# Monotone Constraints in Frequent Tree Mining

**Jeroen De Knijf**[†] JKNIJF@CS.UU.NL
**Ad Feelders** AD@CS.UU.NL

Utrecht University,
Institute of Information and Computing Sciences,
PO Box 80.089,
3508 TB Utrecht.

## Abstract

Recent studies show that using constraints that can be pushed into the mining process, substantially improves the performance of frequent pattern mining algorithms. However the constraints and algorithms have not yet been explored for frequent structure mining. In this paper we present monotone constraints for trees and develop an opportunistic pruning algorithm that mines frequent trees with monotone constraints. We illustrate the use of these constraints within the application of web log mining. Finally, the effect of applying these constraints on synthetic data sets is evaluated. The opportunistic pruning methods leads to a considerable speedup and a reduction in the number of candidates generated compared to the basic algorithm.

## 1. Introduction

In frequent itemset mining the use of constraints is well explored(Ng et al., 1998; Bayardo, 1998; Bayardo et al., 1999; Pei & Han, 2000). Constraints that can be pushed into the mining process — as opposed to post pruning — allow the mining algorithm to reduce the search space and hence result in a substantial efficiency gain. Furthermore, constraints provide the end user with a tool to focus his interest such that many uninteresting rules don't have to be examined. Since the number of rules can be exponential in the size of the input data, these tools are necessary in practical data mining applications.

When structured data is taken into account several algorithms (Asai et al., 2002; Zaki, 2002; Nijssen & Kok, 2003) are available — based upon the same principle as the Apriori (Agrawal & Srikant, 1994) — that mine for frequent structures in the database. A structure may be a sequence, tree or graph, but in this paper we limit the discussion to trees. In general, structured data can be seen as an extension of unstructured data. Structured data arise extensively in domains such as computational biology, XML databases, and molecule databases. Since the structure of an item encodes an important part of its semantics, mining for structures is an important extension of frequent itemset mining.

Because of the advantages of constraints in frequent itemset mining, we extend and explore the scope of constraints to trees. For monotone constraints an extension of (a slightly modified version of) FREQT (Asai et al., 2002) is described that computes exactly the frequent trees that satisfy these constraints. Besides the optimisation based upon the Apriori property, our algorithm also performs opportunistic pruning on the monotone constraints. The pruning strategy does not prune all trees that falsify the constraints, but all pruned trees do falsify the constraints. In this sense the pruning is opportunistic.

The web mining problem serves as an example for mining frequent trees. In the rest of this paper we will use this example to clarify constraints we introduce.

This paper is organised as follows. In section 2 we give the basic concepts of tree mining and of FREQT in particular. In section 3 constraints for trees are explored and a pruning strategy for monotone constraints is described. We performed a number of experiments on synthetic data sets, to evaluate the monotone tree mining algorithm. The results of these experiments are reported in section 4. In the last section we give conclusions and further research directions.

## 2. Background

In this section we provide the basic concepts and notation that is used in the remainder of this paper.

[†]supported by the Netherlands Organisation for Scientific Research (NWO) under grant no. 612.066.304

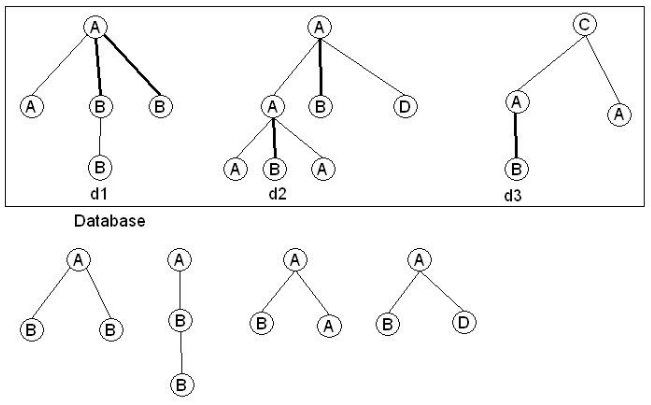


*Figure 1.* Example database with frequent pattern $A - B$ marked with bold lines (upper half). Below the candidate patterns of size three extended from the frequent $A - B$ pattern.

## 2.1. Basic Concepts

A labeled ordered tree $T = \{V, E, \Sigma, L, v_0, \leq\}$ consists of a vertex set $V$, an edge set $E$, an alphabet $\Sigma$ for vertex labels, a labeling function $L : V \rightarrow \Sigma$ that assigns labels to vertices and a binary relation $\leq \subseteq V^2$ that represents a sibling relation among the children of a node. The special node $v_0$ is called the root. If $(u, v) \in E$ then $u$ is the parent of $v$ and $v$ is the child of $u$. For a node $v$, any node $u$ on the path from the root node to $v$ is called an ancestor of $v$. If $u$ is an ancestor of $v$ then $v$ is called a descendent of $u$.

Given two labeled rooted ordered trees $T_1$ and $T_2$ we call $T_2$ an induced subtree of $T_1$, denoted as $T_2 \preceq T_1$ if there exists a injective matching function $\Phi$ of $T_2$ into $T_1$, that satisfies the following conditions for any $v, v_1, v_2 \in V_{T_2}$ :

1. $\Phi$ preserves the parent relation : $(v_1, v_2) \in E_{T_2}$ if $(\Phi(v_1), \Phi(v_2)) \in E_{T_1}$.

2. $\Phi$ preserves the sibling relation :if $v_1 \leq_{T_2} v_2$ then $\Phi(v_1) \leq_{T_1} \Phi(v_2)$.

3. $\Phi$ preserves the labels : $L_{T_2}(v) = L_{T_1}(\Phi(v))$.

Let $D$ denote a database where each transaction $d \in D$ is a labeled rooted ordered tree. For a given pattern tree $t$, which is also a labeled rooted ordered tree, we say $t$ occurs in a transaction $d$ if $t$ is a subtree of $d$. Let $\phi_d(t)$ denote the number of distinct occurrences of $t$ in $d$. Let $\psi_d(t) = 1$ if $\phi_d(t) > 0$ and 0 otherwise. The support of a subtree $t$ in the database $D$ is then defined as $\sum_{d \in D} \psi_d(t)$. A pattern tree $t$ is called frequent if the support of $t$ is greater or equal then a user defined minimum support (minsup) value. The goal of frequent tree mining is to find all frequent trees in a given database. Frequent tree mining algorithms make use of the apriori property: any subtree of a frequent tree is also frequent and any supertree of an infrequent tree is also infrequent.

## 2.2. The FREQT Algorithm

With the background given in section 2.1 we are now able to describe the FREQT (Asai et al., 2002) algorithm. Our work is based upon this algorithm, but can be extended to any other frequent tree mining algorithm that uses the induced subtree relation. The algorithm described below is a slight modification of FREQT: where FREQT uses as a database one data tree, we use a set of trees. The support is defined in FREQT as $\phi_d(t)$ with $d$ the data tree and $t$ the pattern tree. This notion is also known as weighted support. The modification we made does not lead to conceptual differences in the algorithm.

The key notion of FREQT is that frequent subtrees of size $k - 1$, where the size of a tree is defined as the number of nodes, are expanded by attaching a new node only to a node on the rightmost branch of the tree, to yield a larger tree of size $k$. The rightmost branch of a tree is the unique path from the root to the rightmost leaf. This procedure of extending pattern trees ensures that each pattern tree is counted exactly once. The algorithm in pseudo-code is given below:

```
$k \leftarrow 1$
$F_1 \leftarrow$ set of frequent trees of size 1
$RMO_1 \leftarrow$ rightmost occurrences of trees in $F_1$
**While** $F_k \neq \emptyset$
    $k \leftarrow k + 1$
    $C_k, RMO_k \leftarrow$ compute the candidate trees
        and their occurrences from $(F_{k-1}, RMO_{k-1})$
    $F_k \leftarrow$ count $C_k, RMO_k$
```

Consider figure 1 which shows an example database with three data trees. The pattern tree $t$ with nodes $A - B$ occurs twice in $d_1$ and $d_2$ and once in $d_3$, hence $t$ has a support of 3. The candidate trees generated from $t$ according to the rightmost extension technique are shown in the lower half of figure 1. From $d_3$ no candidate trees could be generated.

## 3. Mining Trees with Monotone Constraints

In this section we define monotone constraints for trees and describe an opportunistic pruning method to compute frequent trees.

### 3.1. Constraints for Trees

The different types of constraints defined for sets (Ng et al., 1998) can also be applied in structured data mining. The structure of the data can be ignored and the constraints can be applied as a post-processing step of the mining algorithm. This approach has not all benefits of constraint based mining, because all frequent trees have to be computed and afterward all trees that falsify the constraints are thrown away. Efficient algorithms that directly mine the trees that satisfy the constraints (beside the frequency constraint) are however not available yet. We now give a brief outline how the different classes of item set constraints can be pushed deeply into the mining process of structured data. For this we assume that each vertex has an attribute which holds a measure of interest. This corresponds with e.g. the price of items, or any other defined measure in frequent itemset mining. When we define an SQL based aggregate function over a tree $t$, such as $sum(t)$, $avg(t)$, $min(t)$, the function is applied on the attributes of all the vertexes of $t$.

**Anti-monotone constraints:** A constraint $C$ defined over a tree $t$ has the anti-monotonicity property if

$$C(t) = true \Rightarrow \forall t' \preceq t \;: C(t') = true$$

Examples of anti-monotone constraints are: minimum frequency, $sum(t) \leq constant$ (for positive values of the node attributes of $t$), $size(t) \leq constant$. The minimum frequency constraint is already incorporated in structured data mining algorithms, hence other constraints which have the anti-monotonicity property can be incorporated in the same way.

**Succinct constraints:** A constraint $C$ is succinct if $C$ is pre-counting prunable. Examples are: $min(t) \leq c$ and $max(t) \leq c$. Succinct constraints are computed in frequent itemset mining by first selecting the items that satisfy the constraints and then generating the candidates using these items. In frequent tree mining the same approach could be used, but by applying the rightmost expansion, or any other known method to enumerate all trees that satisfy the constraints, a lot of trees may be missed or enumerated twice. In order to incorporate these constraints, another enumeration technique would be needed.

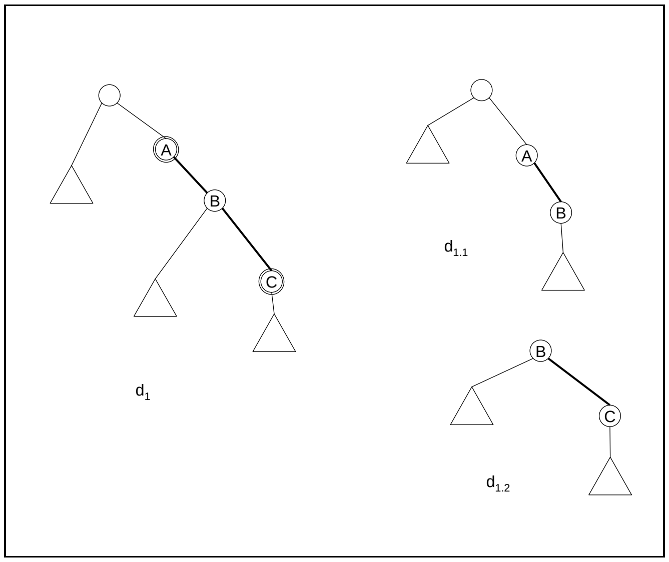
**Convertible constraints:** A constraint $C$ is convertible (anti)monotone if there is an order among the items, such that whenever the items are pushed in order the constraint becomes (anti)monotone. Consider as an example the constraint $avg(t) \geq 10$. This constraints becomes anti-monotone if the items are added in descending order. To incorporate this class of constraint in structured data mining is probably hard, because the order in which the attributes must be processed to become (anti)monotone will in general not coincide with the order imposed by the structure of the data.

**Monotone constraints:** A constraint $C$ defined over a tree $t$ is monotone if:

$$C(t) = true \Rightarrow \forall t' \succeq t \;: C(t') = true$$

Examples of monotone constraints are $sum(t) \geq c$ (for positive values of the node attributes of $t$) and $size(t) \geq c$. Monotone constraints can be mined with a top down algorithm, which starts with the complete set of items as in (Bayardo, 1998). The straightforward adoption of the top down approach does not work for frequent tree mining, because we don't have in general one tree which is a supertree of all frequent trees. Instead there is a set of supertrees, which is exactly the database.

To illustrate the use of monotone constraints for trees consider a weblog application. Suppose we have a database of web access logs at a popular site, where each record (transaction in frequent itemset terms) is the entire forward accesses of a visitor. Suppose we are interested in the most frequently accessed subtrees at that site. Vertices in the tree correspond to webpages, edges to visitor's action going from one webpage to another. To each webpage a value is assigned according



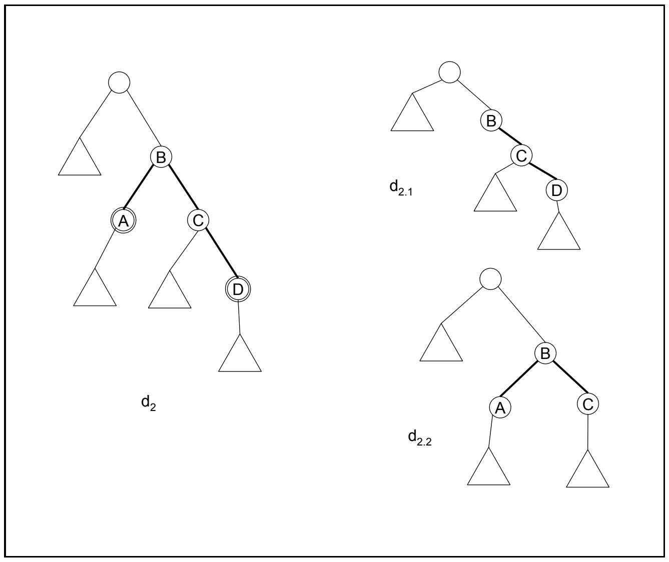


*Figure 2.* A data tree is split up into two subtrees whenever a candidate pattern tree is infrequent (case one left and case two right).

to some interest measure of the end user. For example, we could associate with each vertex (webpage), the number of images contained by the page. Another constraint could be that the end user is only interested in frequent accessed subtrees of size bigger than some constant.

### 3.2. Algorithm

To mine trees with monotone constraints we need the following basic property of trees:

> *each pair of vertices is connected by exactly one path.*

From this property it follows that: candidate pattern trees which are infrequent split the data tree into two or more (not necessarily disjoint) subtrees.

We distinguish two cases. In the first case the pattern tree forms a chain of length $n - i$ with $i < n$, $t_1 = (v_i, v_{i+1}, \ldots, v_n)$. In the second case there is exactly one node of this chain that has two children, say node $v_j$ with $i \leq j \leq n$, hence we have two chains $(v_i, v_{i+1}, \ldots, v_n)$ and $(v_j, v_{j,1}, \ldots, v_{j,m})$. For more general patterns the data tree is split into more than two subtrees, but we will not further discuss these cases because it is of no practical use for the pruning algorithm. For the first case, suppose that the extension of $t_1$ with a node $v_{n+1}$ is infrequent. Then for all data trees $d \in D$ in which $t_1$ occurs, any frequent pattern in which $v_i$ or any ancestor of $v_i$ occurs can not be extended with $v_{n+1}$, because the path from an ancestor of $v_i$ to $v_{n+1}$ leads through $v_i$ and the pattern $t_1$ extended with $v_{n+1}$, is infrequent so any of its supertrees is also infrequent. The node labeled $v_{n+1}$, and any of its descendants, can not be extended with the node labeled $v_i$, because any path from a descendant of $v_{n+1}$ through node $v_i$ leads through $v_{n+1}$, and hence the infrequent pattern $(v_i, v_{i+1}, \ldots, v_n, v_{n+1})$ must be part of any such path. As an immediate result it follows that for all data trees $d \in D$ in which $t_1$ occurs, $d$ can be split into two trees $d_1$ and $d_2$. Here $d_1 = d \setminus t(v_{n+1})$ and $d_2 = t(v_{i+1})$, where $t(v_k)$ is the subtree of $d$ starting at node $v_k$ and $d \setminus t(v_k)$ is the tree $d$ minus it's subtree starting at node $v_k$. Note that both $d_1$ and $d_2$ share the tree $t(v_{i+1}) \setminus t(v_{n+1})$. For the second case the argumentation is similar, given that the pattern tree consisting of the two chains is extended with a node $v_{n+1}$ on the rightmost path $(v_i, v_{i+1}, \ldots, v_n)$ is infrequent. Then for all data trees $d \in D$ in which the pattern tree occurs, $d$ is split into two trees $d_1$ and $d_2$, with $d_1 = d \setminus t(v_{n+1})$ and $d_2 = d \setminus t(v_{j,m})$.

In figure 2 (left) an example is given of the first case, where a frequent pattern $A, B$ is extended with a node $C$. In case that the extended pattern $A, B, C$ is infrequent, $d_1$ is split into $d_{1.1}$ and $d_{1.2}$. The second case is displayed in figure 2(right), where the frequent pattern $A, B, C$ is extended with $D$. The trees $d_{2.1}$ and $d_{2.2}$ are the result of the split of $d_2$.

Recall that monotone constraints have the following property: if a tree $t$ satisfies a monotone constraint, then all its supertrees also satisfy that constraint. Likewise, if a tree $t$ falsifies a monotone constraint then all subtrees of $t$ falsify that constraint as well. The procedure for pruning with a monotone constraint $C$ is as follows.

For all trees $d \in D$ that satisfy $C(d)$, whenever a candidate tree is generated which is infrequent and of case one or two, check the constraint on the two data trees (obtained after splitting) for all occurrences of this infrequent pattern. Whenever a data tree $d$ does not satisfy the monotone constraint, the occurrences of a pattern tree $t$ in $d$ are not counted. Furthermore the pattern $t$ is not extended in $d$; in fact $d$ can be deleted from the database. Suppose that in figure 2 the data tree $d_1$ satisfies a monotone constraint. Because the pattern $(A, B, C)$ is infrequent, $d_1$ is split into $d_{1.1}, d_{1.2}$. If one of the data trees $d_{1.1}, d_{1.2}$ does not satisfy the constraint, these data trees can be deleted. Suppose $d_{1.1}$ falsifies the constraint, then $\phi_{d_1}(A, B)$ is decreased (with one, in this case) which results in a decreased support of the pattern $(A, B)$.

## 4. Experiments

For the experiments we used a slightly modified synthetic data generator provided by Mohamed Zaki. This data generator (described in (Zaki, 2002)) mimics website browsing behavior. Our experiments have been performed on the same dataset, but with different minimum support parameters. The dataset was created by sampling 10000 trees of maximal depth 5 out of a master tree with 10000 nodes and 50 different node labels. Figure 3 shows the distribution of the frequent trees by length for the two minimum support parameters used. Note that in general there are much more possible labeled subtrees of size $n + 1$ then of size $n$. The goal of the experiments is to examine the use of monotone constraints, so we compared the running time and the number of candidates generated for our implementation of FREQT, with the extended version that also performs monotone constraints pruning. We compared the two versions with several values for the constraints, expressed as selectivity. A selectivity of $x\%$ means that $x\%$ of the frequent trees satisfies the constraint. The running time and the number of candidates generated is compared with the basic case for different levels of selectivity, where the basic case (indexed at 100 in figures 4 and 5 ) is the FREQT algorithm without constraints pruning.

The effects of the opportunistic pruning are more visible for the experiments with a higher support. The maximal speedup obtained with minimum support of 1% is 2.40 while the maximal speedup with minimum support 0.5% is 1.80. The same difference holds for the number of candidate trees generated, for the runs with the lowest percentage of selectivity. With a minimum support of 1% the algorithm generated about 41% of the candidate trees generated without constraint pruning, while for a minimum support of 0.5% the algorithm generated about 60% of the candidate trees. The most likely explanation is that the pruning method we use depends on the occurrence of infrequent patterns. When the number of infrequent patterns generated is decreased (by using a lower minsup value), the effect of pruning with monotone constraints flattens.

## 5. Conclusion

In this paper we described types of constraints used in frequent itemset mining, and ways of incorporating these constraints in frequent tree mining. We have described an opportunistic pruning method for tree mining that uses monotone constraints. The opportunistic pruning methods leads to a considerable speedup and a reduction in the number of candidates generated compared to the basic algorithm. In future work, we will explore the development of algorithms that directly compute frequent trees that satisfy succinct and convertible constraints. Besides the constraints inherited from frequent item set mining, it makes sense to define constraints on the structure of the data, and develop algorithms that compute the frequent structures that satisfy these constraints. Extending constraints to embedded subtrees, free trees and graphs is another challenge for future work.

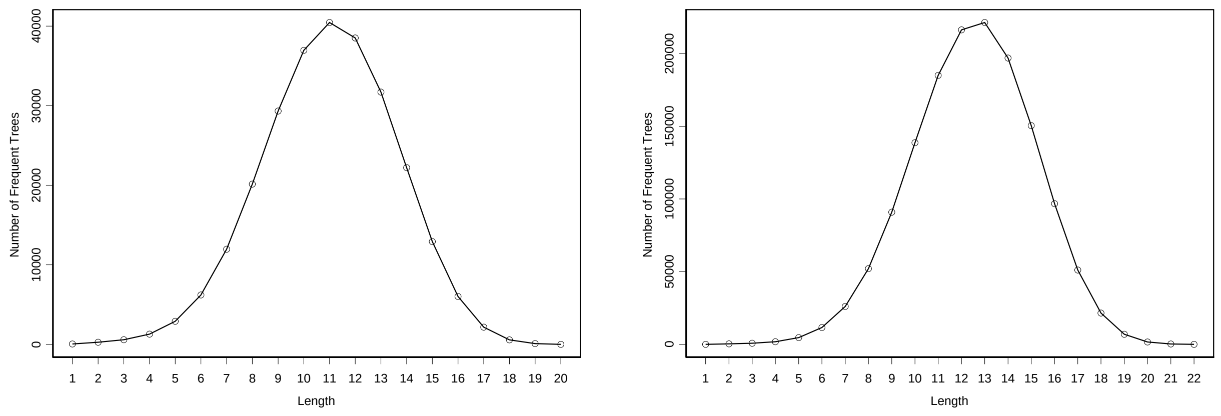


*Figure 3.* Distributions of frequent trees by their size. The distribution on the left is with a minimum support parameter of 1% on the right minsup = 0.5%.

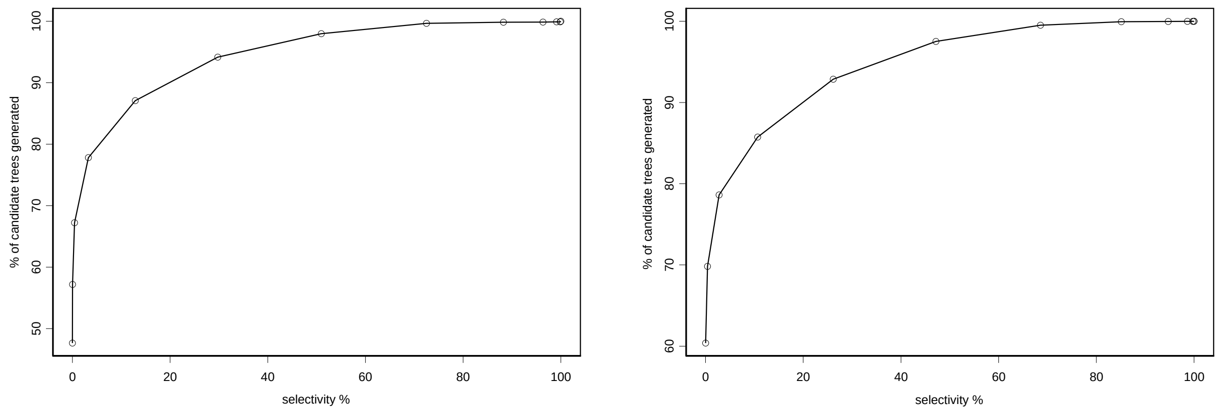


*Figure 4.* The number of candidate trees generated (as a percentage of the number of candidates generated without constraint pruning) for different levels of selectivity. Left of minsup 1% and right with misup 0.5 %

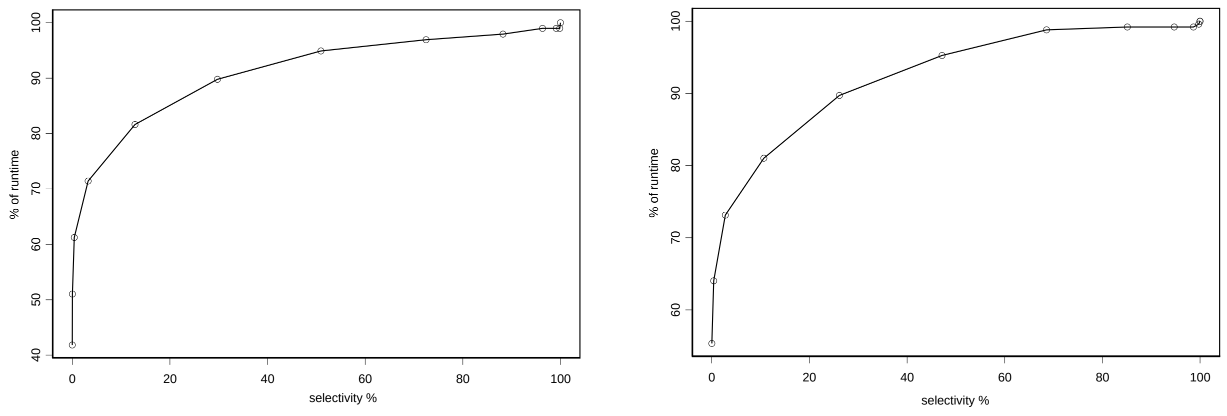


*Figure 5.* The run time for different levels of selectivity for minsup 1 % left and minsup 0.5 % right.The runtime is here also displayed as a percentage of the runtime without constraint pruning.

## References

Agrawal, R., & Srikant, R. (1994). Fast algorithms for mining association rules. *Proc. 20th Int. Conf. Very Large Data Bases, VLDB* (pp. 487–499).

Asai, T., Abe, K., Kawasoe, S., Arimura, H., Sakamoto, H., & Arikawa, S. (2002). Efficient substructure discovery from large semi-structured data. *Proceedings of the Second SIAM International Conference on Data Mining.*

Bayardo, R. (1998). Efficiently mining long patterns from databases. *SIGMOD 1998, Proceedings ACM SIGMOD International Conference on Management of Data, June 2-4, 1998, Seattle, Washington, USA* (pp. 85–93).

Bayardo, R., Agrawal, R., & Gunopulos, D. (1999). Constraint-based rule mining in large, dense databases. *Proceedings of the 15th International Conference on Data Engineering* (p. 188).

Inokuchi, A., Washio, T., & Motoda, H. (2000). An apriori-based algorithm for mining frequent substructures from graph data. *Principles of Data Mining and Knowledge Discovery* (pp. 13–23).

Ng, R. T., Lakshmanan, L. V. S., Han, J., & Pang, A. (1998). Exploratory mining and pruning optimizations of constrained association rules. *SIGMOD 1998, Proceedings ACM SIGMOD International Conference on Management of Data, June 2-4, 1998, Seattle, Washington, USA* (pp. 13–24).

Nijssen, S., & Kok, J. (2003). Efficient discovery of frequent unordered trees. *In Proceedings of the first International Workshop on Mining Graphs, Trees and Sequences (MGTS2003)* (pp. 55–64).

Pei, J., & Han, J. (2000). Can we push more constraints into frequent pattern mining? *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining.*

Wang, K., & Liu, H. (2000). Discovering structural association of semistructured data. *Knowledge and Data Engineering*, *12*, 353–371.

Zaki, M. J. (2002). Efficiently mining frequent trees in a forest. *Proc. of the Int'l Conf. on Knowledge Discovery and Data Mining* (pp. 71–80).