and high confidences.

Once instantiations are sorted such that those belonging to the optimized set are toward the front of instArray, then the graph search algorithm that uses intermediate sets for pruning is faster than the depth first algorithm that uses only the current optimized set. One of the reasons for this is that the graph search algorithm prunes the search space more effectively. The other reason is that the graph search algorithm initially concentrates on finding the optimized set using instantiations at the front of instArray and gradually considers instantiations toward the end. In contrast, the depth first search algorithm may have to generate sets containing instantiations located at the end of instArray before all of the instantiations at the front of instArray have been considered.

Now, we turn our attention to $U = U''$. In Figure 4-(b), we plot execution times for the depth first algorithm with prior pruning as minSup is varied from 0.1 to 0.35. For $U = U''$, typically instantiations with large supports have small confidences and the ones with high confidences have low supports. This explains why (1) for all values of w_2, execution times for the algorithm increases with minSup, and (2) for $w_2 = 0.75$, the algorithm performs the best. As minSup is increased, the supports for instantiations that are possible candidates for the optimized set increases. Thus, with increasing minSup, a larger number of instantiations need to be considered by the search algorithms, and their performance degrades.

Furthermore for small values of w_2 (e.g., 0), instArray is sorted by confidence and instantiations with low supports and high confidences are at the front of instArray. These instantiations, however, cannot be in the optimized set if minSup is large – thus, the algorithm performs the worst when $w_2 = 0$ and minSup is high. On the other hand, for large values of w_2 (e.g., 2), instantiations with very high supports and low confidences move up in instArray causing the algorithm perform poorly for smaller values of minSup. The algorithm performs the best for a wide range of minSup values when $w_2 = 0.75$ since this results in instantiations with moderately high values for both supports and confidences making it to the top in instArray.

6.3 Sensitivity to size of domain

For $U = U'$, we found that for a given k, increasing the size of the domain n for attributes actually causes the search times to decrease since there are a lot more instantiations with higher supports and confidences (supports and confidences are uniformly distributed across the instantiations). However, when $U = U''$, even though the number of instantiations with high supports and confidences increases, a disproportionately larger number of instantiations (1) with high confidences have low supports (e.g., points), and (2) with high supports have low confidences (e.g., instantiations with large rectangles). The detailed result can be found in [9].

6.4 Sensitivity to k

Due to the lack of space, we present results in [9].

7 Concluding Remarks

In this paper, we generalized the optimized association rules problem proposed in [5] in three ways – 1) association rules are permitted to contain disjunctions over uninstantiated attributes 2) association rules are allowed to contain an arbitrary number of uninstantiated attributes and 3) uninstantiated attributes can be either categorical or numeric. Since the problem of computing optimized rules is intractable, we had to develop effective mechanisms for both, exploring as well as pruning the search space. We assigned a *weight* to each instantiation, and our search algorithms considered instantiations in the decreasing order of their weights. Thus, based on input parameters (e.g., minimum support, number of disjunctions), by appropriately assigning weights to instantiations, the exploration of the search space can be guided to be efficient. In addition, we proposed a general depth first algorithm that keeps track of the current optimized rule and uses it to prune the search space. For categorical attributes, we also proposed a graph search algorithm that uses intermediate results to eliminate stantiated attributes $A_1, A_2, \ldots, A_m$ in conditions of the form $A_i \in [l_i, u_i]$ and the domain of A_i ranges between 1 and n_i, then the total number of instantiations is $n_1C_2 * n_2C_2 * \cdots * n_mC_2$. In contrast, the number of possible instantiations if every attribute was categorical would be $n_1 \cdot n_2 * \cdots * n_m$. Thus, the number of instantiations to be examined by our search algorithms increases dramatically for optimized association rules with uninstantiated conditions of the form $A_i \in [l_i, u_i]$.

In the remainder of this section, we propose a pruning technique for the optimized confidence problem that reduces the number of instantiations stored in instArray, but still guarantees optimality. Reducing the number of instantiations allows us to lower the overhead of storing (that is, appending) the instantiations to instArray, and to incur smaller costs when sorting the instantiations in the decreasing order of their weights. More importantly, it can also considerably reduce the input size to our search algorithms.

Key Observation: As we mentioned earlier, the number of instantiations for numeric attributes increases significantly as the number of uninstantiated attributes increases. An instantiation for m numeric attributes can be represented as $[(x_1, x_2, \ldots, x_m), (y_1, y_2, \ldots, y_m)]$, where the interval for the uninstantiated numeric attribute A_i is bounded by x_i below and y_i above. Thus, the x_i, y_i pairs determine the bounds for the rectangle for the instantiation along each of the m axis. For instantiations $I_1 = [(x_1, x_2, \ldots, x_m), (y_1, y_2, \ldots, y_m)]$ and $I_2 = [(u_1, u_2, \ldots, u_m), (v_1, v_2, \ldots, v_m)]$, we say that I_1 is contained in I_2 if $u_i \leq x_i \leq y_i \leq v_i$ for all i (that is, the rectangle for I_1 is contained in the rectangle for I_2).

The following theorem provides the basis for pruning instantiations in instArray when computing optimized confidence sets.

Theorem 5.1: *Let I_1 and I_2 be two instantiations such that I_1 is contained in I_2. If both I_1 and I_2 have support at least* minSup *and $conf(I_1) \geq conf(I_2)$, then there exists an optimized confidence set that does not contain I_2.* ∎

Instantiation I_2 can be deleted since I_2's rectangle is the outer rectangle. Thus, if the optimized set were to contain I_2, then replacing I_2 with I_1 results in an optimized set with at least the same confidence and no overlapping instantiations. However, the above pruning rule does not work for optimized support sets since that would require I_1 to be pruned (when I_1's rectangle is contained in I_2's rectangle and $conf(I_2) \geq conf(I_1) \geq$ minConf). Thus, if the optimized set were to contain I_1, then replacing I_1 with I_2 results in an optimized set with as high confidence and support – however, it could contain overlapping instantiations.

Due to lack of space, we do not report in this paper, the details of the algorithm that uses the above pruning rule for pruning instantiations from instArray before optConfPruneOpt is executed. The algorithm, including an analysis of its time and space complexity, can be found in [9]. In [9], we show that the time complexity of the algorithm is $O(m * n_1^2 * \cdots * n_m^2)$, where m is the number of numeric attributes considered and n_i is the number of values for attribute A_i.

6 Experimental Results

In this section, we study the performance of our algorithms for computing optimized confidence and support sets. From our experimental results, we establish that

- Our pruning techniques improve the performance of both, depth first and graph search algorithms, significantly.
- The values for w_1 and w_2 play an important role in reducing the search times of our algorithms.
- Pruning instantiations prior to performing search, whenever possible, can result in further improvements in performance.
- The low execution times of our algorithms in a large number of cases make them suitable to be used in practice.

Since naive algorithms that exhaustively enumerate all possible sets are obviously too expensive, we do not consider them. Also, the data file is read only once at the beginning of each algorithm (in order to generate instantiations). 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 number of attributes and the sizes of their domains.

We performed extensive experiments using a Sun Ultra-2/200 machine with 512 MB of RAM and running Solaris 2.5. However, due to lack of space, we do not report all our experimental results – these can be found in [9].

Synthetic Datasets: The association rule that we experimented with, has the form $U \wedge C_1 \rightarrow C_2$ where U contains m uninstantiated attributes (see Section 3). For simplicity, we assume that the domains of the uninstantiated attributes consist of integers ranging from 1 to n. We consider two forms for U. The first, U', has the form $A_1 = v_1 \wedge \ldots \wedge A_m = v_m$, and can be used for both categorical and numeric attributes. The second, U'', has the form $A_1 \in [l_1, u_1] \wedge \ldots \wedge A_m \in [l_m, u_m]$, and can be used only for numeric attributes. Every instantiation of $U' \wedge C_1 \rightarrow C_2$ (and thus, every point in m-dimensional space) is assigned a randomly generated confidence between 0 and 1 with uniform distribution. Each instantiation of $U' \wedge C_1 \rightarrow C_2$ is also assigned a randomly generated support between 0 and $\frac{2}{n^m}$ with uniform distribution.

6.1 Pruning Prior to Search

We begin by studying the effectiveness of pruning instantiations from instArray prior to performing search. This technique applies only to optimized confidence sets when U has the form U'' – it was introduced

```
procedure optConfPruneInt(intList, curLoc):
1.  newList := $\emptyset$
2.  foreach $S_1 = \emptyset$ and $S_1 \in$ intList do {
3.      $S_2$:=$S_1 \cup$ {instArray[curLoc]}
4.      Append $S_2$ to the end of newList
5.      if $sup(S_2) \geq$ minSup and $conf(S_2) > conf$(optSet)
6.          optSet := $S_2$
7.  }
8.  if curLoc = $n$
9.      return
10. Append elements in newList to intList
11. foreach $S_1 \in$ intList (from head to tail) do{
12.     [minS, maxS] := optConfRange($S_1$, curLoc+1)
13.     if $numInst(S_1) < k$ and minS $\leq$ maxS
14.         foreach $S_2$ preceding $S_1 \in$ intList do {
15.             if optConfCanPrune($S_2$,$S_1$,curLoc+1) = true {
16.                 Delete $S_1$ from intList
17.                 break out of inner for loop
18.             }
19.             else if optConfCanPrune($S_1$,$S_2$,curLoc+1) = true
20.                 Delete $S_2$ from intList
21.         }
22.     else
23.         Delete $S_1$ from intList
24. }
25. minWeight := $w_1$*$conf$(optSet) + $\frac{w_2 * \text{minSup}}{k}$
26. if intList $\neq \emptyset$ or $w$(instArray[curLoc+1]) $\geq$ minWeight
27.     optConfPruneInt(intList, curLoc+1)
```

Figure 3: Graph search alg. for optimized confidence set

solving the quadratic inequality equation. Due to lack of space, the procedure optConfCanPrune to decide whether an intermediate set S_1 can prune another intermediate set S_2 with the next instantiation to extend both intermediate sets is presented in [9].

The overall procedure for computing optimized confidence sets is described in Figure 3. The procedure accepts as input parameters intList which is a list of intermediate sets for the first curLoc-1 instantiations in instArray, and curLoc which is the index of the instantiation (in instArray) to be considered next for extending the intermediate sets in intList. The procedure is initially invoked with arguments intList=$\emptyset$ and curLoc=1, and it recursively invokes itself with successively increasing values for curLoc.

The procedure begins by extending every set in intList with instArray[curLoc] and forms a new list newList containing {instArray[curLoc]} and the extended sets (steps 1–7). Furthermore, if an extended set in newList is found to have support at least minSup and higher confidence than the currently stored optimized set, then optSet is set to the extended set. In steps 10–24, those intermediate sets that cannot be extended further to result in the optimized set are deleted from intList. These include sets already containing k instantiations and sets S for which optConfRange(S, curLoc+1) returns an empty range. In addition, for any two intermediate sets S_1 and S_2 in intList that can be extended to result in an optimized set, if one can prune the other, then the other is deleted (steps 14–21). Finally, steps 25–27 contain conditions that enable the algorithm to terminate early – even though curLoc may be less than n. This occurs when there are no intermediate sets to expand (that is, intList = $\emptyset$) and the weight of instArray[curLoc+1] is less than minWeight, the minimum weight required to generate the optimized set (follows from Theorem 4.2).

5 Numeric Attributes

In this section, we present algorithms for computing optimized sets when association rules contain uninstantiated conditions of the form $A_i \in [l_i, u_i]$ and A_i is a numeric attribute. Thus, unlike the previous section, in which an instantiation was obtained by instantiating each uninstantiated attribute with a single value in its domain, in this section each uninstantiated attribute is instantiated with an interval in its domain. Thus, each instantiation corresponds to a rectangle in m-dimensional space (in the previous section, each instantiation corresponded to a point in m-dimensional space).

Permitting association rules to contain uninstantiated numeric attributes in conditions of the form $A_i \in [l_i, u_i]$ complicates the problem of computing optimized sets in several ways. First, unlike the categorical attribute case in which two distinct instantiations had no overlap (since each instantiation corresponded to a point), two distinct instantiations may overlap. This makes it impossible to use intermediate sets for pruning as described in Section 4.4. For instance, when computing optimized confidence sets, consider two intermediate sets S_1 and S_2 such that for every set S of instantiations that can be used to extend S_2 (to result in a set with confidence at least that of optSet and support at least minSup), $sup(S_1 \cup S) \geq$ minSup and $conf(S_1 \cup S) \geq conf(S_2 \cup S)$. Even though $conf(S \cup S_1) \geq conf(S \cup S_2)$, $S \cup S_1$ may not qualify to be the optimized set since instantiations in S may overlap with instantiations in S_1. However, there may be no overlap between S and S_2, and thus it may still be possible to extend S_2 to result in the optimized set. As a result, S_1 cannot prune S_2. This is not a problem for categorical attributes since there is no overlap between any pair of instantiations.

The depth first algorithm in Section 4.3, however, can be modified to compute optimized confidence sets. Since the optimized set can contain only non-overlapping instantiations, we must not extend curSet with an overlapping instantiation instArray[i] – this cannot result in the optimized set. For a set S of instantiations and an instantiation I, let the function $overlap(S, I)$ return **true** if the rectangle for some instantiation in S and the rectangle for I overlap. Then the modification to optConfPruneOpt is to have the statement "**if** $overlap$(curSet, instArray[i]) = **false**" between steps 1 and 2, and the body of this if-statement includes the steps 2–8.

The next complication is that if there are m unin-

Optimized Support Set: For optimized support sets, constraints on s are similar to constraints (1)–(7) except that $conf$(optSet) and minSup are replaced with minConf and sup(optSet), respectively. In other words, the confidence of curSet $\cup S$ must be at least minConf and the support of curSet $\cup S$ should be no smaller than that of optSet. Thus, optSupRange, the procedure to compute the range of values for s that satisfy constraints (1)–(7), is similar to optConfRange with all occurrences of $conf$(optSet) and minSup replaced with minConf and sup(optSet). Also, optSupPruneOpt, the procedure for computing optimized sets invokes optSupRange instead. Note that if minConf = maxConf, then optSupRange always returns [1,0] for convenience. However, this is a special case for which the optimized support set can be computed directly and is simply the set of all instantiations with confidence maxConf.

4.4 Pruning using Intermediate Sets

The pruning technique presented in the previous subsection used the current optimized set to reduce the search space of the depth-first algorithm for computing optimized sets. A different class of algorithms, *graph search* algorithms, maintain a list of intermediate sets, and in each step, extend one of them. In this case, not only the current optimized set but also the intermediate sets can be used for pruning. The graph search algorithms, however, do incur additional storage overhead since they have to keep track of intermediate sets. In this section, we present new techniques for pruning using intermediate sets.

Optimized Confidence Set: The algorithm, after the p^{th} step, keeps track of, for the p instantiations with the highest weight, (1) the current optimized set, and (2) intermediate sets involving some subset of the p instantiations. The intermediate sets are those that have the potential to be further extended with the remaining instantiations to yield the optimized set. In the $p+1^{th}$ step, it extends every intermediate set using the instantiation with the next highest weight, thus generating new intermediate sets. If any of the new intermediate sets is better than the current optimized set, then optSet is replaced. Next, since instantiations in instArray are stored in the decreasing order of their weights, procedure optConfRange from the previous subsection (that used the current optimized set for pruning) is used to eliminate those intermediate sets that can never be candidates for the optimized set.

Furthermore, consider two intermediate sets S_1 and S_2 for which $numInst(S_1) \leq numInst(S_2)$ and for every set S of instantiations that can be used to extend S_2 (to result in a set with confidence at least that of optSet and support at least minSup), $sup(S_1 \cup S) \geq$ minSup and $conf(S_1 \cup S) \geq conf(S_2 \cup S)$. Due to $numInst(S_1) \leq numInst(S_2)$, it is the case that $S_1 \cup S$ contains no more instantiations than $S_2 \cup S$. Thus, since there is no overlap between any two arbitrary instantiations, for some set S if $S_2 \cup S$ could be an optimized confidence set, then $S_1 \cup S$ is also an optimized set. Consequently, it follows that deleting S_2 from the list of intermediate sets does not affect the ability of our algorithm to discover the optimized confidence set.

The range of supports for a set S that can be used to extend S_2 to result in a set with support minSup and confidence as good as the current optimized set can be obtained using the procedure optConfRange. Furthermore, if s and c are the support and confidence of set S, respectively, $l = k - numInst(S_2)$, and i is the index of the next instantiation to be considered for extending the intermediate sets, then due to constraints (1) and (5) from the previous section, the confidence of S satisfies the following inequality.

$$conf(\text{optSet}) + \frac{(conf(\text{optSet}) - conf(S_2)) * sup(S_2)}{s} \leq c \leq \frac{w(\text{instArray}[i]) * l - w_2 * s}{l * w_1} \quad (8)$$

Suppose, in addition, we compute the range of supports for a set S which when used to extend S_1 causes its support to be at least minSup and for all values of c satisfying Constraint (8), the confidence of $S_1 \cup S$ is at least that of $S_2 \cup S$. Now, if the former range of supports for set S is contained within the latter, then it implies that for every set S (with support s and confidence c satisfying Constraint (8) above) that can be used to extend S_2 to yield an optimized set, the same set S can also be used to extend S_1 resulting in a set with support at least minSup and confidence greater than or equal to that of $S \cup S_2$. Thus, S_2 can be pruned. The challenge is to determine the latter range, which we compute below, as follows. We required the support of $S_1 \cup S$ to be at least minSup – this translates to the following constraint on s.

$$\text{minSup} \leq sup(S_1) + s \quad (9)$$

In addition, we required that for all values of c satisfying Constraint (8), $conf(S_1 \cup S) \geq conf(S_2 \cup S)$. Thus s must satisfy the following constraint for all values of c satisfying Constraint (8).

$$\frac{sup(S_1) * conf(S_1) + s * c}{sup(S_1) + s} \geq \frac{sup(S_2) * conf(S_2) + s * c}{sup(S_2) + s} \quad (10)$$

It can be shown that if $sup(S_1) \leq sup(S_2)$, then for a given value of s, the above constraint holds for all values of c (satisfying Constraint (8)) if it holds for s and $c = c_{min}$, the leftmost term in Constraint (8). Similarly, if $sup(S_1) > sup(S_2)$, then for a given value of s, Constraint (10) holds for all values of c (satisfying Constraint (8)) if it holds for s and $c = c_{max}$, the rightmost term in Constraint (8). Thus, the range of values for s that we are interested in are those that satisfy Constraint (9), and either Constraint (10) with c replaced with c_{min} if $sup(S_1) \leq sup(S_2)$ or Constraint (10) with $c = c_{max}$ otherwise.

Constraint (10) with $c = c_{min}$ and $c = c_{max}$ results in two separate equations of the form $A*s^2+B*s+C \leq 0$ with values of A, B and C. The range of values for s that satisfy the equations can be determined by set S that could be used to extend curSet such that conditions (1) and (2) hold. Due to the condition (2), we have the following constraint.

$$\frac{c * s + conf(\text{curSet}) * sup(\text{curSet})}{s + sup(\text{curSet})} \geq conf(optSet)$$

Re-arranging the terms yields the following constraint.

$$\begin{aligned} c \quad &\geq conf(\text{optSet}) + \\ &\frac{(conf(\text{optSet}) - conf(\text{curSet})) * sup(\text{curSet})}{s} \end{aligned} \quad (1)$$

Also, since the set curSet$\cup S$ must satisfy minimum support, and a set S can consist of at most l instantiations we require s to satisfy the following two constraints.

$$\mathsf{minSup} \leq s + sup(\text{curSet}) \leq 1 \quad (2)$$

$$0 \leq s \leq l * \mathsf{maxSup} \quad (3)$$

Note that since the confidence of set S can be at most maxConf, we have the following constraint: $c \leq \mathsf{maxConf}$. Combining this constraint with Constraint (1) results in the following constraint which, if satisfied by s, ensures that there exists a c that does not exceed maxConf and at the same time, causes the confidence of curSet$\cup S$ to be at least $conf$(optSet).

$$\frac{sup(\text{curSet}) * (conf(\text{optSet}) - conf(\text{curSet}))}{\mathsf{maxConf} - conf(\text{optSet})} \leq s \quad (4)$$

Finally, we exploit the fact that instantiations are sorted in the decreasing order of their weights and only instantiations that follow instArray[i] are used to extend curSet. We utilize the following property of sets of instantiations.

Theorem 4.2: *For an arbitrary set of instantiations S, there exists an instantiation $I \in S$ such that*

$$w(I) \geq w_1 * conf(S) + \frac{w_2 * sup(S)}{numInst(S)}$$

∎

Thus, since S contains only instantiations with weights at most w(instArray[i]), and S can contain at most l instantiations, due to Theorem 4.2, we obtain the following constraint on s and c.

$$w_1 * c + \frac{w_2 * s}{l} \leq w(\text{instArray}[i])$$

Re-arranging the terms, we get

$$c \leq \frac{w(\text{instArray}[i]) * l - w_2 * s}{l * w_1} \quad (5)$$

Since c, the confidence of S, must be at least 0, substituting 0 for c in the above equation results in the following constraint that prevents s from getting so large that c would have to become negative to satisfy the above constraint.

$$s \leq \frac{w(\text{instArray}[i]) * l}{w_2} \quad (6)$$

Finally, constraints (1) and (5) can be combined to limit s to values for which a corresponding value of c can be determined such that the confidence of curSet$\cup S$ does not drop below $conf$(optSet), and the weights of instantiations in S do not exceed w(instArray[i]).

$$\begin{aligned} &\frac{w(\text{instArray}[i]) * l - w_2 * s}{w_1 * l} \geq conf(\text{optSet}) + \\ &\frac{(conf(\text{optSet}) - conf(\text{curSet})) * sup(\text{curSet})}{s} \end{aligned} \quad (7)$$

Any set S used to extend curSet to produce a better set for optimized confidence must satisfy the constraints (2)–(7) (on its support). Thus, if there exists a value of s satisfying constraints (2)–(7), then curSet must be extended. Otherwise, it can be pruned without extending it further.

The procedure for computing the optimized confidence set, illustrated in Figure 2, is similar to the naive algorithm except that it invokes optConfRange to determine if there exists an S that curSet can be extended with to yield the optimized set. It stops extending curSet if the range of values [minS, maxS] for the support of such a set is empty (which happens when minS > maxS).

```
procedure optConfPruneOpt(curSet, curLoc):
1.  for $i$ := curLoc to $n$ do {
2.      [minS,maxS] := optConfRange(curSet, $i$)
3.      if minS > maxS
4.          break
5.      S := curSet ∪ {instArray[$i$]}
6.      if $sup$(S) ≥ minSup and $conf$(S) > $conf$(optSet)
7.          optSet := S
8.      if $numInst$(S) < $k$
9.          optConfPruneOpt(S, $i + 1$)
10. }
```

Figure 2: Depth first alg. for optimized confidence set

The procedure optConfRange computes the range [minS, maxS] which is the range of values for s that satisfy the above constraints. If there does not exist an s that satisfies constraints (2)–(7), then a range for which minS > maxS is returned for convenience. Thus, if minS > maxS for the returned range from optConfRange, we can stop extending curSet. The procedure takes as input curSet and the index i (in instArray) of the next instantiation being considered to extend curSet. A detailed derivation of the ranges satisfying the above constraints and a description of the procedure optConfRange can be found in [9].

```
procedure optConfNaive(curSet, curLoc):
1. for $i$ := curLoc to $n$ do {
2.     S := curSet ∪ {instArray[i]}
3.     if $sup$(S) $\geq$ minSup and $conf$(S) $>$ $conf$(optSet)
4.         optSet := S
5.     if $numInst$(S) $<$ $k$
6.         optConfNaive(S, $i+1$)
7. }
```

Figure 1: Naive Alg. for Optimized Confidence Set

4.2 Naive Algorithm

Optimized Confidence Set: We begin by presenting a naive algorithm for determining the optimized confidence set of instantiations (see Figure 1). In a nutshell, the algorithm employs *depth first search* to enumerate all possible sets containing k or less instantiations, and returns the set with the maximum confidence and support at least minSup.

The algorithm assumes that instantiations are stored in the array instArray. The number of instantiations is $n = n_1 * n_2 * \cdots * n_m$, where n_i is the number of values in the domain of A_i. The algorithm is initially invoked with arguments curSet = $\emptyset$ and curLoc=1. The variable optSet is used to keep track of the optimized set of instantiations encountered during the execution of the algorithm. The algorithm enumerates all possible subsets of size k or less by recursively invoking itself (Step 6) and sets optSet to a set with a greater confidence than the current optimized set (steps 3 and 4). Each invocation accepts as input curSet, the set of instantiations to be further extended, and curLoc, the index of the first instantiation in instArray to be considered for extending curSet (all instantiations between curLoc and n are considered). The extended state is stored in S, and if the number of instantiations in S is less than k, the algorithm calls itself recursively to further extend S with instantiations whose index is greater than the index of all the instantiations in S.

The complexity of the naive algorithm is $\Sigma_{i=1}^{k} nC_i$. When $k \ll n$, the complexity of the algorithm becomes $O(n^k)$. However, as we showed earlier, if there is no restriction on the size of the optimized set (that is, $k = n$), the problem is NP-hard.

Optimized Support Set: The naive algorithm for computing the optimized support set is similar to optConfNaive, except that the condition in Step 3 which tries to maximize confidence is replaced with the following condition: **if** $conf$(S) $\geq$ minConf **and** sup(S) $>$ sup(optSet).

4.3 Pruning Using the Current Optimized Set

The naive algorithm exhaustively enumerates all possible sets with at most k instantiations – this results in high complexity. However, in the naive algorithm illustrated in Figure 1, if we know that the confidence of any set satisfying minimum support and obtained as a result of extending curSet cannot exceed the confidence of the current optimized set (that is, optSet), then we can stop extending curSet immediately and reduce the search space significantly. In this section, we develop *branch and bound* pruning techniques that, with the aid of the current optimized set, considerably reduce the overhead of exploring the entire search space.

For our pruning techniques to be effective, it is imperative that we find a set close to the optimized set early – since it can then be used to eliminate a larger number of sub-optimal sets. It may seem logical that for the optimized confidence problem, since we are trying to maximize confidence, considering instantiations with high confidences first may cause the search to converge on the optimized set more rapidly. However, this may not be the case since the support of the optimized confidence set has to be at least minSup. For a high minimum support, it may be better to explore instantiations in the decreasing order of their supports. Thus, the order in which instantiations must be considered by the search algorithm is a non-trivial problem. In order to investigate this idea of pruning curSet early, we introduce the notion of the weight of an instantiation I (denoted by $w(I)$) and define it below.

$$w(I) = w_1 * conf(I) + w_2 * sup(I)$$

In the definition, w_1 and w_2 are positive real constants. Thus, the weight of an instantiation is the weighted sum of both, its confidence and support. Our search algorithms can then consider instantiations with higher weights first, and by using different values of w_1 and w_2, vary the strategy for enumerating sets. In the remainder of this section, we propose algorithms that store instantiations in instArray in the decreasing order of their weights and explore instantiations with higher weights first. We also present techniques that exploit the sort order of instantiations to prune the search space. The variables maxConf and maxSup are used to store the maximum confidence and maximum support of all the instantiations in instArray, respectively.

Optimized Confidence Set: Suppose the current set of instantiations, curSet, is only extended with instantiations belonging to the set comprising of instArray[i], for some i, and instantiations following it in instArray. The key idea is that we can stop extending curSet if among instantiations being considered to extend curSet, there does not exist a set of instantiations S such that (1) $sup(\text{curSet} \cup S) \geq$ minSup, and (2) $conf(\text{curSet} \cup S) \geq conf(\text{optSet})$.

In order to determine whether a set S satisfying the above conditions (1) and (2) exists, we first derive the constraints that such a set S must satisfy. If the constraints are unsatisfiable with the remaining instantiations that are candidates for extension, a set S satisfying the conditions (1) and (2) does not exist and we can stop extending curSet immediately. Let variables s and c denote $sup(S)$ and $conf(S)$, respectively, and $l = k - numInst(\text{curSet})$. In the following, we derive the constraints on s that must be satisfied by any

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 [1] where $sup(R)$ was defined to be $sup(C_1 \wedge C_2)$. Instead, we have adopted the definition used in [5] and [4]. Also, let minSup and minConf denote the user-specified minimum support and minimum confidence, respectively.

The optimized association rule problem requires optimal instantiations to be computed for an uninstantiated association rule which has the form: $U \wedge C_1 \rightarrow C_2$, where U is a conjunction of m uninstantiated atomic conditions over m distinct attributes, and C_1 and C_2 are arbitrary instantiated conditions. Let U_i denote an *instantiation* of U – thus, U_i is obtained by replacing variables in U with values. An instantiation U_i can be mapped to a rectangle in m-dimensional space – there is a dimension for each attribute and the co-ordinates for the rectangle in a dimension are identical to the values for the corresponding attribute in U_i. Two instantiations U_1 and U_2 are said to be *non-overlapping* if the two (m-dimensional) rectangles defined by them do not overlap (that is, the intersection of the two rectangles is empty).

Having defined the above notation for association rules, we present below, the formulations of the optimized association rule problems.

- **Optimized Confidence Problem:** Given k and an uninstantiated rule $U \wedge C_1 \rightarrow C_2$, determine non-overlapping instantiations $U_1, \ldots, U_l$ of U with $l \leq k$ such that $sup(R) \geq$ minSup and $conf(R)$ is maximized, where R is the rule $(U_1 \vee \cdots \vee U_l) \wedge C_1 \rightarrow C_2$.
- **Optimized Support Problem:** Given k and an uninstantiated rule $U \wedge C_1 \rightarrow C_2$, determine non-overlapping instantiations $U_1, \ldots, U_l$ of U with $l \leq k$ such that $conf(R) \geq$ minConf and $sup(R)$ is maximized, where R is the rule $(U_1 \vee \cdots \vee U_l) \wedge C_1 \rightarrow C_2$.

The problem of computing optimized association rules required instantiations $U_1, \ldots, U_l$ to be determined such that the rule $R : (U_1 \vee \cdots \vee U_l) \wedge C_1 \rightarrow C_2$ satisfies user-specified constraints. Suppose for an instantiation U_i of U, I_i is the instantiated rule $U_i \wedge C_1 \rightarrow C_2$, and for a set $S = \{I_1, \ldots, I_j\}$ of instantiated rules, $sup(S)$ and $conf(S)$ are defined as follows.

$$\begin{aligned} sup(S) &= sup(I_1) + \cdots + sup(I_j) \\ conf(S) &= \frac{sup(I_1) \cdot conf(I_1) + \cdots + sup(I_j) \cdot conf(I_j)}{sup(I_1) + \cdots + sup(I_j)} \end{aligned}$$

Then, we have $sup(S) = sup(R)$ and $conf(S) = conf(R)$. Thus, since (1) for every instantiated rule (or alternatively, instantiation) I_i, $sup(I_i)$ and $conf(I_i)$ can be computed by performing a single pass over the relation, and (2) these, in turn, can be used to compute supports and confidences for sets of instantiations, the optimized association rule problem reduces to the following.

- **Optimized Confidence Problem:** Given k, and $sup(I_i)$ and $conf(I_i)$ for every instantiation I_i, determine a set S containing at most k non-overlapping instantiations such that $sup(S) \geq$minSup and $conf(S)$ is maximized.
- **Optimized Support Problem:** Given k, and $sup(I_i)$ and $conf(I_i)$ for every instantiation I_i, determine a set S containing at most k non-overlapping instantiations such that $conf(S) \geq$ minConf and $sup(S)$ is maximized.

In the remainder of the paper, we use the above formulations instead to develop algorithms for the optimized association rule problems.

4 Categorical Attributes

In this section, we present algorithms for computing optimized support and confidence sets when rules contain only uninstantiated conditions of the form $A_i = v$. Thus, the results of this section are only applicable to categorical attributes and numeric attributes restricted to $A_i = v$[1] (conditions of the form $A_i \in [l_i, u_i]$, where A_i is a numeric attribute, are dealt with in the next section). An example of such a rule is (date $= v$) $\wedge$ src_city = NY $\rightarrow$ dst_country = France (in the rule, date is a numeric attribute while src_city is categorical).

Due to the above restriction, any two arbitrary instantiations are always non-overlapping. This property is essential for the correctness of the pruning technique used by the graph search algorithm presented in Section 4.4.

4.1 NP-Hardness Result

The problem of computing optimized sets, given $sup(I_i)$ and $conf(I_i)$ for every instantiation I_i, can be shown to be intractable, and follows from the following theorem.

Theorem 4.1: *Given $sup(I_i)$ and $conf(I_i)$ for every instantiation I_i, determining if there is a set S containing an arbitrary number of instantiations such that $conf(S) \geq$ minConf and $sup(S) \geq$ minSup is NP-hard.* ∎

In the following subsections, we present schemes for computing optimized sets that employ techniques for pruning the search space in order to overcome the complexity of the problem. In each subsection, we first present the scheme for computing optimized confidence sets, and then briefly describe the modifications to the scheme in order to compute optimized support sets.

[1]Note that if the domain of the numeric attribute A_i is large, then it can be partitioned into a sequence of n_i intervals, and successive intervals can be mapped to consecutive integers in the interval between 1 and n_i.

a minimum support of 0.03, the optimized confidence rule results in the period for which calls from NY in the period are at least 3% of the total number of calls, and the percentage of calls from NY that are directed to France is maximum. With a minimum confidence of 0.5, the optimized support rule results in the period during which at least 50% of the calls from NY are to France, and the number of calls originating in NY is maximum.

A limitation of the optimized association rules dealt with in [5] 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 sizable number of calls from NY to France are made. 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: $(\mathsf{date} \in [l_1, u_1]) \vee \cdots \vee (\mathsf{date} \in [l_k, u_k]) \wedge \mathsf{src\_city} = \mathsf{NY} \rightarrow \mathsf{dst\_country} = \mathsf{France}$. The above framework can be further strengthened by enriching association rules to contain more than one uninstantiated attribute, and permitting attributes to be both numeric (e.g., date and duration) as well as categorical (e.g., src_city, dst_country). Thus, optimal instantiations for the rule $(\mathsf{src\_city} = v_1 \wedge \mathsf{date} \in [l_1, u_1]) \vee \cdots \vee (\mathsf{src\_city} = v_k \wedge \mathsf{date} \in [l_k, u_k]) \rightarrow \mathsf{dst\_country} = \mathsf{France}$ would yield valuable information about cities and periods with a fairly high outward call volume, a substantial portion of which is directed to France. Alternately, information about cities and *specific* dates can be obtained from the rule $(\mathsf{src\_city} = v_1 \wedge \mathsf{date} = u_1) \vee \cdots \vee (\mathsf{src\_city} = v_k \wedge \mathsf{date} = u_k) \rightarrow \mathsf{dst\_country} = \mathsf{France}$. This information can be used by the telecom service provider to determine the most suitable geographical regions and dates for offering discounts on international long distance calls to France.

In this paper, we generalize the optimized association rules problem, described in [5], in three ways – 1) association rules are permitted to contain disjunctions over uninstantiated attributes, 2) association rules are allowed to contain an arbitrary number of uninstantiated attributes, and 3) uninstantiated attributes can be either categorical or numeric. We first show that the problem of computing optimized support and optimized confidence association rules in our framework is NP-hard. We then present a general *depth first search* algorithm for exploring the search space. The algorithm searches through the space of instantiated rules in the decreasing order of the weighted sums of their confidences and supports, and uses *branch and bound* techniques to prune the search space effectively. For categorical attributes, we also present a *graph search* algorithm that, in addition, uses intermediate results to reduce the search. Finally, for numeric attributes, we develop techniques to eliminate certain instantiated rules prior to searching through them. Experimental results indicate that our schemes perform well for a large number of uninstantiated attributes, disjunctions and values in the domain of the uninstantiated attributes. Proofs of theorems presented in the paper can be found in [9].

2 Related Work

Association rules for a set of transactions in which each transaction is a set of items bought by a customer, were first studied in [1]. These association rules for sales transaction data have the form $X \rightarrow Y$, where X and Y are disjoint sets of items. Efficient algorithms for computing them can be found in [2, 7, 6, 10, 11, 3]. In [10, 6], the generalization of association rules to multiple levels of taxonomies over items is studied. Association rules containing *quantitative* and *categorical* attributes are studied in [8] and [11]. The work in [8] restricts association rules to be of the form $A_1 = v_1 \rightarrow A_2 = v_2$ only. They suggest ways to extend their framework to have a range (that is, $A_1 \in [l_1, u_1]$) rather than a single value in the left hand side of a rule. To achieve this, they partition numeric attributes into intervals. However, they do not consider merging neighboring intervals to generate a larger interval. In [11], the authors use a *partial completeness* measure in order to determine the partitioning of numeric attributes into intervals.

The optimized association rule problem was introduced in [5]. The authors permit association rules to contain a single uninstantiated condition $A_1 \in [l_1, u_1]$ on the left hand side, and propose schemes to determine values for variables l_1 and u_1 such that the confidence or support of the rule is maximized. In [4], the authors extend the results in [5] to the case in which rules contain two uninstantiated numeric attributes on the left hand side. They propose algorithms that discover optimized gain, support and confidence association rules for two classes of regions – rectangles and admissible regions (for admissible regions, the algorithms compute approximate, not optimized, support and confidence rules). However, their schemes only compute a single optimal region. In contrast, our algorithms are general enough to handle more than two uninstantiated attributes, which could be either categorical or numeric. Furthermore, our algorithms can generate an optimal set of rectangles rather than just a single optimal rectangle (note that we do not consider admissible regions or the notion of gain in this paper). This enables us to find more interesting patterns.

3 Preliminaries

In this section, we define the optimized association rule 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 $A_i = v_i$, if v_i is a value from the domain of A_i, the condition is referred to as *instantiated*; else, if v_i is a variable, we refer to the condition as *uninstantiated*. Likewise, the condition $A_i \in [l_i, u_i]$ is referred to as instantiated or uninstantiated depending on whether l_i and u_i are values or variables.

Mining Optimized Association Rules with Categorical and Numeric Attributes

Rajeev Rastogi

Bell Laboratories
Murray Hill, NJ 07974
rastogi@research.bell-labs.com

Kyuseok Shim

Bell Laboratories
Murray Hill, NJ 07974
shim@research.bell-labs.com

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 or confidence of the rule is maximized.*

In this paper, we generalize the optimized association rules problem in three ways – 1) association rules are allowed to contain disjunctions over uninstantiated attributes, 2) association rules are permitted to contain an arbitrary number of uninstantiated attributes, and 3) uninstantiated attributes can be either categorical or numeric. Our generalized association rules enable us to extract more useful information about seasonal and local patterns involving multiple attributes. We present effective techniques for pruning the search space when computing optimized association rules for both categorical and numeric attributes. Finally, we report the results of our experiments that indicate that our pruning algorithms are efficient for a large number of uninstantiated attributes, disjunctions and values in the domain of the attributes.

1 Introduction

Association rules, introduced in [1], provide a useful mechanism for discovering correlations among the underlying data. 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 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 schemes for this can be found in [1, 2, 6, 10, 11].

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.

The *optimized association rules* problem, motivated by applications in marketing and advertising, was introduced in [5]. 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). The authors 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 support of the condition $(A_1 \in [l_1, u_1]) \wedge C_1$ is at least the user-specified 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 confidence (referred to as the *optimized support* rule).

Optimized association rules are useful for unraveling ranges for numeric attributes where certain trends or correlations are strong (that is, have high support or confidence). 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 at least a certain minimum number of calls from NY are made and the percentage of calls from NY to France is maximum. The framework developed in [5] 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