Introduction: Autocompletion supports human–computer interaction in software applications that let users enter textual data. We will be inspired by the use case in which medical professionals enter ontology concepts, catering the ongoing demand for structured and standardized data in medicine.

Objectives: Goal is to give an algorithmic analysis of one particular autocompletion algorithm, called multi-prefix matching algorithm, which suggests terms whose words’ prefixes contain all words in the string typed by the user, e.g., in this sense, opt ner me matches optic nerve meningioma. Second we aim to investigate how well it supports users entering concepts from a large and comprehensive medical vocabulary (SNOMED CT).

Methods: We give a concise description of the multi-prefix algorithm, and sketch how it can be optimized to meet required response time. Performance will be compared to a baseline algorithm, which gives suggestions that extend the string typed by the user to the right, e.g. optic nerve m gives optic nerve meningioma, but opt ner me does not. We conduct a user experiment in which 12 participants are invited to complete 40 SNOMED CT terms with the baseline algorithm and another set of 40 SNOMED CT terms with the multi-prefix algorithm.

Results: Our results show that users need significantly fewer keystrokes when supported by the multi-prefix algorithm than when supported by the baseline algorithm.

Conclusions: The proposed algorithm is a competitive candidate for searching and retrieving terms from a large medical ontology.
Algorithmic and User Study of an Autocompletion Algorithm on a Large Medical Vocabulary

Merlijn Sevenster\textsuperscript{a,}\textsuperscript{*}, Rob van Ommering\textsuperscript{a}, Yuechen Qian\textsuperscript{b}

\textsuperscript{a}Healthcare Information Management, Philips Research, High Tech Campus 34, 5656 AA Eindhoven, The Netherlands
\textsuperscript{b}Clinical Informatics, Interventional and Translational Solutions, Philips Research, 345 Scarborough Road, Briarcliff Manor NY 10510, USA

\textit{Keywords:} autocompletion, user experiment, algorithms, human–computer interaction

\textsuperscript{*}Corresponding author

Email addresses: merlijn.sevenster@philips.com (Merlijn Sevenster),
rob.van.ommering@philips.com (Rob van Ommering), yuechen.qian@philips.com (Yuechen Qian)

Preprint submitted to Journal of Biomedical Informatics  
August 22, 2011
1. Introduction

Autocompletion supports human–computer interaction in software applications that let users enter textual data. Well-known instances include query entry support for search engines, completion of mail addresses by e-mail management tools, and suggestion of previously visited URLs by web browsers.

Following [1], we distinguish four benefits of autocompletion, which are not necessarily mutually exclusive. First, autocompletion algorithms save keystrokes, and therefore misspellings. Consequently users require less time to enter the intended data. Second, they increase the user experience of the application in which they are embedded. If the autocompletion algorithm gives suggestions that match the user’s context, this reassures the user that the application “understands” his or her situation. Third, autocompletion algorithms help explore the underlying vocabulary. This enables the user to enter the intended data, even if he or she cannot spell it out. For instance, supercalifragilisticexpialidocious is suggested by Google if we type superc. Finally, they support adoption and usage of standardized vocabulary, such as the vocabulary of ontologies.

The first three benefits seem to be relevant to medical software applications, such as search engines that aid the user in finding health-related information [2, 3, 4]. The fourth benefit is of particular importance to the field medicine in view of the ongoing interest in data standardization in this area. For instance, autocompletion is virtually indispensable in healthcare information management systems in which medical professionals can fill and modify structured data fields.

Various medical vocabularies have been proposed that describe (parts of) the medical domain. Well-known vocabularies include the International Classification of Diseases, RadLex (describing the radiology domain), Breast Imaging-Reporting and Data System (describing the breast cancer domain), Current Procedural Terminology (describing medical services and procedures for billing purposes), Medical Subject Headings, and Systematized Nomenclature of Medicine–Clinical Terms (SNOMED CT).

In this paper we shall focus on the vocabulary of SNOMED CT, a large vocabulary that gives a comprehensive description of the medical domain using more than 480,000 terms.

1 Google accessed February 3rd 2011 by corresponding author
In many applications, the suggestions given by the autocompletion functionality extend the string the user is typing to the right. In SNOMED CT, for instance, in this manner, the string *optic* could yield *optician*, *optic cup*, *optic pit* and *optic axis*. This type of autocompletion algorithms can be implemented efficiently by storing the underlying vocabulary as a prefix tree.

For benchmarking purposes, we developed a baseline algorithm that completes the entered string to the right, as described above. The baseline autocompletion algorithm selects terms in a breadth-first fashion and is thus named the *standard breadth-first* algorithm (SBF).

In this paper we study an autocompletion algorithm that retrieves and ranks the terms whose words’ prefixes contain the words in the string typed by the user. In this way, *opt ne s m* matches *optic nerve sheath meningioma*. The algorithm is order independent, meaning that the latter term would also be retrieved by *m s ne opt*. We give a comprehensive description of our *multi-prefix matching* algorithm (MPM), and prove two computational properties.

We seek answers to the following research questions:

Q1 Does MPM offer more efficient support for entering SNOMED CT terms than SBF?

Q2 What query strategies do users follow to enter SNOMED CT terms with MPM?

Q3 How close are these query strategies to the optimal query strategies, and can this be exploited for further optimizing MPM?

Q4 How can the difference between MPM and SBF be explained from an information-theoretic point of view?

To answer Q1–Q3, we conducted a user experiment in which the participants were invited to produce a preselected set of SNOMED CT terms. To answer Q4 we apply the notion of entropy to the autocompletion task.

Even though we evaluate our algorithms against SNOMED CT, the algorithms are vocabulary independent, that is, they can be used to autocomplete any given vocabulary. We are confident that a significant portion of our results, such as the answers to Q1–Q4, transfer to settings in which the algorithms are based on other vocabularies than SNOMED CT.

The multi-prefix matching algorithm was originally developed for browsing ontology concepts. It is currently implemented in a longitudinal tumor
information management prototype that has more than twenty structured fields some of which accept only terms from a predefined vocabulary (e.g., histology). The user study was conducted in a dedicated environment that bore no resemblance to the prototype whatsoever. Participants for the user study were recruited from the department of the first and second author, and may be somehow representative for the medical IT personnel and clinical professionals targeted by the tumor information management system.

We saw two products that has order independent and word-by-word prefix matching autocompletion: in the contact finder field of Microsoft’s Office Communicator and the concept browser field of CliniClue\(^2\). For instance, in Office Communicator the name of the second author (Rob van Ommering) is retrieved by the query `om ro.\(^3\) We could not find a description of these algorithms or literature that is otherwise related, but we believe that the user experience of these algorithms is most similar to that of the multi-prefix matching algorithm.

In fact, we found surprisingly few publications on algorithmic \([5, 6]\) or usability aspects of autocompletion. This may be due to the fact that a straightforward algorithm, such as our baseline algorithm, suffices on smaller vocabularies. When such an algorithm does not achieve optimal results and the vocabulary increases in size, an advanced data structure needs to be chosen to guarantee acceptable response time. We can imagine that corporations (e.g., companies developing search engines) do not want to disclose these data structures as it might give away their competitive advantage. For the same reason they might be reluctant to publish on the usability of their autocompletion interfaces.

The structure of this paper is as follows. In Section 2 we describe MPM and SBF. In Section 3 we prove two computational properties of MPM, one of which concerns NP-hardness of its optimal query strategy. In Section 4 we describe the user experiments and their results. We use these results to answer our research questions in Section 5. Section 6 concludes the paper.

\(^2\)www.cliniclue.com

\(^3\)Interestingly, unlike our multi-prefix matching algorithm, we found that Office Communicator’s autocompletion algorithm is not monotone: the longer queries `om v ro` and `ro v om` yield no results.
2. Methods

All queries and vocabulary terms are strings, i.e., sequences of characters. For two strings $s$ and $s'$ we write $s \preceq s'$ if $s$ is a prefix of $s'$, that is, if $s'$ starts with $s$ or is equal to $s$. A word is a string that does not contain the space character. A word in $s$ is a substring $s_i$ of $s$ such that each substring of $s$ that contains $s_i$ is not a word. We shall often write a string $s$ as an (ordered) sequence $(s_1, \ldots, s_n)$ or set $\{s_1, \ldots, s_n\}$ of its words.

The length of a string $s$, i.e., the number of characters it contains, is denoted by $|s|$. The empty string, i.e., the string of length 0, is denoted by $\epsilon$.

2.1. Autocompletion algorithms

In this section we describe the baseline algorithm and the multi-prefix matching algorithm. It is important to point out that both algorithms execute two tasks: exploring SNOMED CT for matching terms and ranking the retrieved terms. The ranking of the terms determines the order in which they are suggested to the user. The ranking aspect of autocompletion algorithms is especially important on larger vocabularies, as even queries of medium length may yield more matches than the user can expected to inspect.

In terms of efficiency, we required that the implementations of both algorithms can handle each query instantly ($< 0.1$ sec) so that the list of shown suggestions can be refreshed as the user types. We implemented both algorithms in C#; their description in this paper is language independent though.

2.1.1. Multi-prefix matching

Suppose we want to select the concept for meningioma in the optic nerve but we have insufficient knowledge of SNOMED CT to anticipate the way this concept might be described in this vocabulary. For instance, we conceive it possible that SNOMED CT contains meningioma in optic nerve, meningioma in optical nerve, optic nerve meningioma or optical nerve meningioma. The multi-prefix matching algorithm allows the user to execute an order-independent search on prefixes. So, for instance, the queries opt ner me and me opt ner yield the same list of SNOMED CT terms, including the intended term optic nerve meningioma.

Given a query, the multi-prefix matching retrieves all terms in SNOMED CT that match it, and consecutively ranks the retrieved terms by the quality of the matching.
A matching from $q = \{q_1, \ldots, q_m\}$ to $t = \{t_1, \ldots, t_n\}$ is a function $F : \{1, \ldots, m\} \to \{1, \ldots, n\}$ such that

- **Injectivity:** for every $1 \leq i, j \leq m$, if $i \neq j$ then $F(i) \neq F(j)$;
- **Prefix preserving:** for every $1 \leq i \leq m$, $q_i \preceq F(i)$.

Injectivity implies that two query words may not be matched to the same term word; functionality of $F$ implies that every query word must be mapped to a term word. So there is a matching from `optic nerve` to `optic nerve sheath meningioma`, but there is no matching from `optic atrophy`.

When no confusion threatens, we shall not make a distinction between the index $i$ of a query or term word and the string $q_i$ or $t_i$ itself, respectively, and write $F(q_i)$ instead of $F(i)$.

In addition to knowing if there is a matching, we want to know the quality of the matching. To this end, we introduce a scoring function $\sigma$ that assigns high score to a matching $F$ if it maps every query word $q_i$ to a term word $F(q_i)$ that is (almost) equal to $q_i$, i.e., that has a high average score for $|q_i|/|F(q_i)|$. So, for a matching $F : \{q_1, \ldots, q_m\} \to \{t_1, \ldots, t_n\}$ we define

$$
\sigma(F) = \frac{1}{n} \sum_{i=1}^{m} \frac{|q_i|}{|F(q_i)|}.
$$

The matching from `optic nerve` to `optic nerve` has score $3/5$, whereas the matching from `optic nerve` to `optic nerve sheath meningioma` has score $3/10$. The factor $1/n$ penalizes matches with terms that contain words that are not involved in the matching. Note that the notion of matching is order independent: the query `ner opt` matches the same terms as `opt ner`, with the same score. The scoring function can be refined by rewarding terms that respect the word order in the query, but we will not pursue this topic in this paper.

Algorithm 1 constructs a matching from a query $q$ to a term $t$. In this algorithm, the $\cup$-operator returns `null` if any of its arguments is `null`. Assuming the query words $(q_1, \ldots, q_m)$ are ordered by decreasing word length, we prove in Section 3.1 that GetMatching finds a matching if there is one, and that it finds the matching with highest score if there are multiple matchings. This property ensures that the process of ranking matchings by their score is not affected by biases in the routine that construes the matches. The worst-case
Algorithm 1: GetMatching

```
input: a sequence of query words \((q_1, \ldots, q_m)\) and a set of term words \(t\)

if \(m = 0\) then
    return \(\emptyset\)
else
    let \(V\) be \(\{t_i \in t : q_1 \preceq t_i\}\)
    if \(V = \emptyset\) then
        return null
    else
        let \(v\) be an element of \(V\) minimizing \(|v|\)
        return \(\{(q_1, v)\} \cup \text{GetMatching}((q_2, \ldots, q_m), t - \{v\})\)
```

complexity of GetMatching for a query with \(m\) words and a target with \(n\) words is \(O(mn)\).

As an example of GetMatching, consider \(q = (ab, a)\) and \(t = \{ab, abc\}\). In the first iteration \(ab\) is mapped to \(ab\). In the second iteration, \(a\) is mapped to \(abc\). The score of this matching is \(2/3\). Note that there is another matching than the one constructed by GetMatching. It is the one that maps \(a\) to \(ab\) and \(ab\) to \(abc\). This matching has score \(7/12\).

The multi-prefix matching algorithm sorts the matched terms by the score of their matching with the query. A straightforward, but not so economic, implementation is given in Algorithm 2. To achieve acceptable response time for all queries (< 0.1 sec.) on SNOMED CT, we use three techniques that restrict the number of GetMatching calls (not reflected in Algorithm 2).

First we order the words \(w\) in all SNOMED CT terms alphabetically, and assign to each \(w\) the SNOMED CT terms in which it appears. This alphabetical ordering allows us to find the range of words \(w_1, \ldots, w_k\) starting with query word \(q_i\) through binary search. For each query word \(q_i\) we select the set of candidate terms \(C_i\) by taking the union of the sets assigned to \(w_1, \ldots, w_k\). Then we take the intersection of all these sets for all query words \(q_1, \ldots, q_m\):

\(C = \cap_{i=1}^m C_i\). Instead of iterating through all SNOMED CT terms, we iterate through \(C\). One of the reviewers points out that this technique of reducing the set of candidates is quite similar to the way support is calculated in association rule mining [7].

We associate with each term \(t\) a bitstring \(b_t\) of length 26 in which the \(i\)th bit has value 1 if, and only if, there is a word in \(t\) starting with the
ith character (using the canonical alphabetical ordering). The bitstring $b_t$ is called $t$'s finger print. For each term $t$ in $C$, we check whether the finger print $b_q$ of the query $q$ is a subset of the finger print of $t$. This can be implemented as a single bitwise-and operation. If it is indeed a subset, we compute $\text{GetMatching}(q,t)$, if it is not we proceed to the next $t$ in $C$.

The above techniques suffice to achieve acceptable response time for queries of length 2 or more. The third technique supports 1-letter queries by precomputing their autocomplete offline and caching them.

Algorithm 2: Multi-prefix matching (MPM)

```
input: a query $q$
let $L$ be an empty list
let $q'$ be the sequence of words in $q$ ordered by decreasing word length
foreach $t = \{t_1, \ldots, t_n\} \in \text{SNOMED CT}$ do
    let $F$ be $\text{GetMatching}(q',\{t_1, \ldots, t_n\})$
    if $F$ is not null then
        add $(t, \sigma(F))$ to $L$
    end if
end foreach
sort $L$ on the score $\sigma(F)$ in $(t, \sigma(F))$
return $L$
```

2.1.2. Standard breadth first

The suggestions of the standard breadth-first autocompletion algorithm extend the string the user is typing to the right, that is, for a given query $q$, all its suggestions start with $q$. A prefix tree (also known as trie [8]) is the natural data structure to support this type of autocompletion. The prefix tree of a corpus of terms is a tree in which every prefix of a term is represented by a node and in which the nodes are optimally shared between prefixes. For instance, the prefix tree of the small corpus brain, brain stem, breath test, breast and breast pain is shown in Figure 1. Every node is assigned a boolean indicating if the prefix to which the node corresponds is a term in the corpus. Such nodes are drawn as boxes in Figure 1. Note that the node corresponding to breath is not boxed, since breath is not a term in the corpus.

Prefix trees can be compressed, by, for instance, merging a node with its successor if it has only one successor. These optimizations will not be pursued in the present paper.

The baseline algorithm conducts a breadth-first search in the prefix tree starting from the node that corresponds to the query, see Algorithm 3. For
Figure 1: The prefix tree of a five-term corpus
instance in the prefix tree in Figure 1, the query bre yields breast, breast pain and breath test. Note that it does not yield breath since this is not a term in the corpus. The query bar yields no suggestions as the prefix tree contains no node corresponding to bar.

**Algorithm 3:** Standard breadth first (SBF)

input: a query q
let L be an empty list
if there is a node T that corresponds to q then
    let \( T = \{ T \} \)
while \( T \neq \emptyset \) do
    foreach \( T \in T \) do
        if \( T \) corresponds to a term in the corpus then
            add the term corresponding to \( T \) to \( L \)
            add every child node of \( T \) to \( T \)
        remove \( T \) from \( T \)
return \( L \)

The SBF algorithm can be used in a cascading way, i.e., using autocompletion suggestions to obtain parts of the target. For instance, suppose we want to produce the term infection by dipylidium caninum. Using SBF, infection is the second suggestion for the query infe. Thus, we can obtain it by selecting the second suggestion (by pressing DOWN) and copying it to the query field (by pressing ENTER). The query now reads infection and we can continue producing infection by dipylidium caninum from there.

3. Theory

In this section we give an algorithmic analysis of the multi-prefix matching algorithm. First we prove that GetMatching returns the matching with maximum score whenever there is a matching. This result shows that there is no hidden bias in the GetMatching routine in the way matchings are constructed. Second we prove that in general (i.e. on a hypothetical vocabulary) finding the shortest query that yields an arbitrary term from the vocabulary is NP-hard. This shows that part of keystroke savings of the MPM algorithm are theoretical, and that there is an important gap between the theoretical analysis of MPM and its practical usability.
The contents of this section are independent from the user study described in the next section. These two sections can thus be consumed in arbitrary order.

3.1. Completeness and maximality of GetMatching

**Theorem 1.** Let $q$ and $t$ be strings, and write $Q = (q_1, \ldots, q_m)$ for the sequence of words in $q$ ordered by decreasing word length and $T = \{t_1, \ldots, t_n\}$ for the set of words in $t$. Let $F = \text{GetMatching}(Q, T)$.

- $F$ is **null** if, and only if, there is no matching from $Q$ to $T$.

- If $F$ is not null, then there is no matching $F'$ from $Q$ to $T$ with $\sigma(F') > \sigma(F)$.

**Proof.** We prove by induction on $m$. If $m = 1$, the claim is immediate, as there is only one matching, which is obviously found by GetMatching.

For the inductive case, assume $m > 1$. It is straightforward to check that if $F$ is not null, then there is a matching from $Q$ to $T$.

Conversely, suppose $F$ is null. We distinguish two cases. First: the set $V$ is empty, meaning that there is no $v$ in $T$ starting with $q_1$. In this case, there is no matching from $Q$ to $T$ and we are done.

Second: $V$ is not empty, but $\text{GetMatching}(Q', T')$ is null, where $Q' = (q_2, \ldots, q_m)$ and $T' = T - \{v\}$. For the sake of deriving a contradiction, suppose there exists a matching $F$ from $Q$ to $T$. Let $w \in T$ be the word in $T$ for which $F(q_1) = w$. It follows that $q_1 \preceq w$, or, put differently, $w \in V$. Since $F$ is a matching from $Q$ to $T$ there also is a matching from $Q'$ to $T - \{v\}$.

Let $v$ be among the shortest elements in $V$. By definition of $V$ we have that $q_1 \preceq v$, and since $v$ is among the shortest elements in $V$, $v \preceq w$. Since there is a matching from $Q'$ to $T - \{v\}$, there must also be a matching from $Q'$ to $T'$, because every query word $q_i$ that can be mapped to $v$ can also be mapped to $w$. By the inductive hypothesis, $\text{GetMatching}(Q', T - \{v\})$ is not null. Contradiction.

For the second item, suppose $F$ is the matching $\text{GetMatching}(Q, T)$ and $G$ is a matching from $q$ to $t$ such that $G(q_1) \neq v$. Let $q_k$ be the query word in $Q$ for which $G(q_k) = v$ and let $v^*$ be the term word in $T$ for which $G(q_k) = v^*$.

Let $H$ be the function defined by

$$
H(q_i) = \begin{cases} 
  v & \text{if } i = 1 \\
  v^* & \text{if } i = k \\
  G(q_i) & \text{otherwise.}
\end{cases}
$$

$H$ is the function that maps $q_1$ to $v$, $q_k$ to $v^*$ and that is otherwise like $G$.

To see that $H$ is a matching from $q$ to $t$, we need to check that $q_1 \preceq v$ and that $q_k \preceq v^*$. The first claim follows from the same reasoning as above.

As for the second claim, observe that $q_k \preceq v$, since $G(q_k) = v$. We further have that $q_1 \preceq v$ and that $q_1$ is among the longest strings in $Q$. Therefore, $q_1$ is at least as long as $q_k$ and it follows that $q_k \preceq q_1$.

From the fact that $|v| \leq |v^*| \leq |q_k| \leq |q_1|$ it follows that

$$\frac{|q_1|}{|v|} + \frac{|q_k|}{|v^*|} \geq \frac{|q_k|}{|v|} + \frac{|q_1|}{|v^*|}.$$  

Hence, the score of $H$ is at least as high as that of $G$: $\sigma(H) \leq \sigma(G)$.

The score of $F$ can be written as

$$\sigma(F) = \frac{1}{n} \left( \frac{|q_1|}{|v|} + \sigma(F') \right),$$

where $F' = \text{GetMatching}(Q', T - \{v\})$. By the inductive hypothesis, there is no matching from $Q'$ to $T - \{v\}$ that has a higher score than $F'$. So in particular the restriction of $H$ to $Q'$ and $T - \{v\}$ does not have a score higher than that of $F'$. Since $F$ and $H$ agree on the term word to which $q_1$ is mapped, it follows that $\sigma(H) \leq \sigma(F')$ and therefore $\sigma(G) \leq \sigma(F)$. We conclude that there is no matching that has a higher score than the one produced by GetMatching. \hfill $\Box$

### 3.2. Computing the minimum number of keystrokes is NP-hard for MPM

To research improvements of MPM we wish to explore the gap between the strategies users follow to produce a term and the optimal query strategy. We define the optimal query for a given term $t$ as the shortest string for which MPM yields $t$ as the first suggestion. In this section we prove that the task of computing the optimal query is NP-hard in the general case (i.e., in the case where the vocabulary is not fixed) [9]. This implies that there is no efficient algorithm (i.e., polynomial-time algorithm) that computes the optimal query strategy, unless $P = NP$. It is generally accepted that NP-hard problems are not efficiently computable by humans. It follows that users can generally not execute the optimal query strategy for MPM.

We define the task of computing the shortest query for MPM as the decision problem $\text{Min-TPS-QUERY}$. For a given vocabulary of strings $C$, target string $t \in C$ and integer $k$, is there a query $q$ such that $t$ is the first suggestion
of MPM on $q$ and $|q| \leq k$? As usual we shall prove NP-hardness of our decision problem by reduction from another decision problem $A$ that is known to be NP-hard. A reduction $R$ is a polynomial-time computable function that maps instances $\alpha$ of $A$ to instances of MIN-MPM-QUERY in such a way that $\alpha$ is a positive instance of $A$ if, and only if, $R(\alpha)$ is a positive instance of MIN-MPM-QUERY. If $\alpha$ is a positive instance of $A$, we also write $\alpha \in A$, and likewise for positive instances of other decision problems. The idea behind a reduction is that the problem to which $A$ is reduced is at least as hard as the problem $A$, modulo the polynomial overhead of transforming an $A$ instance to a MIN-MPM-QUERY instance.

We reduce from the decision problem EXACT-COVER-BY-3-SETS, which is defined as follows. Let a family $F$ of sets $\{S_1, \ldots, S_{3k}\}$ be given such that $k > 1$ and $|S_i| = 3$ for all $1 \leq i \leq 3k$. Are there $k$ sets in $F$ whose union cover $\bigcup F$?

A selection of sets that cover all elements in $\bigcup F$ is called a cover. If the cover contains $k$ sets, then each pair of distinct sets in the cover have empty intersection. In this case it is called an exact cover. Observe that if $F$ has an exact cover, $\bigcup F$ contains precisely $3k$ elements. It is known that EXACT-COVER-BY-3-SETS is NP-complete [10].

**Theorem 2.** MIN-MPM-QUERY is NP-hard.

**Proof.** We reduce from EXACT-COVER-BY-3-SETS, that is, we give a polynomial-time computable function that transforms an instance $F$ to an instance $(C, t^*, k')$ of MIN-MPM-QUERY in such a way that

$$F \in \text{EXACT-COVER-BY-3-SETS} \text{ iff } (C, t^*, k') \in \text{MIN-MPM-QUERY}. \quad (1)$$

The reduction is exemplified right after the proof. Denote $\bigcup F$ by $U$. We use the following numbers:

- $n = 3k$ (i.e., the size of $F$)
- $\ell = \lceil \log_2(n) \rceil$
- $N = \frac{(2\ell - 1)kn}{2\ell}$
- $M = 3n^3$.

In our computations, $M$ can be taken as any large number greater than $3n^3$. For integers $0 \leq i < 2^\ell$, $b_\ell(i)$ denotes the binary representation of $i$ in $\ell$ bits. For integers $j$, $0^j$ denotes the string $0 \ldots 0$ of length $j$. 

14
For every object \( a \in U \), we introduce a term \( t(a) = (t(a)_1, \ldots, t(a)_n) \) in which
\[
t(a)_i = \begin{cases} 
  b_{\ell}(i)0^{(n-\ell)} & \text{if } a \in S_i \\
  b_{\ell}(i)0^{M-\ell} & \text{if } a \not\in S_i.
\end{cases}
\]

Further we introduce for each \( 0 < \ell' < \ell \), for each string \( c \) in \( \{0, 1\}^{\ell'} \) and for each \( 1 \leq j \leq k \) the term \( r(c, j) = (r(c, j)_1, \ldots, r(c, j)_n) \) such that
\[
r(c, j)_i = \begin{cases} 
  c & \text{if } i = j \\
  b_{\ell}(i)0^{M-\ell} & \text{otherwise}.
\end{cases}
\]

Finally we introduce the term \( t^* = (t^*_1, \ldots, t^*_n) \) to \( C \) with \( t^*_i = b_{\ell}(i)0^{N-\ell} \), for every \( 1 \leq i \leq n \). Collect all the terms thus introduced in \( C \). \( C \) contains \( 3k + k2^{\ell-1} + 1 \) terms, which is polynomial in \( k \).

All terms in \( C \) have precisely \( n \) term words. Therefore if there is a matching \( F \) from a given string \( q \) with \( m \) words to a term \( t \in C \), the fraction \( 1/n \) in \( \sigma(F) \) is constant for all terms in \( C \). For this reason we shall use the simpler score function
\[
d(q, t) = \sum_{i=1}^{m} \frac{|q_i|}{|F(q_i)|},
\]
where \( F \) is the matching produced by \text{GetMatching}(q, t), if any.

We proceed to prove (1) for the instance \((C, t^*, \ell k)\) thus obtained.

From left to right. Suppose \( G \) is an exact cover of \( F \), i.e. \( G \) is a subset of \( F \) with size \( k \) such that \( \bigcup G = U \). Let \( q = (q_1, \ldots, q_n) \) be the query such that
\[
q_i = \begin{cases} 
  b_{\ell}(i) & \text{if } S_i \in G \\
  \epsilon & \text{if } S_i \not\in G.
\end{cases}
\]
The function that maps \( q_i \) to \( t^*_i \) is the only matching from \( q \) to \( t^* \). Since \( t^*_i \) has length \( N \) for every \( 1 \leq i \leq n \), it follows that
\[
d(q, t^*) = \frac{\ell k}{N} = \frac{2\ell^2}{(2\ell - 1)n}.
\]

We have to prove that \( d(q, t^*) > d(q, t) \), for all terms \( t \) in \( C \) other than \( t^* \) itself.

Let \( t(a) \) be the term associated with the object \( a \). Note that the function that maps \( q_i \) to the \( i \)th word in \( t(a) \) is the only matching from \( q \) to \( t(a) \).

Since \( G \) is an exact cover there is precisely one set \( S_i \in G \) that contains \( a \).
For this index \(i\), the \(i\)th term word in \(t(a)\) has length \(n\) and since \(a \in S_i\), \(q_i\) adds \(|q_i|/n = \ell/n\) to the total score of \(t(a)\). For the \(j \neq i\) for which \(q_j \neq \epsilon\) (of which there are \(k - 1\)), since \(a \not\in S_i\), \(q_j\) adds \(\ell/M\). The remaining query words \(q_j\) for which \(q_j = \epsilon\) add nothing to the score of \(t(a)\). Hence,

\[
d(q, t(a)) = \frac{\ell}{n} + \frac{(k-1)\ell}{M}.
\]

It is straightforward to verify that \(d(q, t^*) > d(q, t(a))\).

Consider the term \(r(c, j)\), whose \(j\)th word is \(c\) with length \(|c| < \ell\). Since every query word in \(q\) has length \(\ell\), no matching (if any) maps a query word from \(q\) to \(c\). So if there is a matching,

\[
d(q, r(c, j)) = \frac{k\ell}{M},
\]

which is clearly smaller than \(d(q, t^*)\).

From right to left. Let \(q\) be a query of length \(k\ell\) such that \(d(q, t^*) > d(q, t)\), for all \(t\) in \(C\) other than \(t^*\). By the same reasoning as in the from-left-to-right direction,

\[
d(q, t^*) = \frac{\ell k}{N} = \frac{2\ell^2}{(2\ell - 1)n}.
\]

Suppose there is a query word \(q_i\) in \(q\) for which \(|q_i| > \ell\). Then, \(q_i\) adds \(|q_i|/k\) to \(d(q, t(a))\), for any \(a \in S_i\). Hence, \(d(q, t(a)) \geq |q_i|/k\). It is easy to check that \(d(q, t(a)) > d(q, t^*)\). Contradiction. We may thus assume without loss of generality that \(|q_i| \leq \ell\), for every query word \(q_i\) in \(q\).

Suppose there is query word \(q_i\) in \(q\) for which \(|q_i| < \ell\). Let \(r(c, j)\) be a term from \(C\) such that \(c = q_i\). One can verify that there is matching from \(q\) to \(r(c, j)\). Since \(c = q_i\), \(q_i\) adds 1 to the score of \(d(q, r(c, j))\). But then \(d(q, r(c, j)) > d(q, t^*)\). Contradiction. We may thus assume without loss of generality that each query word \(q_i\) in \(q\) has length \(\ell\).

Let \(\mathcal{G}\) be the cover induced by \(q\), that is, the cover such that \(S_i \in \mathcal{G}\) if, and only if, \(q_i \neq \epsilon\). For the sake of contradiction, assume that \(\mathcal{G}\) is not an exact cover. This implies that \(\mathcal{G}\) has two or more overlapping sets. For the moment, assume we have two sets \(S\) and \(S'\) that have an object \(a\) in common.

Since \(a\) is contained in at least two sets in \(\mathcal{G}\) we have that

\[
d(q, t(a)) \geq \frac{2\ell}{n}.
\]
It follows that \( d(q, t(a)) > d(q, t^*) \). Contradiction. It follows that the cover \( \mathcal{G} \) induced by \( q \) is an exact cover. □

As an example of the reduction, consider \( \mathcal{F} = \{ S_1, \ldots, S_6 \} \). In this case, \( k = 2, n = 6, \ell = 3, N = 10, \) and \( M = 1,296 \). Let us suppose that the element 1 is only contained in \( S_1 \) and \( S_2 \). Then, the words of \( t(1) \) are the following:

\[
\begin{align*}
t(1)_1 &= 000000 \\
t(1)_2 &= 001000 \\
t(1)_3 &= 010000 \ldots 000 \ldots 000 \\
t(1)_4 &= 011000 \ldots 000 \ldots 000 \\
t(1)_5 &= 100000 \ldots 000 \ldots 000 \\
t(1)_6 &= 101000 \ldots 000 \ldots 000 \\
\end{align*}
\]

For each \( c \in \{0,1\}^1 \cup \{0,1\}^2 \) and \( 1 \leq j \leq 6 \), we introduce the object \( r(c,j) \). For instance, \( r(01,3) \) has the following term words:

\[
\begin{align*}
r(01,2)_1 &= 000000 \ldots 000 \ldots 000 \\
r(01,2)_2 &= 001000 \ldots 000 \ldots 000 \\
r(01,2)_3 &= 01 \\
r(01,2)_4 &= 011000 \ldots 000 \ldots 000 \\
r(01,2)_5 &= 100000 \ldots 000 \ldots 000 \\
r(01,2)_6 &= 101000 \ldots 000 \ldots 000 \\
\end{align*}
\]

The objects \( r(c,j) \) act as distractors: if a query has a term word with length less than \( \ell = 3 \) (assuming the other query words start with the binary encoding position of their index), there is a matching with at least one of the terms \( r(c,j) \) which has very high score (\( > 1 \)). Such queries can therefore not be used to retrieve the target term \( t^* \).

The words in the target term have intermediate length, i.e., smaller than \( M \) but larger than \( n \):

\[
t^*_1 = 000\overbrace{000000}^{N-\ell}
\]

17
\[
\begin{align*}
t_2^* &= 001\,0000000 \\
t_3^* &= 010\,0000000 \\
&\vdots
\end{align*}
\]

We have shown that queries whose term words are not of length \(\ell = 3\) cannot retrieve the target term. Consider the element 1, which is contained in the sets \(S_1\) and \(S_3\). Consider the query \(q = (000, \epsilon, 010, \epsilon, \epsilon, \epsilon)\). There is a matching between this query and \(t(1)\), and its score is \(3/6 + 3/6 = 1\). This is higher than \(q\) can achieve with \(t^*\). Thus, \(q\) fails to retrieve \(t^*\). In fact, we need to construe a query \(q^*\) for which each query word matches at most one short term word in every \(t(i)\). The reason is that a matching between such a query \(q^*\) and any \(t(i)\) has score \((1/2 + 3/1,296)\) or \((3/1,296 + 3/1,296)\), which is lower than the score of the matching of \(q^*\) with \(t^*\): \(3/10 + 3/10\). The query \((000, 001, \epsilon, \epsilon, \epsilon, \epsilon)\) is such a query \(q^*\), and it encodes the exact cover \(\{S_1, S_2\}\).

4. Results

In this section we describe the design of our user experiment, and we present its results.

4.1. User experiment

During the experiment each participant was shown a series of terms (targets), one at a time, in a dedicated experiment tool. In total 80 targets were used, which were divided in two subcollections of 40 targets each, called subcollection A and B. Participants received support from SBF for producing targets from the first subcollection, and from MPM for the second subcollection. We divided the participants in two groups. Group 1 produced subcollection A with SBF and subcollection B with MPM, whereas group 2 produced subcollection B with SBF and subcollection A with MPM. This setup of the experiment guarantees that each algorithm supported the completion of each target six times, thus eliminating potential synergies between the algorithms and the subcollections.

The experiment tool was a stand-alone application developed for the sake of this research. The tool contains a field showing the target, a text box in which the user was invited to enter characters and a pick list that was dynamically filled and refreshed with autocompletion suggestions as the user was typing. In our evaluations we used the entire 2009 distribution of SNOMED.
ct, containing 480,691 terms (more than 15.0 MB). The user was able to delete characters from the text box using **DEL** and **BACK** in the usual fashion. Finally, the tool contained a button. Pressing it would advance the participant to the next target. This button could only be pressed when the text box contained a string equal to the target.

The pick list contained at most eight items, ordered by score, see Figure 2. By default the first suggestion in the pick list was selected. The selected suggestion can be moved up and down by pressing **DOWN** and **UP**, respectively. Whenever the user presses **ENTER**, the string typed by the user is replaced by the selected suggestion.

The participants could only interact with the experiment tool by means of the keyboard, so in particular no mouse was provided.

For each target, all keystrokes (alphanumerical and non-alphanumerical) were logged that were used by the participant to produce it. The **number of keystrokes** required by a participant to produce a given target denotes the number of keystrokes logged in the participant’s session for that target. The **percentage of required keystrokes** of a target denotes the number of keystrokes required to produce that target with divided by its length. These will be our metrics to assess the performance of the autocompletion algorithms.

In our analysis we will distinguish **control keys** (**DOWN**, **UP** and **ENTER**), which manipulate the selected suggestion, and **correction keys** (**BACK**, **DEL**, **LEFT** and **RIGHT**), which are used to make a correction.

Since no mouse was provided in the experiment, the participant’s behavior could be reconstructed solely on the basis of the logged keystrokes. Whence, our metric, based on the number of keystrokes, gives a quantitative account of all interactions between the participant and the experiment tool.

Twelve participants were recruited on voluntary basis from the department of the first and second author (Healthcare Information Management of Philips Research) in order of availability, see Table 1. All participants are active as researcher or software engineer in the field of clinical informatics and/or electronic health. This group may be somehow representative for the medical IT personnel and clinical professionals that are targeted by the tumor information management system in which the algorithm is being integrated. The participants were assigned to a group in alternating order.

Prior to the experiment, participants were informed about the experiment’s goal; it was also made clear that the experiment does not involve a (covert) competition. The participants received instructions on the workings of both autocompletion algorithms, and were invited to familiarize themselves...
with the algorithms by producing three training targets. No participant reported any difficulty understanding the workings of the algorithms before or during the experiment. See Appendix A for the experiment’s instructions.

The set of 80 targets was obtained as follows. We randomly selected 20 SNOMED CT terms with 2 words, only containing alphabetical characters, with length at least 30 and at most 44. Similarly we selected 20 terms with 3, 4, and 5 words. We checked that the targets were not controversial so as to not upset participants. No targets were removed; no other checks were performed on the targets. The mean target length is 33.73, standard deviation 3.96. The target collection was subsequently split in two subcollections of 40 targets, each containing 10 terms with $n$ words, for each $2 \leq n \leq 5$. The targets were presented in random order.

4.2. Keystroke results

To investigate the learning effect of interacting with the experiment tool, we averaged the 1st, 2nd, 3rd, ..., 40th target that the participants produced for both algorithms, see Figure 3. Since the order of the targets was randomized for each participant, the values in Figure 3 do not correspond to the mean percentage of required keystrokes for one and the same target. We did not see a significant learning effect of interacting with the experiment tool for either of the two algorithms. Note that this does not indicate that interacting with the algorithms proper has no learning curve, as participants may have been exposed previously to (variants of) either of the autocompletion algorithms. Since we observed no learning effect, we decided to include all datapoints in the remainder of the analysis.
Figure 4 differentiates between the participants; each boxplot in this diagram is based on all 80 targets of the corresponding participant. On average the participants require 59.8% of the keystrokes with respect to the target length (standard deviation 34.3%). The mean percentage of required keystrokes ranges from 51.8% (participant 9) to 71.2% (participant 10).

Figure 5 shows a histogram differentiating between the two autocompletion algorithms. The mean percentage of required keystrokes is 67.0% for SBF (sd 30.0%, median 60.0%) and 52.3% for MPM (sd 36.7%, md 43.3%). This amounts to $0.67 \times 33.7 = 22.6$ and $52.3 \times 33.7 = 17.9$ keystrokes, respectively. Thus, MPM saves an additional 4.7 keystrokes on average.

Participants required significantly fewer keystrokes when producing SNOMED CT terms with MPM support than with SBF support (unpaired, one-tailed Wilcoxon test, $p > 0.999$). There was no significant difference between the percentage of required keystrokes for group 1 and 2 (unpaired, two-tailed Wilcoxon test, $p > 0.9$).

### 4.3. Query analysis

In this section we investigate the queries used by the participants. First we measure the number of correction keystrokes in the experiment. On average users typed 2.1 correction keys when supported by SBF and 2.6 when

<table>
<thead>
<tr>
<th>Participant</th>
<th>Sex</th>
<th>Age</th>
<th>Background</th>
<th>Group</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>M</td>
<td>34</td>
<td>Medicine</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>M</td>
<td>28</td>
<td>Computer science</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>M</td>
<td>30</td>
<td>Biomedical engineering</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>F</td>
<td>27</td>
<td>Computer science</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>M</td>
<td>50</td>
<td>Physics</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>M</td>
<td>28</td>
<td>Computer science</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>M</td>
<td>35</td>
<td>Computer science</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>F</td>
<td>25</td>
<td>Computer science</td>
<td>2</td>
</tr>
<tr>
<td>9</td>
<td>M</td>
<td>57</td>
<td>Psychology</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>M</td>
<td>43</td>
<td>Medical informatics</td>
<td>2</td>
</tr>
<tr>
<td>11</td>
<td>M</td>
<td>31</td>
<td>Bioinformatics</td>
<td>2</td>
</tr>
<tr>
<td>12</td>
<td>F</td>
<td>27</td>
<td>Psychology</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 1: Participants’ information
Figure 3: The percentage of required keystrokes for the $i$th target presented ($1 \leq i \leq 40$) averaged over the participants.
Figure 4: Boxplots showing the variety of percentage of required keystrokes per participant. For each participant, the results of both algorithms are combined. Thus each boxplot is based on 80 targets.
Figure 5: Histogram showing the frequency of percentage of required keystrokes for both algorithms (bin width 5%).
supported by MPM; these figures amount to 7.6% and 6.4% of the mean target length, respectively. The variance in the number of correction keystrokes is larger for SBF than for MPM: standard deviation is 8.2 vs. 4.6 keystrokes. We observed during the experiments that participants detected misspellings sooner if they were supported by SBF than by MPM. Thus, whenever the participants made a mistake when supported by MPM, they needed more correction keystrokes to repair it.

To analyze the query strategies the participants use, we introduce the notion of initial query and final query. The initial query is the query users try before they check the suggestions given by the autocompletion algorithm. We approximate the initial queries from the logs, by considering the string the user types before he or she presses a correction or control key. The final query is the string typed by the user right before the target appears in the text field. We measure the average length of the initial and the final queries, and compare them to the average total number of keystrokes used to produce the targets. The results are shown in Figure 6, differentiating between the number of words $n$ in the target.

We see that the mean lengths of the initial and final queries increases when the number of words in the target increases. The difference between the mean lengths of the initial and final queries ranges from 1.28 ($n = 3$) to 2.62 ($n = 5$). The difference between the mean final query length and the mean total number of keystrokes required ranges from 5.85 ($n = 2$) to 3.42 ($n = 3$). This difference accounts for the mean number of correction and control keystrokes.

A number of targets turned out to have syntactically highly similar terms in SNOMED CT. Participants found it invariably challenging to detect the incorrect character(s) whenever they had selected a suggestion that was almost similar to the target; e.g., microcytic normochromic anaemia vs. microcytic normochromic anemia and suprapubic urinary catheterisation vs. suprapubic urinary catheterization. These targets typically induced several correction keystrokes. We observed that in the case of such complex targets, the participants, probably trying to focus their attention, occasionally navigated the cursor through the string in the text box while trying to locate a misspelling. Thus more keystrokes were used than there were characters in the target. This explains how the percentage of required keystrokes could exceed 100% and twice even 300%, see participants 5 and 8 in Figure 4.

To investigate the structure of the queries, we differentiate between the number of words $n$ of the targets. We measure the mean length of the $i$th
word in the initial and final queries, where $1 \leq i \leq n$. The results are shown in Figures 7. The mean lengths of the (initial/final) query words do not add up to the corresponding lengths in Figure 7, because the space characters are not accounted for in the former figure.

Figures 7 shows that the participants used the multi-prefix utility, and that they consistently wrote the longer query words in the beginning. I.e., the participants are inclined to write \texttt{opt ner sh m} instead of \texttt{o n s mening} when asked to produce \texttt{optic nerve sheath meningioma}. These findings match our observations during the experiment, with the exception of participant 1 who consistently wrote the first letters of the first $1 \leq i \leq n - 1$ words and the final word until the target appeared in the suggestion list. This strategy obviously fails if the last word is not sufficiently discriminative. For instance, the query \texttt{a e overdose} does not yield the target \texttt{accidental epirubicin overdose} among the first eight suggestions. Consequently, the user cannot pick it from the suggestion list and has to extend the first or second query word to retrieve the target (e.g., \texttt{a epi overdose}), which goes at the cost of pressing \texttt{LEFT} at least nine times.

4.4. Optimal query strategy

In this section we compare the participants’ number of required keystrokes with the length of the optimal query that produces the target. We compute the latter automatically.

For a given target $t$, in the context of the MPM autocompletion algorithm, we defined the optimal query in Section 3.2 as the shortest string $q$ that yields $t$ as first suggestion. This definition gives an upper bound on the least number of keystrokes that is required to produce $t$, as the user can save keystrokes by using navigation keys to select the second, third, etc. suggestion. We saw in Section 3.2 that the problem of computing the optimal query for a given target is NP-hard. For this experiment we developed a brute-force algorithm that essentially iterates through the sequence of all queries, ordered by length.

In the context of SBF we define the optimal query as the shortest sequence of keystrokes that yields the target term. The keystrokes may include control keys, so as to account for the cascaded use of the suggestion list. The cascading strategy was used at least once by participants 1–3 and 5–10. The optimal query can be computed efficiently by means of a shortest path algorithm.

In this analysis, for both algorithms, we assume that the optimal query is produced without the use of correction keys.
Figure 6: Mean initial and final query length compared to the mean total number of keystrokes required to produce a target, differentiating the targets by the number of words they contain.
Figure 7: Mean word length of the initial and final queries. Figures 7(a)–7(d) differentiate between the targets with $2 \leq n \leq 5$ words, respectively.
The distribution of percentage of required keystrokes (with respect to the target length) is shown in Figure 8, comparing the participants’ results with the optimal queries using SBF and MPM. The participants’ boxplots are based on the same data as the histogram in Figure 5 (recall: SBF: mean 67.0%, md 60.0%, sd 30.0%; MPM: mean 52.3%, md 43.4%, sd 36.7%). Not surprisingly, the optimal query strategy outperforms the participants’ behavior in terms of keystrokes (SBF: mean 41.0%, md 39.5%; MPM: mean 23.5%, md 23.0%). Also we observe that the optimal query strategies have less variation than the participants (SBF: sd 10.8%; MPM: sd 4.1%). It is interesting to observe that the median of optimal query strategy for SBF is only slightly better than the median of the participants’ strategy on MPM: 39.5% vs. 43.4%.
4.5. Entropy

In this section we will apply the information-theoretic notion of entropy to the autocompletion task. This will help us understand the workings of the MPM algorithm.

To motivate the information-theoretic view on autocompletion, we observe that in SNOMED CT each term that starts with accidental cut, also starts with accidental cut, puncture, perforation or hemorrhage during (2)

Consequently, each character in the string puncture, perforation or hemorrhage during has no information value as we can predict it from the initial query accidental cut.; or, put differently, when extending accidental cut, not a single SNOMED CT term is eliminated until we reach (2). The MPM algorithm allows the user to skip the low-information characters by jumping to the beginning of the informative words, which reside at the end of the term. For instance, with the help of MPM, ac cu vac yields accidental cut, puncture, perforation or hemorrhage during vaccination.

We can use the information-theoretic notion of entropy to measure the amount of information that is conveyed by the $n$th character in SNOMED CT. According to this notion the entropy of the 20th character given the string accidental cut, pun is 0. The weighted entropy of the $n$th character is basically the entropy of the $n$th character averaged over all SNOMED CT terms of length $n - 1$ or more. See Appendix B for the definition. The weighted entropy of the $n$th character in SNOMED CT is given in Figure 9(a), for $1 \leq n \leq 20$. We see that the weighted entropy decreases sublinearly. Also we see that only the first four characters carry more than 1 bit of information.

This notion of entropy should be contrasted to the weighted average entropy of the first character of the second term word, given the first $n$ characters of the first term word, which is shown in Figure 9(b). We see that the value decreases linearly as $n$ grows.

Comparing Figures 9(a) and 9(b), we conclude that the first character of the second word of a target term has generally higher entropy than the second character of the first term word. However, switching from the first term word to the second goes at the cost of of 1 keystroke, SPACE. From Figures 9(a) and 9(b) we can infer that the optimal point to shift to the second word is after having typed the first 3 characters of the first word. The
weighted average entropy of the fourth character of the first term is namely 1.38 bits, whereas the entropy of the first character of the second term given the first 3 characters of the first word is 3.39 bits. So, compensating for the additional space that is required to shift to the second word, we have that 3.39/2 > 1.38.

5. Discussion

In this section we use the results from the previous section to answer the research questions Q1–Q4 raised in Section 1. Then we discuss directions for future research.

5.1. Research question 1

In our experiment, participants required significantly fewer keystrokes when producing SNOMED CT terms with MPM support than with SBF support, see Section 4.2. This result suggests that the answer to Q1 is affirmative.

The experiment was unbalanced in the sense that all participants first used SBF and then MPM, which is a limitation of our study. We could have balanced the participants by dividing them in four groups instead of two. In addition to the subcollection \{A, B\} used in the first half of the experiment,
each group then would also be characterized by the algorithm \{SBF, MPM\} that supports completion of the first subcollection.

5.2. Research question 2

Our analysis in Section 4.3 showed that the participants in our experiments initially try to retrieve the target term with a multi-prefix queries, in which the words inside each query typically decrease in length. The final query, i.e., the query that retrieves the target term, is generally similar to the initial query modulo extensions of the individual query words. We conclude that the multi-prefix philosophy of the MPM algorithm is recognized and utilized by our participants.

5.3. Research question 3

We showed in Section 4.4 that on average there is a gap of 28.8% \((52.3 - 23.5)\) of required keystrokes between the participants’ and the optimal strategies. In Section 4.3 we saw that 7.6% of these 28.8% are correction keystrokes. (We assumed the optimal strategy does not make misspellings, and therefore does not use correction keys). One part of the remaining 21.2% \((28.8 - 7.6)\) are the keystrokes that need to be corrected; the other part of the 21.2% of keystrokes are due to suboptimal usage of the MPM algorithm. In the extreme case where every correction keystroke repairs 1 misspelling, we have that 13.6% \((21.2 - 7.6)\) of the keystrokes are due to suboptimal usage. This is in fact an unrealistic lower bound on the percentage of “suboptimal keystrokes,” as in reality correcting 1 misspelling requires multiple correction keys (e.g., moving to the position of the misspelled character with \texttt{LEFT} and then removing it with \texttt{BACK}). We conclude that suboptimal usage of MPM accounts for 13.6% to 21.2% of all keystrokes.

The NP-hardness result (Theorem 1) shows that it is computationally very intensive to compute the optimal query for a given target with MPM support. We take it that this result suggests that the optimal query strategy for the MPM algorithm is not within the reach of users. These results are not conclusive though. It may well turn out that there is a near-optimal query strategy that is still efficiently computable.

5.4. Research question 4

A good autocompletion algorithm exploits the patterns of syntactic redundancy in the vocabulary it works on. We can appreciate this point by considering the vocabulary of all strings \(A^{40}\) in the alphabet \(A\) of length 40,
say. Assuming that we have no *a priori* knowledge on the frequency of use of these strings, each string is as likely to be the intended string as the other. Consequently, every character in each string of this vocabulary is highly non-redundant as it eliminates as many strings as there are characters in $A$ minus 1. Autocompleting this vocabulary is unlikely to save many keystrokes: for a query $q$ of length $n$ there are $|A|^{40-n}$ possible suggestions. The odds that the intended string is high enough on the suggestion list is negligible, even for $n = 38$.

In natural language concepts are typically not so densely coded, which leads to redundant syntactic items. For instance, in English the character $q$ is usually followed by the character $u$. One could thus design an autocompletion algorithm that exploits the redundancy of the character following $q$, by suggesting the same string the user has typed plus the character $u$ whenever the string ends on $q$. It is obvious that this autocompletion functionality will not dramatically reduce the average number of required keystrokes.

We applied the information-theoretic notion of entropy to quantify the degree of redundancy of the character position in SNOMED CT terms. We saw that the weighted entropy of the $n$th character decreases sublinearly as $n$ increases. This implies that, roughly speaking, the $n$th character is likely to be redundant given the previous $n-1$ characters. On the other hand we saw that the weighted entropy of the first character of the second word, given the first $n$ characters of the first word, also decreases as $n$ increases, but less rapidly than the weighted entropy of the $n$th character.

These results seem to offer an explanation as to why MPM outperforms SBF on SNOMED CT in the user experiment: The MPM algorithm allows the user to jump ahead to the first letter of the second word, also if he or she has not completed the remainder of the first word. Thus he or she can jump ahead to the high-entropy positions in the target. The SBF algorithm on the other hand, requires the user to type the low-entropy characters of the first word before he or she can move on to the second. The cascading usage of SBF prevents this to a certain extent, but not consistently.

### 5.5. Improvement areas

Section 5.3 gives two leads for future research. The first is handling misspelled queries. Misspellings can be detected by an unexpected drop in the number of suggestions. For instance, MPM gives 263 suggestion for optic $n$ but none for optic nw. Once a misspelling is detected, we can maintain the list of suggestions for the latest correct query. The latter solution does not
require any changes to the MPM algorithm. Alternatively, we ignore the query
word in which the misspelling was detected (e.g., optic nwrve men becomes
optic men), or we can replace the misspelled character by a wildcard (e.g.,
optic nwrve becomes optic n*rve).

The second lead would be to research near-optimal strategies, and guid-
ing users towards these strategies. The latter can be done by making adjust-
ments to the user interface, or by educating users by means of tip-of-the-day
information panels.

Another direction, following feedback of participant 1, is to allow for
instant abbreviations. Participant 1 commented that he would find it intuitive
to retrieve metabolic thrombocytopenic purpura by the query mtp. Note that
SNOMED CT does not use MTP as a synonym for metabolic thrombocytopenic
purpura, nor does Wikipedia list it as a medical abbreviation. Technically
we can accommodate such instant abbreviations by letting the suggestions
of short 1-word queries q be given by the standard suggestions for q (e.g.,
mtp) plus the suggestions for the query that is obtained from q by separating
all characters by spaces (e.g., m t p). We anticipate that this technique will
give too many suggestions in large vocabularies such as SNOMED CT, but it
might prove useful in smaller vocabularies.

Yet another direction for improving autocompletion algorithms is by
building up knowledge about the a priori likelihood that a given term is
the target term. This can be done by, for instance, logging the behavior of
users or by taking contextual parameters into account. The latter approach
was shown successful in [1], where the context was represented by a scor-
ing of the terms in SNOMED CT. The philosophy behind this approach is
that the higher the score of a term, the closer it is to the user’s context and
the more likely it is that this is the user’s intended term. Other “semantic
approaches” to autocompletion focus on disambiguation of queries [11] and
visual groupings of suggestions [12].

6. Conclusions

In this paper we described a word-based prefix matching autocompletion
algorithm, MPM, and compared it to a baseline algorithm, SBF, whose sug-
gestions extend the user’s query strictly to the right. We evaluated the algo-
rithms in a user experiment with 12 participants who were invited to produce

---

4Wikipedia accessed February 4th, 2011
40 SNOMED CT terms with SBF and another 40 with MPM. Our results show that the participants used significantly fewer keystrokes when supported by MPM than by SBF. Our information-theoretic analysis of SNOMED CT sheds some light on this result. It turns out, namely, that MPM allows the user to jump ahead to the beginning of the next word, which has generally higher entropy than the remainder of the first word. Our query analysis showed that participants, when supported by MPM, use multiple query words each of which is not necessarily a proper English word. We outlined four ways to improve the performance of the MPM algorithm.

The keystroke-based analysis allows us to compare the two autocompletion algorithms quantitatively. In its current form each keystroke counts as 1. In a more complicated analysis, we would assign different weights to each key, indicating that some keys are “easier” to use correctly than others. For instance, given its size and position on the keyboard, the space key is probably easier to use correctly than other keys. The order in which keys are used is another aspect that might influence the weight of that key. In the sequence LEFT-LEFT, for example, the second keystroke is “easier” as the user does not have to reposition his or her finger. Consequently, we assign lower weight to the second LEFT than to the first.

The keystroke-based metric used in this paper clearly serves to quantify keystroke savings. As such it explicitly addresses one of the potential benefits of autocompletion, listed in Section 1. It is not evident how the results reported in this paper pertain to the other three benefits mentioned: user experience of the autocompletion algorithm, the extent to which the algorithms help the user to explore their underlying vocabulary, and supporting adoption of standardized vocabulary.

On the one hand, we feel that keystroke savings are positively correlated with the other benefits. For instance, fewer keystrokes are likely to imply that the autocompletion algorithms has a greater reassuring effect on the user, consequently increasing the user experience, which in turn facilitates easier adoption of standardized vocabulary. However, on the other hand, certain applications may call for algorithms that are especially suitable for purposes of exploration, and one may want to redesign one of the described algorithms, or introduce an entirely new algorithm. We believe these are interesting research questions, on the interface of human–computer interaction and computer science.
7. Acknowledgments

The authors gratefully acknowledge Dieter van de Craen and Jan Korst for carefully reviewing an earlier version of this paper. They further extend their gratitude to all participants of the user study, Tim Hulsen and the anonymous referees.

Appendix A. Experiment instructions

In this experiment you are shown 80 medical terms, one at a time, and you are asked to produce them with the keyboard. You will be assisted by two different completion techniques. The first completion technique gives suggestions to the right of what you’re typing. For instance, suppose that the medical term clicking acromioclavicular joint is shown. If we type clicking a the completion algorithm gives us two suggestions:

[Screen shot shown here]

Use the up and down buttons to move the selected suggestion up and down. Do not use the mouse for this! Also don’t use shortcuts (e.g. CTRL+Z) and HOME and END. The selected suggestion is copied to the free text field if we press ENTER. Please familiarize yourself with the system by producing the following medical terms:

• metabolic thrombocytopenic purpura

• autosomal chromosome mosaicism

• mitochondrial cristae fragmentation.

The second completion technique allows you to type only the beginning of the words of the given medical term. For instance, if we type cli ac we get the following eight suggestions:

[Screen shot shown here]

We can freely swap the words in the text box. For instance, ac cli j has precisely one suggestion:

[Screen shot shown here]
Again we can change the selected suggestion with UP and DOWN and we can copy the selected suggestion by pressing ENTER. Please familiarize yourself with the system by producing the three medical terms.

May you have any question do not hesitate to contact the supervisor of the experiment.

Appendix B. Entropy definitions

The entropy of a random variable $X$ is given by

$$H(X) = -\sum_a p(X = a) \log_2 p(X = a),$$

where $a$ ranges over the set of values $X$ can assume and $P(X = a)$ denotes the probability that $X$ assumes value $a$.

In our context, we are interested in the random variable $X$ over the set $A$ of characters appearing in SNOMED CT and the random variable $Y$ over the set $A^n$ of strings over $A$ of length $n$. For a string $s$, $p(Y = s)$ denote the probability that a randomly drawn SNOMED CT term starts with $s$:

$$p(Y = s) = \frac{|\{t \in \text{SNOMED CT} : s \preceq t\}|}{|\text{SNOMED CT}|};$$

we let $p(X = a|Y = s)$ denote the probability that a randomly drawn SNOMED CT term that starts with $s$, starts with $sa$:

$$p(X = a|Y = s) = \frac{|\{t \in \text{SNOMED CT} : sa \preceq t\}|}{|\{t \in \text{SNOMED CT} : s \preceq t\}|}.$$

We define the weighted entropy of the $n$th character as follows:

$$-\sum_{s \in A^n} p(Y = s) \sum_{a \in A} p(X = a|Y = s) \log_2 p(X = a|Y = s).$$

As an example consider the case where $s = \text{accidental cut, pun}$. We know that the next character will be $c$, and therefore the information value of the 20th character, given $s$, is 0. We compute:

$$p(X = c|Y = s) = 1$$

$$p(X = c|Y = s) \log_2 p(X = c|Y = s) = 0.$$
The notion of entropy relevant for the MPM algorithm should be capable of distinguishing word boundaries. We will not develop the full version, instead, we will content ourselves with the notion that quantifies the information of the first character in the second word given the first $n$ characters of the first word. Let $B = A - \{\_\}$ be the set of characters appearing in SNOMED CT excluding the space character.

For, $s \in B^n$, let $\tilde{p}(X = a|Y = s)$ be the probability that the first character of the second word of a randomly drawn SNOMED CT term starting with $s$ is $a$:

$$\tilde{p}(X = a|Y = s) = \frac{|\{(t_1, \ldots, t_n) \in \text{SNOMED CT} : s \preceq t_1, a \preceq t_2\}|}{|\{(t_1, \ldots, t_n) \in \text{SNOMED CT} : s \preceq t_1\}|}.$$

We define the weighted average entropy of the first character of the second term word, given the first $n$ characters of the first term word as

$$- \sum_{s \in B^n} p(Y = s) \sum_{a \in B} \tilde{p}(X = a|Y = s) \log_2 \tilde{p}(X = a|Y = s).$$

References


