A Recommendation Engine based on Adaptive Automata

Paulo Roberto Massa Cereda and João José Neto

Escola Politécnica, Departamento de Engenharia de Computação e Sistemas Digitais, Universidade de São Paulo, Av. Prof. Luciano Gualberto, s/n, Travessa 3, 158, CEP: 05508-900, São Paulo, SP, Brasil

Keywords: Adaptive Automata, Recommendation Engine.

Abstract: The amount of information available nowadays is huge and in raw state; systems have to act proactively on selecting and presenting context-relevant information, but such feature is time-consuming an exhaustive. This paper presents a recommendation engine based on an adaptive rule-driven device – namely, an adaptive automata – as a lightweight scalable alternative to usual approaches on resource recommendation. The technique employed here is based on frequency analysis instead of relying on usual machine learning.

1 INTRODUCTION

The growth of social media and the diffusion of the Internet itself brought several challenges to the way data are traditionally analyzed; the amount of information available is huge and in raw state. In order to enhance browsing and learning experience, systems have to be proactive by selecting and presenting context-relevant information, for example, offering related subjects in an e-learning environment or by suggesting movies based on customer’s history records; such feature is usually time-consuming and exhaustive for the whole computational infrastructure (Cereda and José Neto, 2014).

Adaptivity is the term used to denote the ability of a device to modify its own behavior without external interference (José Neto, 1994). For example, in a chess game, such modifications are triggered as the application of a rule pattern given a certain situation, like a pawn that advances all the way to the opposite side of the chess board and triggers a rule which promotes it to another piece of that player’s choice; the pawn now has its own behavior changed to another piece and may act like so. Any rule-driven device can exploit the self-modification feature by adding an adaptive layer on top of its underlying rule set. In this paper, we will use an adaptive automata as computational model, which is proven to be Turing-complete (Rocha and José Neto, 2000).

We present here a simple and fast recommendation engine that uses a well-known computational model enhanced with an adaptive mechanism as a lightweight alternative to usual approaches for tackling raw data and trying to extract patterns and behaviors in situations for which absolute answers are too costly, and suboptimal solutions can be accepted. The use of an adaptive rule-driven device allows a compact yet expressive model which escalates on demand based on the current usage scenario (Cereda and José Neto, 2014).

2 BACKGROUND

There is a vast literature on how to identify and recommend resources that are relevant for a person or a group of people based on a couple of criteria. One of the most known methods for this purpose is called a recommender system (Resnick and Varian, 1997).

Recommender systems usually rely on two techniques known as collaborative and content-based filtering. The former is preference-oriented, recommending resources according to information obtained from groups of people that share similar trends and behaviors (Su and Khoshgoftaar, 2009). The latter is profile-oriented, which classifies resources and recommends similar ones based on interests of some person (Lops et al., 2011). Both techniques do have shortcomings, such as data sparsity and scalability (Cacheda et al., 2011); hybrid versions are usually employed in order to explore the strongest points of each original approach without ignoring the current context being considered (Adomavicius and Tuzhilin, 2005).

More recently, machine learning algorithms have been used for extracting association rules based on
Available data and finding patterns and trends (Gotardo et al., 2013). The main purpose is to establish correlation rules amongst resources. Although being extremely powerful, such techniques are very expensive in terms of computational efforts and might demand an impractical time, depending on the size of the data and how resources are represented in the system (Yahia and Murtada, 2010).

Cereda et al. (Cereda et al., 2009) describe an alternative approach for recommender systems, relying on discrete metrics and adaptive techniques. The recommendation process was managed by an adaptive rule-driven recognizer; by submitting two resources \( r_i \) and \( r_j \) to the recognizer \( M \) as an input string \( w \), for example, if \( w \in L(M) \) means that \( r_j \) could be a good recommendation given \( r_i \) and vice versa. One of the most interesting aspects pointed out by that early work was the model efficiency, since it could handle queries on a pair of related resources in polynomial time based on the length of the input string.

The main shortcoming of this approach, however, is the lack of proactivity, that is, the recognizer accepts/rejects queries about related resources, but it does not answer which resource could be a recommendation. Besides, the model suffers from data sparsity, leading to a problem known as cold start (Basiri et al., 2010). Inspired by this initial work, we propose here a new model, built from scratch, which aims at minimizing the effects of data sparsity and cold start and offering a lightweight scalable alternative to usual approaches on resource recommendation.

It is worth noting that most approaches so far take the artificial intelligence path (Adomavicius and Tuzhilin, 2005), including graph-based recommender systems using an \( n \)-layer approach in order to establish correlation rules amongst relevant resources (Huang et al., 2004); however, the number of layers and resources might greatly increase data sparsity and increase computational processing. Our goal in this paper is an attempt to provide a new model based on an adaptive rule-driven device – namely, an adaptive automaton – that escalates on demand according to the current scenario, reducing the effects of data sparsity and handling queries in polynomial time in order to recommend context-relevant resources. Such features are desirable specially when dealing with a huge amount of raw data.

\[ \text{The Recommendation Engine} \]

In this section, we present an adaptive rule-driven device being used to implement a recommendation engine. We opted for an adaptive automaton (José Neto, 1994; José Neto, 2001), which consists of a traditional automaton as underlying device with the addition of an adaptive mechanism, as seen in Fig. 1; such mechanism allows the device to change its own configuration on runtime by invoking adaptive functions which can modify the underlying set of rules.

![finite automaton](Q, \Sigma, P, q_0, F)

[Figure 1: An adaptive automaton.]

We provide a list of elements used in this section for quick reference; refer to Table 1 for a complete description of the symbols used in the text. Let us consider a set \( R \) of resources. A resource might be any object relevant to a given context, e.g., books in a bookstore or dishes in a restaurant. For our example, the number of elements in \( R \) is fixed, although it is easy to change this model in order to obtain a recommendation engine that considers resources added on-the-fly.

The adaptive automaton of our model has all available resources from \( R \) represented as states \( Q_0 \) subset), but only related resources are connected through transitions. Connection between two resources may contain intermediate states indicating the strength of the connection; the first occurrence of a pair of resources triggers the adaptive mechanism which adds transitions from one to another, and later occurrences will increase this path by adding intermediate states. The number of intermediate states in a path is used to discover the best resource \( r_j \) given \( r_i \). It is worth noting that paths are established if and only if there is an occurrence of pairs of resources, greatly reducing the model size.

A resource \( r_j \) may establish connection with an arbitrary number of other resources from \( R \); such connections indicate possible trends from a group of people or even describe mutual dealings among resources (set \( T \) of special symbols). Our choice was modeling resource connections not as a single two-way symbol but as a pair of unidirectional ones, that is, a link between \( r_i \) and \( r_j \) will result in both \( \tau_{(r_i, r_j)} \) and \( \tau_{(r_j, r_i)} \). This is helpful if we want to keep track of the path to follow when analyzing possible recommendations or if we do not want to consider commutativity between two resources. For the scope of our paper, we are considering that commutativity is allowed.

Fig. 2 represents the adaptive automaton \( M \) of our recommendation engine. For illustration purposes,
we wrote a very simple automaton with just a pair of resources, $R = \{r_1, r_2\}$, so the recommendation here is quite obvious; nevertheless, it is enough for allowing the observation of how the adaptive layer operates when changing the automaton topology.

As seen in Fig. 2 (a), the initial state $q_0$ has a loop for each resource $r_i$ with an associated parametric adaptive function $A$ with argument $r_i$ triggered before the symbol consumption from the string of resource names. The call to the adaptive function $A(r_i)$

<table>
<thead>
<tr>
<th>Element</th>
<th>Meaning</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>$Q$</td>
<td>set of states</td>
<td>$Q = {q_0, q_{pizza}, q_{chips}}$</td>
</tr>
<tr>
<td>$F \subseteq Q$</td>
<td>subset of accepting states</td>
<td>$F = {q_{pizza}, q_{chips}}$</td>
</tr>
<tr>
<td>$q_0 \in Q$</td>
<td>initial state</td>
<td>—</td>
</tr>
<tr>
<td>$R$</td>
<td>set of resources</td>
<td>$R = {pizza, chips}$</td>
</tr>
<tr>
<td>$Q_R \subseteq Q$</td>
<td>subset of states associated with resources, $</td>
<td>Q_R</td>
</tr>
<tr>
<td>$T$</td>
<td>set of special symbols representing links amongst resources, $</td>
<td>T</td>
</tr>
<tr>
<td>$\tau_{(r_i, r_j)} \in T$</td>
<td>link between two resources $r_i$ and $r_j$, with $r_i, r_j \in R, i, j \in \mathbb{N}, i \neq j$</td>
<td>$\tau_{(pizza, chips)}$</td>
</tr>
<tr>
<td>$D$</td>
<td>set of adaptive functions</td>
<td>$D = {A, B}$</td>
</tr>
<tr>
<td>$\Sigma$</td>
<td>input alphabet</td>
<td>$\Sigma = {\alpha, pizza, chips}$</td>
</tr>
<tr>
<td>$\alpha$</td>
<td>resource pair delimiter</td>
<td>—</td>
</tr>
<tr>
<td>$P$</td>
<td>mapping relation</td>
<td>$P = {(A(pizza), q_0, pizza) \mapsto (q_{pizza}, e, e), (A(chips), q_0, chips) \mapsto (q_{chips}, e, e)}$</td>
</tr>
</tbody>
</table>

$\mathcal{A}(r)$ | adaptive function $\mathcal{A} \in D$ with argument $r_i$ triggered before the symbol consumption | — |

$\mathcal{B}(q, x, z)$ | adaptive function $\mathcal{B} \in D$ with arguments $q, x, z$ triggered after the symbol consumption | — |

$f_Q: R \mapsto Q_R$ | function that takes a resource $r_i \in R$ and returns its associated state $q_{r_i} \in Q_R$ | $f_Q(\text{pizza}) \mapsto q_{\text{pizza}}$ |

$G: Q_R \mapsto R$ | output relation that maps each state $q_{r_i} \in Q_R$ representing a resource $r_i \in R$ into a resource $r_j \in R - \{r_i\}$ | $G(\text{pizza}) \mapsto \text{chips}$ |

$\arg\max_{\mathcal{A} \subseteq R} A$ | $y_{\max} = \max_{(x, y) \in \mathcal{A}} y$, then $\arg\max_{\mathcal{A}} A = \{x \in R \mid (x, y_{\max}) \in \mathcal{A}\}$ is the set of $x$ on which $y$ is maximized | $\arg\max_{\mathcal{A}} A = \{(\text{egg}, 7), (\text{pizza}, 5), (\text{bacon}, 7), (\text{spam}, 6)\} = \{\text{egg}, \text{bacon}\}$ |

$\mathcal{A}(a, b) \rightarrow (d, e, f)$ | rule-searching elementary adaptive action that searches $P$ for rules matching the given pattern $(a, f \subseteq D \cup \{e\}, b, d \subseteq Q, c \subseteq \Sigma, e \subseteq \Sigma \cup \{e\})$ | — |

$- \mathcal{A}(a, b) \rightarrow (d, e, f)$ | rule-erasing elementary adaptive action that removes rules matching the given pattern from $P$ $(a, f \subseteq D \cup \{e\}, b, d \subseteq Q, c \subseteq \Sigma, e \subseteq \Sigma \cup \{e\})$ | — |

$+ \mathcal{A}(a, b) \rightarrow (d, e, f)$ | rule-inserting elementary adaptive action that adds rules with a specified pattern to $P$ $(a, f \subseteq D \cup \{e\}, b, d \subseteq Q, c \subseteq \Sigma, e \subseteq \Sigma \cup \{e\})$ | — |
(from set \( D \) of adaptive functions) will evoke a new configuration, with a transition from \( q_0 \) targeting state \( q_i \), consuming the symbol \( r_j \). In our example, \( q_0 \) has now a transition to \( q_{r_2} \), consuming \( r_1 \) (Fig. 2 (b)); according to the input string format, \( r_2 \) is related to \( r_1 \), so a call to \( \mathcal{A}(r_2) \) in \( q_{r_1} \) will evoke a new configuration (Fig. 2 (c)), where these two resources are now related.

If a resource has not been considered in the recommendation engine yet (Fig. 2 (a) with resource \( r_1 \)), a modification is performed in the automaton topology in order to accommodate the newly considered resource; that is done through a call to the adaptive function \( \mathcal{A} \) (Algorithm 1), which will remove and add transitions based on a set of elementary adaptive actions defined in the adaptive function body. Similarly, if an unconsidered resource is related to an existing one, not only this resource must be considered, but the recommendation path will also be built between them (as seen in Fig. 2 (b) with the loop in \( q_{r_1} \), and the resulting recommendation path in (c) between \( r_1 \) and \( r_2 \)). Adaptive function \( \mathcal{A} \) takes one parameter denoting the resource to be considered. The syntax used in the function body and a complete description of elementary adaptive actions can be found in (José Neto, 1994).

If both resources \( r_i \) and \( r_j \) have been considered in the engine before, there will be a transition from \( q_0 \) to \( q_i \), consuming \( r_i \), and from \( q_i \) to \( q_j \), consuming \( r_j \) with a call to the adaptive function \( \mathcal{B} \). Adaptive function \( \mathcal{B} \) increments the number of transitions and intermediate states between two resources by adding a newly inserted state (chosen by a generator \( g^2 \) in the function body) and modifying existing transitions in order to accommodate such new element in the recommendation path. The function takes three parameters denoting the source state followed by two link symbols \( \tau_1, \tau_2 \in T \).

**Algorithm 1: Adaptive function \( \mathcal{A}(r) \).**

**Summary:** this function modifies the automaton topology in order to accommodate the newly considered resource \( r \); it creates recommendation paths between \( r \) and each considered resource; at last, it removes the selected transition from the mapping.

**adaptive function \( \mathcal{A}(r) \)**

**variables:** \( \forall, \forall, r, z \rightarrow q_0, \mathcal{A}(r) \)

\[-(q_0, r) \rightarrow q_0, \mathcal{A}(r),\]

\[+(q_0, r) \rightarrow f_0(r),\]

\[\mathcal{A}(q_0, r) \rightarrow 2x,\]

\[+(f_0(r), q) \rightarrow f_0(q), \mathcal{B}(q, r, \tau_{r_1}), \tau_{r_2}, \tau_{r_1, r_2})\]

\[+(f_0(q), r) \rightarrow f_0(r), \mathcal{B}(r, \tau_{r_1}), \tau_{r_1, r_2})\]

\[+(f_0(r), \tau_{r_1}) \rightarrow f_0(q)\]

\[+(f_0(r), z) \rightarrow f_0(r),\]

\[+(f_0(r), z) \rightarrow f_0(r), \mathcal{A}(z)\]

\[+(f_0(r), x) \rightarrow q_0, \mathcal{A}(x)\]

end adaptive function

An example of how adaptive function \( \mathcal{B} \) modifies the automaton topology is illustrated in Fig. 3. For simplicity, only relevant parts of the automaton are shown.

As seen in Fig. 3, \( \mathcal{B} \) inserts the new state \( q_i \) between \( r_i \) and \( q_j \), rearranges existing transitions and adds new ones. When any transition holding an associated adaptive function \( \mathcal{A} \) is executed, the corresponding recommendation path will increase in number of intermediate states.

Resources \( r_i \) and \( r_j \) are connected via a recommendation path, composed by their associated states, \( q_{r_i} \) and \( q_{r_j} \), and an intermediate state \( q_k \) with transi-
Algorithm 2: Adaptive function \( B(q, \tau_1, \tau_2) \).

**Summary:** this function adds a new element in the recommendation path between two resources. The higher the number of elements, the stronger the path.

**adaptive function** \( B(q, \tau_1, \tau_2) \)

**variables:** \(?y\)

**generators:** \( g^* \)

\( ?(q, \tau_1) \rightarrow ?y \)

\( -(q, \tau_1) \rightarrow ?y \)

\( +?(\tau_2) \rightarrow q \)

\( +(g^*, \tau_2) \rightarrow q \)

\( +(g^*, \tau_1) \rightarrow ?y \)

\( +(\tau_2) \rightarrow g^* \)

**end adaptive function**

---

Figure 3: An example of the adaptive function \( B \) being triggered during the transition from \( q_0 \) to \( q_1 \), consuming \( r_j \), resulting on adding a new state \( q_1 \) and a couple of new transitions consuming \( \tau_{(r_j,r_j)} \) and \( \tau_{(r_j,r_j)} \). Intermediate states act as a counter. At the beginning, this path contains no intermediate state from \( q_0 \) to \( q_j \); resources are directly connected through transitions consuming \( \tau_{(r_j,r_j)} \) and \( \tau_{(r_j,r_j)} \), as seen in Fig. 2 (c); this is by design and indicates the strength of the connection between these resources. The number of intermediate states in the path from \( q_i \) to \( q_j \) will be used as a parameter for deciding whether \( r_j \) should be recommended given \( r_i \); the more states, the higher chance for \( r_j \) to be selected. If we go the other way in the path, that is, from \( q_j \) to \( q_i \), the decision changes to whether \( r_i \) should be recommended given \( q_j \). Alternatively, recommendation paths could be represented by a global counter associated with each pair of resources and an adaptive function that triggers an increment operation on demand.

It is important to observe that \( r_i \) and \( r_j \) share the same recommendation path (but of course, not the same transitions as \( q_i \) reaches \( q_j \) through \( \tau_{(r_j,r_j)} \) and \( q_j \) reaches \( q_i \) through \( \tau_{(r_j,r_j)} \)), so our model may end up with at most \( \binom{|R|}{2} \) recommendation paths, one for each \( (r_i, r_j) \) subset, considering that all resources are related and thus connected. If commutativity does not hold between a pair of resources, every ordered pair \( (r_i, r_j) \) should have its own recommendation path, thus, the maximum number of recommendation paths should be \( |T|^2 \); the link set \( T \) already offers symbols to support non-shared recommendation paths, if this rule is chosen.

The adaptive automaton \( M \) of our recommendation engine accepts strings over the alphabet \( \Sigma = R \cup T \cup \{\alpha\} \), \( w \in \Sigma^* \), with \( L(M) = \{ w \in \Sigma^* \mid w = r_1 r_2 (\alpha r_3 r_4)^* \mid r_i, r_j \in R, r_i \neq r_j \} \), in one of the forms presented as follows.

The first form, \( w \in R \), denotes \( w \) as a single resource for which we want to get a recommendation. Here the idea is to submit \( w \) to our automaton and get an output string \( w' \in R \cup \{\varepsilon\} \), with \( w' \neq w \). For example, with \( R = \{\text{egg, bacon, spam}\} \), and submitting \( w = \text{egg} \) to the automaton, we should get an output string \( w' = \text{bacon, meaning that bacon, based on our recommendation engine, is a good side dish for eggs}. \)

The occurrence of \( \varepsilon \) in the output string \( w' \) indicates that no recommendation was found based on the input string \( w \). For practical purposes, the output relation always returns a resource \( r_i \in R \), but that is a domain-specific decision.

The second form, \( w = r_1 r_2 (\alpha r_3 r_4)^* \), with \( r_i, r_j, r_k, r_l \in R, r_i \neq r_j, \alpha \in \Sigma \) and \( r_k \neq r_l \), denotes that \( w \) has at least one pair of resources, or a list of pairs separated by a delimiter \( \alpha \). A string in this form leads the automaton to change its topology according to the frequency of each pair of resources; modifications are possible due to the adaptive mechanism action triggered by a call to a parametric adaptive function \( B \) associated to specific transitions.

The string \( w \) is built from a subset of resources \( R' \subseteq R, |R'| \geq 2 \), in a transaction, for example, a basket containing grocery items in a purchase or a café tab.

...
In order to build a proper string to denote occurrences of each pair of resources from \( R' \), we need to obtain all subsets of two elements of \( R' \), that is, \( P_2(R') \), and separate them by using a special symbol \( \alpha \). Algorithm 3 illustrates the building process of the string \( w \) from a subset \( R' \subseteq R \). For instance, let us consider a set of resources \( R = \{ \text{egg, sausage, bacon, spam} \} \) from a café menu and \( R' = \{ \text{egg, bacon, spam} \} \) as a breakfast order; a call to \( \text{BUILD}(R') \) would give us the string \( w = \langle \text{egg} \rangle \langle \text{spams} \rangle \langle \text{bacon} \rangle \alpha \langle \text{bacon} \rangle \langle \text{spam} \rangle \).

**Algorithm 3:** Building string \( w \) of shape \( r_i r_j (\alpha r_i r_j) \) from \( \subseteq R, |R'| \geq 2 \).

**Summary:** this procedure builds an input string \( w \) to be recognized by the adaptive automaton of our recommendation engine from a subset \( R' \) of resources; it basically iterates through all subsets of two elements of \( R' \) and concatenates the intermediate results with the resource pair delimiter.

```plaintext
procedure \text{BUILD}(R')
\begin{align*}
&w \leftarrow \epsilon \\
&R'' \leftarrow P_2(R') \\
&X \leftarrow r_i \in R' \\
&\text{for } x \in X \text{ do} \\
&w \leftarrow w \cdot x \\
&\text{end for} \\
&R'' \leftarrow R'' - X \\
&\text{if } R'' \neq \emptyset \text{ then} \\
&\text{for } X' \in R'' \text{ do} \\
&w \leftarrow w \cdot (\alpha) \\
&\text{for } x' \in X' \text{ do} \\
&w \leftarrow w \cdot x' \\
&\text{end for} \\
&\text{end if} \\
\end{align*}
return \( w \)
end procedure
```

Special symbol \( \alpha \) plays the role of a pair separator when building the input string \( w \) in the second form; since all transitions consuming this symbol target the initial state, the automaton behaves exactly the same with all pairs. The set of accepting states \( F \) is the union of the set of states \( Q_R \) representing resources from \( R, F = Q_R, F \subseteq Q \).

One of the goals of the proposed recommendation engine is to submit an input string \( w \) in the first form, that is, \( w \in R, \) to our automaton and get an output string \( w' \in R \cup \{ \epsilon \} \), with \( w' \neq w \), as a recommended resource. In order to achieve that, we need to define an output relation \( G \) associated to each state \( q_r \in Q_R \).

In order to determine which resource \( r_j \) to recommend in terms of resource \( r_i \), we rely on counting the number of intermediate states for every recommendation path from \( r_i \) as a metric to indicate a suitable option. The number of intermediate states between two resources \( r_k \) and \( r_l \) (dynamically added by adaptive function \( S \)) gives us the occurrences of the pair \( \{ r_k, r_l \} \); the higher that number, the more frequent the pair is, and thus, resource \( r_l \) can be a recommendation for resource \( r_k \) and vice versa. Alternately, an approach using global counters would simplify the comparison step since all values are directly available.

In order to count the intermediate states between \( r_i \) and \( r_j \) and choose the longest recommendation path, we present a procedure \( \text{RECOMMEND}(r) \) that takes a resource \( r \) and return its best recommendation, according to the highest pair frequency; this is done by inspecting the current automaton shape. The algorithm relies on counting specific transitions, so the number of intermediate states from \( r_i \) to \( r_j \) is \( t - 1 \), with \( t \) being the number of transitions in \( T \) which consume \( \tau_{r_i r_j} \). Algorithm 4 illustrates the steps taken by such procedure in order to get the most suitable resource.

**Algorithm 4:** Obtaining the best resource as recommendation for \( r \in R \).

**Summary:** this procedure that takes a resource \( r \) and return its best recommendation, according to the highest pair frequency; this is done by counting specific intermediate states.

```plaintext
procedure \text{RECOMMEND}(r)
\begin{align*}
&X \leftarrow \{ \} \\
&\text{for } r_i \in R, r_i \neq r \text{ do} \\
&\text{if } q \in P \land \tau_{r_i r_j} \text{ then} \\
&\text{for } q' \in Q \text{ do} \\
&X' \leftarrow \{ q', \tau_{r_i r_j} \} \\
&X \leftarrow X \cup \{ q', \tau_{r_i r_j} \} \\
&\text{end for} \\
&X'' \leftarrow \arg max X' \\
&\text{return } X'' \\
\end{align*}
end procedure
```

Using the procedure presented in Algorithm 4, the output relation \( G \) maps every state \( q_r \in Q_R \) representing resource \( r_i \) into a call to \( \text{RECOMMEND}(q_r) \) which returns a resource \( r_j \neq r_i \) as recommendation; in other words, we get \( G = \{ (q_r) \rightarrow \text{RECOMMEND}(q_r) \mid q_r \in Q_R \} \).

**4 EXPERIMENTS AND DISCUSSIONS**

We developed two experiments in order to illustrate an evaluation of the recommendation engine proposed in this paper. First, we generated a data sample from a set \( R \) of 5 resources, \( R = \{ \text{egg, sausage, bacon, beans, spam} \} \), simulating several breakfast orders in a café; the data sample size ranged from 10 to 1000 entries. It is important to
mention that orders with just one item were discarded since you cannot establish a connection with other items in the same order.

**First Experiment** — The first experiment consisted of checking the hits of our recommendation engine [#1] compared to traditional association rule algorithms, namely Apriori [#2] (Agrawal and Srikant, 1994), PredictiveApriori [#3] (Scheffer, 2001), FP-Growth [#4] (Han et al., 2000) and Tertius [#5] (Flach and Lachiche, 1999). For this experiment, we used a sample size of 100 entries. All algorithms were executed in a 64-bit Linux environment with the help of the WEKA data mining software (Hall et al., 2009). The results are presented in Table 2. The parameters for each algorithm were automatically set by WEKA.

### Table 2: Hits in the first experiment.

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>egg</td>
<td>bacon</td>
<td>spam</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>sausage</td>
<td>egg</td>
<td>egg</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>bacon</td>
<td>spam</td>
<td>spam</td>
<td>spam</td>
<td>beans</td>
<td></td>
</tr>
<tr>
<td>beans</td>
<td>bacon</td>
<td>bacon</td>
<td>beans</td>
<td>bacon</td>
<td></td>
</tr>
<tr>
<td>spam</td>
<td>bacon</td>
<td>beans</td>
<td>bacon</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>


According to Table 2, Apriori and FP-Growth could not provide valid association rules based on the sample data (note that some rules are not valid in our context and were discarded, for example, if a customer doesn’t order bacon, he won’t order spam; we want to recommend resources solely based on positive input); by increasing the sample size, both algorithms start converging to the rules obtained from our model. PredictiveApriori and Tertius had a very similar output, although the latter had one miss when trying to extract a rule for an certain order.

It is also worth noting that the number of resources in $R$ plays an important role in our experiment, since traditional association rule algorithms extract rules based on the power set of $R$, that is, $\mathcal{P}(R)$, and this set might become huge with many elements in $R$. Our model simply tries to establish rules based on one resource in terms of another, using just all subsets of two elements of $R$, that is, $\mathcal{P}_2(R)$, which is quite sufficient for our domain-specific problem.

In general, the results were significant since the design of our model employs an optimized straight-forward technique (namely, frequency analysis) and at some extent the outcome came very close to more sophisticated algorithms such as the ones used in the experiment. We were able to extract meaningful connections at a low cost, minimizing the cold start problem. The key point for a recommendation context is that suboptimal answers are also valid, and thus a more balanced model in terms of hits and performance might be an interesting approach.

**Second Experiment** — The second experiment consisted of measuring the execution time of our recommendation engine compared to traditional association rule algorithms (the ones used in the first experiment). We tested the performance of each approach with the data sample size ranging from 10 to 1000 entries, for 1000 times.; the overview of the second experiment is in Table 3.

### Table 3: Execution time in seconds of all approaches in the second experiment, with the data sample size ranging from 10 to 1000 entries. Values are represented in scientific notation, $\times 10^{-3}$.

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>10.0</td>
<td>3.5</td>
<td>72.1</td>
<td>0.1</td>
<td>11.2</td>
</tr>
<tr>
<td></td>
<td>50.0</td>
<td>3.6</td>
<td>203.0</td>
<td>2.5</td>
<td>16.2</td>
</tr>
<tr>
<td></td>
<td>100.0</td>
<td>3.9</td>
<td>336.0</td>
<td>1.4</td>
<td>29.2</td>
</tr>
<tr>
<td></td>
<td>250.0</td>
<td>4.2</td>
<td>454.5</td>
<td>3.8</td>
<td>40.9</td>
</tr>
<tr>
<td></td>
<td>500.0</td>
<td>8.7</td>
<td>821.9</td>
<td>6.3</td>
<td>120.1</td>
</tr>
<tr>
<td></td>
<td>750.0</td>
<td>16.8</td>
<td>1187.6</td>
<td>9.5</td>
<td>180.6</td>
</tr>
<tr>
<td></td>
<td>1000.0</td>
<td>18.3</td>
<td>1110.4</td>
<td>9.1</td>
<td>178.6</td>
</tr>
</tbody>
</table>

According to Table 3, and based on data from Table 2, the algorithms with better predictions were also the more time-consuming ones. This is quite understandable, since several calculi are employed in order to get the best possible association rule. We can observe that our model had one the best performances amongst all approaches and still with a significant prediction rate. Perhaps the most important fact from this experiment is that our model does not suffer from data sparsity, since resource connections are only established when needed.

### 5 CONCLUSIONS

In this paper, we presented a recommendation engine that uses an ad-hoc adaptive automaton specially designed as a lightweight alternative to existing approaches on resource recommendation. The model is compact, efficient and offers suboptimal answers which are good enough for several specific contexts (Hatem and Rumli, 2014).

As a result of this work, we have achieved a recommendation engine based on a formalism that can be easily modified or extended in order to cover sev-
eral contexts (Cereda and José Neto, 2014). We have reached an interesting balance between performance and prediction, as seen in the two experiments in Section 4, with no significant effects of data sparsity and cold start. As future work, an additional adaptive function $C$ can be used to modify recommendation paths on higher levels (for example, $r_i$ to $r_j$ through $r_k$).

The use of an adaptive rule-driven device provides simplicity and representation power; the recommendation engine shown here has a wide scope and may also be used along with usual artificial intelligence techniques, exploring the strongest points of both sides, and achieving good results when solving real world problems (José Neto, 2001; Cereda and José Neto, 2014).

REFERENCES


