The Shortest Kinship Description Problem*

Chao Xu† Qian Zhang‡

Abstract

We consider a problem in descriptive kinship systems, namely finding the shortest sequence of terms that describes the kinship between a person and his/her relatives. The problem reduces to finding the minimum weight path in a labeled graph where the label of the path comes from a regular language. The running time of the algorithm is $O(n^3 + s)$, where n and s are the input size and the output size of the algorithm, respectively.

To the memories of Jiaqi Zhao(1994-2016).

1 Introduction

Consider a person who is inexperienced with (consanguineous) kin terms in a particular language (e.g. Chinese). She tries to form a description of her relation to a kin. A desirable description should be concise, and only use terms she knows. This captures the shortest kinship description problem (**SKDP**). The scenario sounds like a small hurdle exclusive to new language learners. But in reality, this can be a frustrating problem even for native speakers. The Chinese kinship terminology is very complicated. There are few applications built to address the complication. LessLoop Limited developed an application to calculate the correct Chinese kin term. The application gathered over 100,000 downloads and a wide media coverage [1]. Mi Calculator implemented a similar feature before Chinese New Year 2017. It helped users with addressing their visiting relatives in a correct manner [2]. Yet, both applications give up when the relationship is too complicated: neither application can report a description involving more than *one* term. For example, "my maternal granddaughter's maternal granddaughter" is a relation that cannot be described with a single Chinese term. This prompts the investigation in this paper.

A *kin type* is an abstract concept of the kinship between two people. The concept of mother is a kin type. A monoid is an algebraic structure with a binary associative operation, called product, and an identity element. The kinship monoid is an algebraic model of kin types. The monoid has four primitives: `f`, `m`, `s` and `d` representing father, mother, son and daughter, respectively (we will use typewriter font to indicate the 4 elements). The product of the elements represents the composition of the relations. For example, `fsm` would be "(ego's) father's son's mother". Some products represent the same kin type. For example, `fs` and `ms` both represent brother. A *kin term* is a name people use to refer to a kin type. For instance, "paternal grandfather" for `ff`, "brother" for `fs` and "wife" for `sm`. Kin terms depend on the language, dialect, and culture [14]. The kin terms also compose by taking product of their representing kin types. The goal is to have a concise expression of a kin type through composing available kin terms.

Related work Modeling aspects of kin types and kin terms as a monoid has a long history (See [12] for a historical overview). Murdock used a similar formalism of the kinship monoid, but had four extra primitives [10]. Morgan outlined six major kinship systems [9]. Our algorithm handles the Sudanese system, the most complicated one. For non-Sudanese patterns, Boyd analyzed Arunta, Kariera and Ambrym kinship algebraically [6]. Read analyzed the American kinship and found the underlying space is $\mathbb{Z}^2$ [12].

For a monoid M, a subset S and $x \in M$, the *submonoid membership problem* asks for a product of some elements in S that equals x. The optimization version minimizes the number of elements in the product. It is called the *submonoid membership optimization problem* (**SMOP**). The shortest kinship description problem is a special case of **SMOP**. In general, **SMOP** is undecidable even for simple algebraic structures [11]. Hence, it is unclear whether an algorithm for the shortest kinship description problem exists.

*Chao is supported in part as a State Farm Companies Foundation Doctoral Scholar.

†Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, IL 61801 chaoxu3@illinois.edu

‡School of Civil and Environmental Engineering, Cornell University, Ithaca, NY 14853 qz283@cornell.edu

Our contribution We give the first algorithm that runs in polynomial time with respect to the input size and linear time with respect to the output size. It is also the first algorithm that can handle the case where the output is more than a single term.

2 Preliminaries

2.1 Rewriting systems

We assume standard knowledge on regular and context-free languages. For an introduction, see [13]. We adapt the notations for rewriting systems in [5]. We review a few standard notions. Let X be a set of strings, X^* is the *Kleene star* operation: the smallest superset of X that contains the empty string and is closed under string concatenation. If X is set of symbols, X^* is the set of all strings using symbols in X. For a non-terminal A in a context-free grammar, $\mathbb{L}(A)$ is the set of all strings generated by A. For a regular expression R, $\mathbb{L}(R)$ is the set of strings matched by R. A context-free grammar is a *Chomsky normal form* if all rules are of the form $A \to BC$ or $A \to a$ for some non-terminals A, B and C, and terminal a. Each context-free language has a context-free grammar in Chomsky normal form. A *(string) rewriting system* R on X is a relation on X^*. A *(rewriting) rule* is an element in a rewriting system. Instead of (a, b), we write $a \to b$ to emphasis it is a rule. If $a \to b \in R$, $w = uav$, and $w' = ubv$ for some strings u and v, we write $w \to_R w'$. w' is the result after *applying* rule $a \to b$ to w. $w \leftrightarrow_R w'$ if either $w \to_R w'$ or $w' \to_R w$. $w \xrightarrow{*}_R w'$ if $w = w'$ or there exists w'' such that $w \to_R w''$ and $w'' \xrightarrow{*}_R w'$. w *derives* w' if $w \xrightarrow{*}_R w'$. $w \leftrightarrow^*_R w'$ if $w = w'$ or there exists w'' such that $w \leftrightarrow_R w''$ and $w'' \leftrightarrow^*_R w'$. $\leftrightarrow^*_R$ is an equivalence relation, and we denote the equivalence class containing w as $[w]_R = \left\{w' \mid w' \leftrightarrow^*_R w\right\}$.

A string w is R *normal* with respect to a rewriting system R, if $w \xrightarrow{*}_R w'$ implies $w = w'$. A normal string y is called a *normal form* of a string x if $x \xrightarrow{*}_R y$. R is *convergent* if every string has a unique normal form. The unique normal form of w is denoted as $\underline{w}$ when there is no ambiguity about the rewriting system. A rule $a \to b$ is *length reducing* if $|a| > |b|$, *length preserving* if $|a| = |b|$, and *length non-increasing* if it is either length reducing or length preserving.

For a rewriting system R on alphabet X and $L \subseteq X^*$, the set of strings can be derived from some string in L is $\Delta^*_R(L) = \left\{w' | w \in L, w \xrightarrow{*}_R w'\right\}$. The set of strings that derive some string in L is $\nabla^*_R(L) = \left\{w | w' \in L, w \xrightarrow{*}_R w'\right\}$. If R is a convergent rewriting system, $w \leftrightarrow^*_R w'$ if and only if $\underline{w} = \underline{w'}$ [3, Theorem 2.1.9].

2.2 The kinship monoid and problem definition

A presentation is an ordered pair $\langle X \mid R\rangle$. X is the *generating set* or *alphabet*, and $R \subseteq X^* \times X^*$ are the *relators*. We write $S = \langle X \mid R\rangle$ if S is a monoid isomorphic to $X^*/\leftrightarrow^*_R$ (i.e. the monoid on $\{[x]_R | x \in X^*\}$ where the product is defined as $[x]_R[y]_R = [xy]_R$). For more introduction on presentation, see [7].

We proceed to define the kinship monoid. The alphabet of the kinship monoid is $\Sigma = \{\mathtt{f}, \mathtt{m}, \mathtt{s}, \mathtt{d}\}$. A *word* is another name for strings in Σ^*. A sign string for a word $w \in \Sigma^*$, denoted $\mathrm{sgn}(w)$, is the string obtained by replacing each `f` and `m` with `+`, and each `s` and `d` with `-`.

We define a set of relators Γ. For all $a, b, c \in \Sigma$,

1. (abc, c) is in Γ if $\mathrm{sgn}(abc) \in \{\mathtt{+-+}, \mathtt{-+-}\}$, (generation cancellation)
2. (abc, a) is in Γ if $\mathrm{sgn}(abc) = \mathtt{--+}$, and $\{a, c\} = \{\mathtt{s}, \mathtt{f}\}$ or $\{a, c\} = \{\mathtt{d}, \mathtt{m}\}$, (Child's child's parent is child if gender agrees)
3. (ac, bc) is in Γ if $\mathrm{sgn}(ac) = \mathrm{sgn}(bc) \in \{\mathtt{-+}, \mathtt{+-}\}$. (siblings/spouse)

Γ is also a rewriting system on Σ. We will use $[w]$ to denote $[w]_\Gamma$. The *kinship monoid* is $K = \langle \Sigma \mid \Gamma\rangle$.

Problem 1 (SMOP) *Let M be a monoid, given $X \subseteq M$ and $y \in M$ with cost function $\$: X \to \mathbb{N}$. Find $x_1, \ldots, x_n \in X$ such that $y = x_1 \cdot \ldots \cdot x_n$ and $\sum_i \$(x_i)$ is minimized, or return no solution exists.*

The shortest kinship description problem is **SMOP** on K, the kinship monoid. X is the kin type of the allowed kin terms. Each kin type $x \in K$ is given as an element in $w \in \Sigma^*$, such that x is the product of w, when w is seen as a sequence of elements in K. We also want a concise output. Instead of outputting the sequence of elements in X, we order the element in X, and output the sequence of indices. Hence we obtain the *shortest kinship description problem (***SKDP***)*.

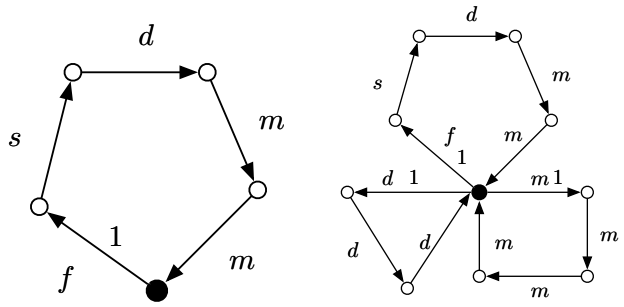


Figure 2.1: On the left is C_w, where $w = \texttt{fsdmm}$ and cost of w is 1. On the right is $G_{W,\$}$ where $W = \{\texttt{fsdmm}, \texttt{ddd}, \texttt{mmmm}\}$, and the cost of each word is 1. The black vertex is the special start vertex q. Edges without weight label have weight 0.

Problem 2 (Shortest Kinship Description Problem) *Let $K = \langle \Sigma \mid \Gamma \rangle$ be the kinship monoid. $W = w_1, \ldots, w_k$ is a sequence of elements in Σ^* and $w \in \Sigma^*$ is the target element. The input also contains a cost function $\$: W \to \mathbb{N}$. Find a sequence $a_1, \ldots, a_s$ such that $[w] = [w_{a_1} \ldots w_{a_s}]$ and $\sum_i \$(w_{a_i})$ is minimized, or return no solution exists.*

For a cost function $\$: W \to \mathbb{N}$, we extend it to $\$^* : W^* \to \mathbb{N}$ as follows.

$$\$^*(w) = \min_{\substack{w = w_1 \ldots w_n \\ w_i \in W}} \sum_{i=1}^{n} \$(w_i)$$

Another way of phrasing Problem 2 is finding a word $w' \in W^* \cap [w]$ that minimizes $\$^*(w')$.

2.3 Labeled graphs and algorithms

A weighted labeled graph is a graph $G = (V, E)$ with a label function $\ell : E \to \Sigma$ and a weight function $\gamma : E \to \mathbb{N}$. Let $P = e_1, \ldots, e_k$ be a path. The label and weight extend to paths. $\ell(P) = \ell(e_1) \ldots \ell(e_k)$ is the concatenation of all the labels following the path. $\gamma(P) = \sum_{i=1}^{k} \gamma(e_i)$ is the total weight of the edges in the path. The length of a path is the number of edges in the path.

Consider a weighted labeled graph $G = (V, E)$ and a context-free grammar N. For $u, v \in V$ and A a non-terminal in N, we define $P_{G,N}(u, v, A)$ to be a minimum weight uv-path such that its label is an element in $\mathbb{L}(A)$ (there might be multiple paths satisfying this property, ties are broken arbitrarily). $D_{G,N}(u, v, A)$ is the weight of $P_{G,N}(u, v, A)$. $D_{G,N}(u, v, A) = \infty$ if such path does not exist.

Theorem 2.1 ([4]) *Given a weighted labeled graph G with n vertices and a context-free grammar N in Chomsky normal form. N consists of t non-terminals and r rules. It takes $O(n^3 rt)$ time to find a data structure such that the following queries take $O(1)$ time. Given u, v, A,*

1. *return the value of $D_{G,N}(u, v, A)$.*
2. *If $D_{G,N}(u, v, A) = D_{G,N}(u, x, B) + D_{G,N}(x, v, C)$ for some non-terminal B and C, return the rule $A \to BC$ and x.*
3. *If $D_{G,N}(u, v, A) = D_{G,N}(u, v, a)$ for some terminal a, return the rule $A \to a$.*

Theorem 2.2 ([4]) *Given a weighted labeled graph G with n vertices and a regular expression R of length r. It takes $O(rn^2)$ time to compute the minimum weight st-path in G with labels in $\mathbb{L}(R)$.*

3 Rewriting system for Kinship monoid

We construct a convergent rewriting system so all words in $[w]$ have the same normal form.

Let R be all length reducing rules in Γ, and S be length preserving rules $\{\texttt{fd} \to \texttt{md}, \texttt{fs} \to \texttt{ms}, \texttt{sf} \to \texttt{df}, \texttt{sm} \to \texttt{dm}\}$. The desired rewriting system is $T = R \cup S$. The symmetric closure of T is Γ, so $w \leftrightarrow_\Gamma^* w'$ if and only if $w \leftrightarrow_T^* w'$. Additionally, let R_1 consists of all the $abc \to c$ rules in R, and R_2 consists of all the $abc \to a$ rules in R.

Theorem 3.1 *T is a convergent rewriting system.*

Proof: Apply the Knuth–Bendix completion algorithm on T [8]. The output shows T is convergent. Alternatively, we have a tedious proof in Appendix A. □

From this point on, $\underline{w}$ is the unique normal form of w under T. In particular, we now have that $[w] = [w']$ iff $\underline{w} = \underline{w'}$. In other words, $[w] = \nabla_T^*(\underline{w})$, the set of all words that can derive $\underline{w}$. We can compute the normal form $\underline{w}$ in $O(|w|)$ time by successively applying the rules in T.

Next, we prove some basic properties of the rewriting system. We can first apply only rules in R until we get a R normal form, and then apply S rules to reach a T normal form. In other words, $\nabla_T^*(\underline{w}) = \nabla_R^*(\nabla_S^*(\underline{w}))$. First, we require a lemma.

Lemma 3.2 *w is a R normal form iff $w \xrightarrow{*}_S \underline{w}$.*

Proof: One direction is easy. $w \xrightarrow{*}_S \underline{w}$, then $|w| = |\underline{w}|$. Suppose we can apply a R rule to w and obtain w'. $\underline{w}$ is not a T normal form of w', because all rules in T are length non-increasing and $|w'| < |\underline{w}|$. A contradiction to $\underline{w}$ as the unique T normal form of w.

For the other direction, we show that if w is a R normal form, and $w \to_S w'$, then w' is a R normal form. Indeed, rules in S do not change the sign string. A rule in R_1 can be applied to w, then it can be applied w' because rules in R_1 only depend on sign string. Therefore no rule in R_1 can be applied to w. We now show no rules in R_2 can be applied to w either. Assume $w = xaby$, and $w' = xa'by$. If we can apply a R_2 rule to w', this means we have to apply a rule of the form $a'bc \to c$ or $uva' \to a'$ for some $u, v, c \in \Sigma$. The sign string of $a'b$ is either `-+` or `+-`, so $a'bc \to c$ is not in R_2. The sign string of $uva'b$ would be `--+-`. Since `-+-` cannot be a substring of the sign string in w, we cannot apply rules of the form $uva' \to b$. Hence w' is R normal form. □

Theorem 3.3 *For any $w \in \Sigma^*$, $\nabla_T^*(\underline{w}) = \nabla_R^*(\nabla_S^*(\underline{w}))$.*

Proof: $\nabla_R^*(\nabla_S^*(\underline{w})) \subseteq \nabla_T^*(\underline{w})$ is clear. For the other direction, consider a string in $w' \in \nabla_T^*(\underline{w})$. Let w'' be a R normal form of w', hence $w' \in \nabla_R^*(w'')$. There exists a S normal form of w'', say w'''. Hence $w' \in \nabla_R^*(\nabla_S^*(w'''))$. w''' is also a R normal form by Lemma 3.2. Hence $w''' = \underline{w}$ is a T normal form. This shows that $w' \in \nabla_R^*(\nabla_S^*(\underline{w}))$. □

4 A polynomial time algorithm for shortest kinship description problem

In this section, we prove the main theorem.

Theorem 4.1 (Main Theorem) *Let n be the total length of the input for the shortest kinship description problem. There is an algorithm that returns a solution in $O(n^3 + s)$ time, where s is the output length.*

We outline our high level approach. Recall in **SKDP**, the input is a word w, a sequence of words W and a cost function $\$: W \to \mathbb{N}$. We seek a (concise description of a) minimum cost word in $[w] \cap W^*$. Our goal is to reduce **SKDP** to a minimum weight path problem with its label in a regular language, so we can apply Theorem 2.2. **SKDP** is equivalent to find a minimum cost word in $\nabla_S^*(\underline{w}) \cap \Delta_R^*(W^*)$ for an appropriate extension of the cost function to $\Delta_R^*(W^*)$. The cost function over $\Delta_R^*(W^*)$ can be modeled as a graph G'. A short regular expression describes $\nabla_S^*(\underline{w})$. The problem reduces to finding a path with its label in $\nabla_S^*(\underline{w})$ on G', and we can apply Theorem 2.2 as desired.

For a word $u \in W$, a labeled graph C_u is a directed cycle with a special start vertex q. The label following q around the cycle C_u is u. For each $u \in W$, construct C_u and assign the outgoing edge from q with weight $\$(u)$, and all other edges with weight 0. Identify all C_u's by the start vertex. Call the resulting weighted labeled directed graph $G_{W,\$}$ with label function ℓ and weight γ. See Figure 2.1 for example. Any qq-path in the graph has its label in W^*. $G_{W,\$}$ have the property $\$^*(\ell(P)) = \gamma(P)$ for every qq-path P. A minimum weight qq-path in $G_{W,\$}$ with label in $[w]$ gives us the desired minimum cost word in $[w] \cap W^*$.

However, output the minimum weight qq-path in $G_{W,\$}$ is not our goal. We want to find the sequence $a_1, \ldots, a_s$ such that $w_{a_1} \ldots w_{a_s}$ is the label of the qq-path. It is easy to add auxiliary information on the edges to return the sequence instead. Indeed, when we construct $G_{W,\$}$, we also add an *output function* π to the edges. For exactly one edge e in C_{w_i}, $\pi(e) = i$, and $\pi(e)$ is the empty sequence everywhere else. The output of a path $\pi(P)$ is the concatenation of the output of the edges. For any qq-path P, the output forms the desired index sequence $a_1, \ldots, a_n$ such that $\ell(P) = w_{a_1} \ldots w_{a_n}$.

Input: $w, W, \$$
  $G \leftarrow G_{\$,W}$
  Compute the data structure for $D_{G,N'}$
  Compute G'
  $P \leftarrow$ min weight qq-path in G' with its label in $\nabla_S^*(\underline{w})$
  if P does not exist
    return `NO SOLUTION`
  for each edge $e = (u, v)$ with label a in P:
    $A_e \leftarrow \pi(P_{G,N'}(u, v, S'_a))$
  return concatenation of all A_e following the path P

Figure 4.1: The algorithm

Instead of finding a minimum cost word in $W^* \cap \nabla_T^*(\underline{w})$, we can seek a minimum cost word in $\Delta_R^*(W^*) \cap \nabla_S^*(\underline{w})$. We extend the cost function $\$^*$ from W^* to $\Delta_R^*(W^*)$. Let $\$_R^* : \Delta_R^*(W^*) \to \mathbb{N}$ defined as follows,

$$\$_R^*(w) = \min_{\substack{w' \xrightarrow{*}_R w \\ w' \in W^*}} \$^*(w')$$

Theorem 4.2 *For every $w \in \Sigma^*$, $W \subseteq \Sigma^*$ and cost function $\$^* : W^* \to \mathbb{N}$,*

$$\begin{aligned} &\min\left\{\$^*(w') \mid w' \in W^* \cap \nabla_T^*(\underline{w})\right\} \\ =&\min\left\{\$_R^*(w') | w' \in \Delta_R^*(W^*) \cap \nabla_S^*(\underline{w})\right\} \end{aligned}$$

Proof: Let $a = \min\left\{\$^*(w') \mid w' \in W^* \cap \nabla_T^*(\underline{w})\right\}$ and $b = \min\left\{\$_R^*(w') | w' \in \Delta_R^*(W^*) \cap \nabla_S^*(\underline{w})\right\}$. For every $w' \in W^* \cap \nabla_T^*(\underline{w})$, there exists $w'' \in \Delta^*(w') \cap \nabla_S^*(\underline{w})$. Hence $\$_R^*(w'') \le \$^*(w')$. Therefore $a \ge b$. On the other hand, for every $w' \in \Delta_R^*(W^*) \cap \nabla_S^*(\underline{w})$, there exists $w'' \in W^*$ such that $w'' \in \nabla_R^*(w')$ and $\$^*(w') = \$_R^*(w'')$. By Theorem 3.3, $\nabla_T^*(\underline{w}) = \nabla_R^*(\nabla_S^*(\underline{w}))$. We have $w'' \in W^* \cap \nabla_T^*(\underline{w})$. Therefore $a \le b$. □

Let G' be a graph such that the label of qq-paths forms $\Delta_R^*(W^*)$, and the minimum qq-path with label w has cost equals $\$_R^*(w)$. Finding a minimum qq-path in G' with its label in $\nabla_S^*(\underline{w})$ would solve the problem. The existence of such graph follows from a general construction in [11], where we shall describe here. Let $G = G_{W,\$}$. Consider the context-free grammar N define as follows: For every rule $a_1a_2a_3 \to a \in R$, there is a rule $S_a \to S_{a_1}S_{a_2}S_{a_3}$. For every S_a, there is a rule $S_a \to a$. The rules are defined so $\mathbb{L}(S_a) = \nabla_R^*(a)$. N is not a Chomsky normal form, but there exists a Chomsky normal form N' with non-terminals S'_a for each S_a, such that $\mathbb{L}(S'_a) = \mathbb{L}(S_a)$. We define the graph G' as follows: the vertices are the vertices in G, for each $u, v \in V$ and $a \in \Sigma$, if $D_{G,N'}(u, v, S'_a) \ne \infty$, then there is an edge uv with label a and cost $D_{G,N'}(u, v, S'_a)$.

Next, we establish $\nabla_S^*(\underline{w})$ can be matched by a regular expression of length $O(|\underline{w}|)$. No rule in S changes the sign string. Because `-+-` and `+-+` are not substring of the sign string of a R normal word, the positions where one can apply S rules are all disjoint. Hence we have the following lemma.

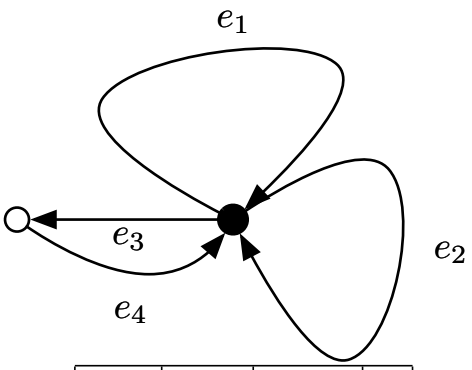
Lemma 4.3 *For some n, $\underline{w} = u_1v_1u_2 \dots v_nu_{n+1}$ such that for each i, $\mathrm{sgn}(u_i)$ consist of the same sign, $|v_i| = 2$, and $w_i \to v_i \in S$ for some w_i. For any element $w' \in \nabla_S^*(\underline{w})$, it is of the form $w' = u_1v'_1u_2 \dots v'_nu_{n+1}$, where $v'_i \in \{v_i, w_i\}$ for all $1 \le i \le n$.*

By Lemma 4.3, it is easy to construct a regular expression of $O(|\underline{w}|)$ length that matches $\nabla_S^*(\underline{w})$ in linear time. First find $\underline{w}$ from w in $O(|w|)$ time. If $\underline{w} = u_1v_1u_2 \dots v_nu_{n+1}$ as in Lemma 4.3, the regular expression is simply

$$u_1(v_1|w_1)u_2 \dots (v_n|w_n)u_{n+1}.$$

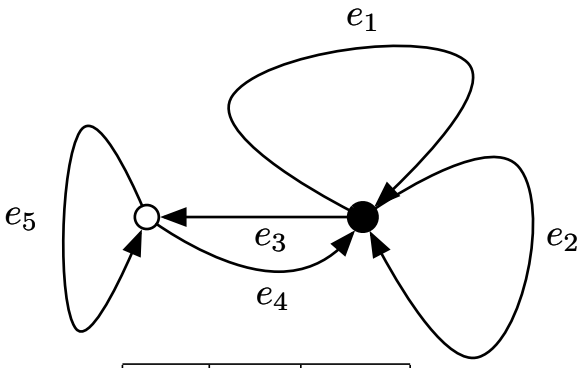
4.1 The algorithm

Figure 4.1 summarizes the algorithm. The rest of the paper analyzes its complexity. Let n be the input size. Computing $G = G_{\$,W}$ can be done in $O(n)$ time. G is a graph with $O(n)$ edges and vertices. Computing the data structure for $D_{G,N'}$ takes $O(n^3)$ time by Theorem 2.1 because N' has constant size. Computing G' takes $O(n^2)$ time, as there are only $O(n^2)$ edges. $\nabla_S^*(\underline{w})$ is a regular language that can be represented with a regular expression of



Edge	Label	Weight	π
e_1	d	3	1
e_2	m	1	2
e_3	d	1	3
e_4	d	0	

Figure 5.1: The example G in Section 5. The black vertex is q.



Edge	Label	Weight
e_1	d	2
e_2	m	1
e_3	d	1
e_4	d	0
e_5	d	2

Figure 5.2: The example G' in Section 5. Note the additional edge e_5. The black vertex is q.

length $O(|\underline{w}|)$. Finding a minimum weight qq-path in G' with its label in $\nabla_S^*(\underline{w})$ takes $O(|\underline{w}|n^2) = O(n^3)$ time by Theorem 2.2. The minimum weight path have $O(n)$ edges because all elements in $\nabla_S^*(\underline{w})$ has length $|\underline{w}| = O(n)$. Hence, the loop is executed $O(n)$ times. Inside each iteration of the loop, we proceed by dynamic programming to compute output A_e. If $P_{G,N'}(u,v,A) = P_{G,N'}(u,x,B)P_{G,N'}(x,v,C)$, we find $\pi(P_{G,N'}(u,v,A))$ by concatenating $\pi(P_{G,N'}(u,x,B))$ and $\pi(P_{G,N'}(x,v,C))$. To determine the x, B and C, we query the data structure for $D_{G,N'}$ once. For each distinct $P_{G,N'}(u,v,A)$, there is only one query. The dynamic programming algorithm queries the data structure $O(n^2)$ times, and each query takes $O(1)$ time by Theorem 2.1. Outputting A_e takes an additional $O(|A_e|)$ time. Together, the total running time is $O(n^3 + s)$.

We have shown the running time of the algorithm is polynomial to the input size and linear to the output size. It is open if the output size is bounded by a polynomial of the input size.

5 An example

Consider the following input for the problem. We have $W = \{w_1, w_2, w_3\}$, where $w_1 = \mathtt{d}$, $w_2 = \mathtt{m}$ and $w_3 = \mathtt{dd}$. The cost is defined as $\$(w_1) = 3$ and $\$(w_2) = \$(w_3) = 1$. We are interested in finding the minimum cost solution to $w = \mathtt{ddd}$. First, we compute $G = G_{\$,W}$, as shown in Figure 5.1. Now, we compute G'. Note this would update the edge weight and also create an new edge. See Figure 5.2. As a demonstration, we consider how edge e_1 is created. The label of e_1 is $\mathtt{d}$. The weight is the minimum over all paths in G that goes from q to itself, and with label in $\nabla_R^*(\mathtt{d})$. In particular, the path $e_3e_4e_2$ in G has label $\mathtt{ddm}$, which is the same as $\mathtt{d}$. It has weight 2, and no other path with label in $\nabla_R^*(\mathtt{d})$ has smaller weight. Let the vertex incident to e_5 be v. e_5 was created because there is a vv-path $e_4e_2e_3$ in G with label $\mathtt{dmd}$. Finally, we compute a qq-path in G' with minimum weight and has label $\mathtt{ddd}$. One possible minimum weight path in G' is $e_1e_3e_4$. It reflects a path in G, namely $P = e_3e_4e_2e_3e_4$. The output corresponds to this path is $\pi(P)$, which is $3, 2, 3$. Thus, the desired solution is $w = w_3w_2w_3$.

Acknowledgement

The authors would like to thank Alan J. Cain for helpful pointers to monadic rewriting systems.

References

[1] 三姑六婆 - 親戚稱呼計算機. https://www.relativescalc.com/. Accessed: 2017-09-04.

[2] 计算器-小米应用商店. http://app.mi.com/details?id=com.miui.calculator. Accessed: 2017-09-04.

[3] F. Baader and T. Nipkow. *Term Rewriting and All That*. Cambridge University Press, New York, NY, USA, 1998.

[4] C. Barrett, R. Jacob, and M. Marathe. Formal-language-constrained path problems. *SIAM J. Comput.*, 30(3):809–837, May 2000.

[5] R. V. Book and F. Otto. *String-Rewriting Systems (Monographs in Computer Science)*. Springer, 1993.

[6] J. P. Boyd. The algebra of group kinship. *Journal of Mathematical Psychology*, 6(1):139 – 167, 1969.

[7] J. M. Howie. *Fundamentals of Semigroup Theory (London Mathematical Society Monographs)*. Clarendon Press, 1996.

[8] D. E. Knuth and P. B. Bendix. *Simple Word Problems in Universal Algebras*, pages 342–376. Springer Berlin Heidelberg, Berlin, Heidelberg, 1983.

[9] L. Morgan. *Systems of Consanguinity and Affinity of the Human Family*. Bison book. University of Nebraska Press, 1870.

[10] G. Murdock. *Social Structure*. A Free press paperback: Sociology. Macmillan Company, 1949.

[11] A. Myasnikov, A. Nikolaev, and A. Ushakov. Knapsack problems in groups. *Mathematics of Computation*, 84(292):987–1016, 2015.

[12] D. W. Read, J. Atkins, I. R. Buchler, M. Fischer, G. D. Meur, E. Lally, B. Holbrook, D. B. Kronenfeld, H. W. Scheffler, S. Seidman, and W. D. Wilder. An algebraic account of the american kinship terminology [and comments and reply]. *Current Anthropology*, 25(4):417–449, 1984.

[13] M. Sipser. *Introduction to the Theory of Computation*. Cengage Learning, 3rd edition, 2012.

[14] A. F. C. Wallace and J. Atkins. The meaning of kinship terms. *American Anthropologist*, 62(1):58–80, 1960.

A Proof of Theorem 3.1

A length non-increasing rewriting system T is convergent if T is locally confluent [3]. That is, if $a \to_T b$ and $a \to_T c$, then there exists some z such that $b \xrightarrow{*}_T z$ and $c \xrightarrow{*}_T z$. We consider all minimal substrings with overlapping rules, and show it is locally confluent.

Overlapping rules both in R_1

Case 1 For a word $a_1a_2a_3a_4$ where R_1 rules applies to both $a_1a_2a_3$ and $a_2a_3a_4$.

- $(a_1a_2a_3)a_4 \to_{R_1} a_3a_4$
- $a_1(a_2a_3a_4) \to_{R_1} a_1a_4$

Resulting pairs both have sign string +- (or -+). Applying a rule in S to one of the two strings results the same word.

Case 2 For a word $a_1a_2a_3a_4a_5$ where R_1 rules applies to $a_1a_2a_3$ and $a_3a_4a_5$.

- $(a_1a_2a_3)a_4a_5 \to_{R_1} a_3a_4a_5 \to_{R_1} a_5$
- $a_1a_2(a_3a_4a_5) \to_{R_1} a_1a_4a_5 \to_{R_1} a_5$

Overlapping rules both in R_2 It is impossible for both overlapping rules to be in R_2, since there is no sign string of length ≤ 5 where - - + appears twice.

Overlapping rules both in S

Case 3 For a word $a_1a_2a_3$ where S rules apply to both a_1a_2 and a_2a_3.

- $(a_1a_2)a_3 \to_S a_1'(a_2a_3) \to_S a_1'a_2'a_3$
- $a_1(a_2a_3) \to_S (a_1a_2')a_3 \to_S a_1'a_2'a_3$

Overlapping rules in R_1 and R_2

Case 4 For a word $a_1a_2a_3a_4$ where a R_1 rule applies to $a_2a_3a_4$ and a R_2 rule applies to $a_1a_2a_3$.

- $a_1(a_2a_3a_4) \to_{R_2} a_1a_4$
- $(a_1a_2a_3)a_4 \to_{R_1} a_1a_4$

Case 5 For a word $a_1a_2a_3a_4a_5$ where a R_1 rule applies to $a_1a_2a_3$ and a R_2 rule applies to $a_3a_4a_5$. (so sign string -+--+)

- $(a_1a_2a_3)a_4a_5 \to_{R_1} a_3a_4a_5 \to_{R_2} a_3$
- $a_1a_2(a_3a_4a_5) \to_{R_2} a_1a_2a_3 \to_{R_1} a_3$

Case 6 For a word $a_1a_2a_3a_4a_5$ where a R_1 rule applies to $a_3a_4a_5$ and a R_2 rule applies to $a_1a_2a_3$. (so sign string --+-+)

- $(a_1a_2a_3)a_4a_5 \to_{R_2} a_1a_4a_5$
- $a_1a_2(a_3a_4a_5) \to_{R_1} a_1a_2a_5$

We can apply a S rule to either a_2a_5 or a_4a_5 to obtain the same word.

Overlapping rules in R_1 and S

Case 7 For a word $a_1a_2a_3$ where a R_1 rule applies to $a_1a_2a_3$. A S rule applies to a_1a_2 or a_2a_3.

- $a_1a_2a_3 \to_{R_1} a_3$
- $(a_1a_2)a_3 \to_S a_1'a_2a_3 \to_{R_1} a_3$
- $a_1(a_2a_3) \to_S a_1a_2'a_3 \to_{R_1} a_3$

Case 8 For a word $a_1a_2a_3a_4$ where a R_1 rule applies to $a_1a_2a_3$ and a S rule applies to a_3a_4.

- $(a_1a_2a_3)a_4 \to_{R_1} a_3a_4 \to_S a_3'a_4$
- $a_1a_2(a_3a_4) \to_S (a_1a_2a_3')a_4 \to_{R_1} a_3'a_4$

Case 9 For a word $a_1a_2a_3a_4$ where a R_1 rule applies to $a_2a_3a_4$ and a S rule applies to a_1a_2.

- $a_1(a_2a_3a_4) \to_{R_1} a_1a_4 \to_S a_1'a_4$
- $(a_1a_2)a_3a_4 \to_S a_1'(a_2a_3a_4) \to_{R_1} a_1'a_4$

Overlapping rules in R_2 and S

Case 10 For a word $a_1a_2a_3a_4$ with sign string +--+, a R_2 rule applies to $a_2a_3a_4$ and a S rule applies to a_1a_2.

- $a_1(a_2a_3a_4) \to_{R_2} a_1a_2 \to_S a_1'a_2$
- $(a_1a_2)a_3a_4 \to_S a_1'(a_2a_3a_4) \to_{R_2} a_1'a_2$

Case 11 For a word $a_1a_2a_3a_4$ with sign string --+-, a R_2 rule applies to $a_1a_2a_3$ and a S rule applies to a_3a_4.

- $(a_1a_2a_3)a_4 \to_{R_2} a_1a_4$
- $a_1a_2(a_3a_4) \to_S (a_1a_2a_3')a_4 \to_{R_2} a_1a_4$

Case 12 For a word $a_1a_2a_3$ with sign string --+, a R_2 rule applies to $a_1a_2a_3$ and a S rule applies to a_2a_3.

- $a_1a_2a_3 \to_{R_2} a_1$
- $a_1(a_2a_3) \to_S a_1a_2'a_3 \to_{R_2} a_1$