Scalable Constraint Programming approach for Mining Frequent Sequence with gap constraints

http://sites.uclouvain.be/cp4dm/spm/

John AOGA, Pierre Schaus
UCLouvain, ICTEAM, Belgium

Tias Guns
KU Leuven, DTAI Research group, Belgium

Keywords: Sequence Pattern Mining, PrefixSpan, Constraint Programming

Abstract

Sequence mining is an important tool for analyzing large databases of timed events, such as in click stream mining and event log mining. Recently, constraint programming (CP) approaches for pattern mining are gaining interest, due to the modularity of the framework and flexibility to add additional constraints. While CP systems were less scalable than specialized mining systems, we recently showed this can be overcome by hybridizing advanced CP techniques (trailing) with algorithmic improvements. In this work, we study the more involved task of mining under the restriction that the time gap between two matching events must be smaller then a threshold. We show that this too can benefit greatly from hybridization.

1. Sequential Pattern Mining (SPM) under gap\([M,N]\) constraint

Given a set of sequences SDB and a threshold \(\theta\), the goal is to find all subsequences that are included in at least \(\theta\) of the sequences. Consider the following sequence database, where we assume that each event is represented by an individual letter, and the time of the event is the index of the event in the sequence (e.g. ‘B’ happens at time 2 and 5 in \(sid_1\)):

\[
\{ (sid_1, \langle ABDCB \rangle), (sid_2, \langle BAADCB \rangle), (sid_3, \langle ABDDDBEC \rangle), (sid_4, \langle ACCB \rangle) \}
\]

The pattern \(\langle BC \rangle\) has frequency 3, it is included in \(sid_1\) at corresponding positions (2, 4), in \(sid_2\) at position (1, 5) and in \(sid_3\) at (2, 7) and (5, 7).

A gap\([M,N]\) constraint changes when a subsequence is included in a sequence, namely iff the gap between two subsequent symbols is larger or equal than \(M\) and smaller or equal to \(N\). For example, with a gap\([0,2]\) constraint, \(\langle BC \rangle\) has only frequency 2, at positions (2, 4) in \(sid_1\) and (5, 7) in \(sid_3\).

More formally, the problem of SPM under gap\([M,N]\) is to find all patterns \(p = \langle p_1, p_2, \ldots, p_l \rangle\) such that \(|S \in SDB \text{ s.t. } \exists(e_1, e_2, \ldots, e_l) \text{ where } \forall i, S[e_i] = p_i \land i \in [2, l], M \leq e_{i-1} - e_i - 1 \leq N| \geq \theta\); so at least \(\theta\) matching sequences, where \(e_i\) represent the matching position of item \(p_i\) in a sequence \(S\).

Without gap constraint, it does not matter that a subsequence can be embedded in a sequence at different positions, only the smallest position matters. This means that storing this smallest position, the pseudo-projection (Pei et al., 2001), is sufficient and a linear scan of each sequence is enough to compute the projection of an extension of the pattern, e.g. from \(\langle BC \rangle\) to \(\langle BCA \rangle\).

In (Kemmar et al., 2015) a global constraint is introduced based on the pseudo-projection of PrefixSpan idea, adapted to handle gap constraints. The idea is to compute and store all possible embeddings of a pattern in a sequence.

The contributions of our work are that we show how to improve on this approach by 1) precomputing the last position of each symbol in a sequence and using this to quickly determine that an embedding can...
not be extended; 2) using a trail-based backtracking aware datastructure to store the embeddings efficiently. It is based on our earlier work (Aoga et al., 2016) which showed that hybridizations of algorithmic improvements and CP-inspired backtrack-aware datastructures can outperform existing CP and specialized methods. Our new results show that with gap restrictions, where needing to store the embeddings increases the memory requirements, the use of backtrack-aware datastructures is even more beneficial.

2. Backtracking-aware datastructures

A constraint programming problem consists of variables that can take certain values, and constraints over these variables. At the heart of a constraint solver is a generic depth-first search algorithm. Conceptually, it recursively assigns a variable to one of its values and then calls each constraint. A constraint removes values from other variables, or fail if it is violated.

In our case, the variables are the symbols in the pattern $\alpha = v_1, \ldots, v_i$ and variables will be assigned in order. At the heart of our approach is the PPICgap constraint, which stores the embeddings of the pattern in the data, and uses this to count in how many sequences the pattern is included.

To do so, the constraint will internally build the so-called projected database of a pattern $\alpha$ under the gap constraint: $SDB_{[M,N]}^{[\alpha]}$. Each time the pattern is extended, the new gap-projected database can be constructed from the previous one. When the search backtracks (an extension is removed again), the gap-projected database of the original pattern must be restored. E.g. from ($BCA$) back to ($BC$), to consider extension ($BCB$).

We propose to store and restore projected-database by using CP trailing techniques as illustrated in Fig. 1. We use three vectors: $sid$, $emb_size$ and $embs$, respectively for the sequence id, the number of embeddings of the pattern in that sequence, and the set of actual embeddings. These vectors are reversible vectors: when a pattern is extended, the embeddings are read and the new embeddings are appended at the end of the vector, together with the start position and the length ($\phi$ and $\varphi$). If the search backtracks, the previous start position $\phi$ and length $\varphi$ are used and values after $\phi + \varphi$ will be overwritten. This is much more memory efficient then having to copy and delete the embeddings in memory each time.

3. Experiments

We report experimental with an implementation in Scala in OscaR solver (OscaR Team, 2012) on two real-life datasets.

![Figure 1. Reversible vectors technique.](image)

![Figure 2. CPU times for PPICgap and GAP-SEQ for several minsup under gap$^{[1,7]}$.](image)

References


1https://sites.google.com/site/cp4spm/