Intersection Optimization is NP-Complete

Guillaume Bonfante[1] and Joseph Le Roux[2]

[1]INRIA – LORIA
[2]Nancy Universités – LORIA

July 29, 2007

Abstract

Finite state methods for natural language processing often require the construction and the intersection of several automata. In this paper we investigate the question of determining the best order in which these intersections should be performed. We take as an example lexical disambiguation in polarity grammars. We show that there is no efficient way to minimize the state complexity of these intersections.

1 Introduction

The main concern of this paper is to answer the following question: given a set $\{A_1, \ldots, A_k\}$ of finite state automata, can we guess an order on them to efficiently perform their intersection? More precisely, can we find a permutation π for which the following algorithm will run as quickly as possible ?

```
A = A[pi[1]];
for i = 2 to k do
   A = A intersect A[pi[i]]
done
```

Observe that computing the intersection as above takes in the worst case exponential time. Indeed, the size of the result, that is to say the number of states, is exponential $|\bigcap_{i\leq k} A_i| = \prod_{i\leq k} |A_i|$. We refer to Saaloma and Yu to learn more about state complexity (YZS94; Yu06). But this is not the issue addressed here. The question is to find the order in which we have to perform the intersections. And we show that this part of the problem is also inherently difficult. To get rid of the size problem, we consider the ordering problem with regards to some a priori upper bound on the size of automata. The decision problem will be proved NP-complete.

A standard NP-complete problem about automata intersections is the emptiness of the result. See for instance (GJ79) or (KLV00) which give explicit upper bounds. But, here, we are more concerned with the intersection process than the result itself. An analogous question to our present issue is matrix multiplication. Given a sequence of matrices $M_1, \ldots, M_k$ of different sizes, the way one parenthesizes the expression $M_1 \times \cdots \times M_k$ has a huge impact on the cost of evaluating the product (see (CLR90)). For this problem, computing the best order can be done in polynomial time by a dynamic programming procedure.

Let us now present the practical application which originally motivated the present study: disambiguation for lexicalized polarized grammars (PGs) like Categorial Grammars (Moo96), Interaction Grammars (Per04) or Polarized Unification Grammars (Kah06). A lexicalized grammar is defined by its lexicon, which associates a set of *syntactic items* to every word of the language. Each of these items specifies a grammatical construction in which the corresponding word participates.

One of the main features of PGs is that each syntactic item is equipped with polarized features. Polarities are used to guide the process of syntactic composition: features with the same type but with opposite polarities try to neutralize each other. The process ends successfully in a parse structure for a sentence where all polarized features are neutralized.

Syntactic items, if we forget their structure, become bags of polarized features. A necessary condition for a tagging to be successful is that summing polarized features in the bag must end with a zero. Automata are a well-suited way of factorizing this counting. The crux is that one may count different features, each of which provides an automaton. Hence, the resulting necessary condition is given by the intersection of these automata (BGP04). Unfortunately, it is known (Tap97) that when performing multiple intersections, intermediate automata can possibly be huge, even if the final automaton is small.

We prove that looking for the order in which intersections have to be performed to create the minimal number of intermediate states is actually NP-hard. For that reason, we have used heuristics[1] in our implementation.

2 Polarized Grammars and Lexical Disambiguation

In this section, we present a general lexical disambiguation method for PGs relying on automata intersection.

2.1 Polarized Grammars and Parsing

We give here a very brief description of such grammars. Any reader who wants a wider presentation of these grammars should refer to (Moo96; Per04; Kah06). A

[1] We cannot present these heuristics here for lack of space.

polarized grammar is equipped with:

- a set W of words (for instance English vocabulary);
- a set S of items (for example "noun phrase coordination");
- a function $\ell : W \to \mathcal{P}_{\text{fin}}(S)$ which associates words with finite sets of items;
- a set of feature names $\mathcal{F}$ (e.g. "category" and "gender") and a set of feature values $\mathcal{V}$ (e.g. "noun" and "masculine");
- a function $\rho : S \times \mathcal{F} \times \mathcal{V} \to I[\mathbb{Z}]$ which associates to any item and feature name/value a finite interval over the integers. This function counts the polarized features of items. For instance, $\rho(\text{give\_Verb}, \text{"cat"}, \text{"noun phrase"}) = [-3, -1]$ because a verb like *give* can be intransitive (expecting 1 noun phrase), transitive (2 noun phrases) or ditransitive (3 noun phrases).

Given a sentence $w_1, \ldots, w_n$ of words in W, the parsing process consists of a) selecting one item for each word of the sentence, say $s_1, \ldots, s_n$ and b) checking that this selection verifies some properties depending on the grammatical framework. Still, there is one common property to all PGs which is that 0 must be an element of the sum of the intervals in the selection, where intervals are summed according to $[a, b] + [c, d] = [a + c, b + d]$. This property can be stated: $\forall f, v \in \mathcal{F} \times \mathcal{V} : 0 \in \sum_{i \leq n} \rho(s_i, f, v)$. We call this property the *global neutrality criterion* and it reflects the neutrality constraint on final structures.

2.2 Counting with Automata

We assume a sentence $w_1 w_2 \ldots w_n$ to parse with a PG G. Given a feature name and a feature value (f, v), consider the automaton $A(f, v)$ as follows:

- A state of the automaton is a pair (i, p), where i corresponds to the position of the word in the sentence and p is an interval of $\mathbb{Z}$, which represents the counting of polarities up to position i.
- Transitions have the form $(i, p) \xrightarrow{s_\alpha} (i + 1, q)$, where $s_\alpha \in \ell(w_i)$, $q = p + \rho(s_\alpha, f, v)$.
- The initial state is $(0, [0, 0])$.
- The accepting states are states (n, p) such that 0 is an element of p.

Every path in $A(f, v)$ from the initial state $(0, [0, 0])$ to an accepting state represents a lexical selection that verifies the global neutrality criterion. Other paths can be deleted. So, any path to an accepting state is a candidate for selection.

Actually, it is a necessary condition for a correct lexical selection to be recognized by polarity automata, for every *choice* of name f and value v. As a consequence, the intersection of polarity automata gives an automaton which also contains the valid solutions. The principle of our selection method is to build the automaton[2] $\bigcap_{(f,v)\in\mathcal{F}\times\mathcal{V}} A(f,v)$.

For example, in our implementation for Interaction Grammars, for a ten word long sentence we usually make twelve intersections. With this method we go from 5000 raw selections to 10 selections respecting the neutrality criterion. We have noticed some performance issues depending on the order in which we performed the intersection. On some sentences, we experienced tenfold variations in the number of states of the intermediate automata..

3 NP-Completeness of the Problem

In this section, we review three problems, which we prove to be NP-complete, related to our disambiguation technique based on automata intersections.

In these three problems we ask whether it is possible to determine the right order in which the intersection of several automata must be performed to minimize the number of intermediate states.

We prove NP-hardness of these problems by reduction from the Traveling Salesman Problem (TSP) (GJ79). To fix the notations, we first recall this illustrious problem.

An instance of the TSP is a triple (V, d, K) where $V = \{1, \ldots, n\}$ is a set of cities, d is a distance function between any pair of different cities, $d(i, j) \in \mathbb{N}$, and a bound $K \in \mathbb{N}^+$. The problem is to decide whether there exists a tour of all cities with a length less than K or in other words if there exists a permutation π of the cities such that $(\sum_{i=1}^{i=n-1} d(\pi(i), \pi(i+1))) + d(\pi(n), \pi(1)) \leq K$. For clarity, when π is a function $[1..n] \to [1..n]$ and the context is clear, we write $\pi(n+1)$ for $\pi(1)$ and $\pi(0)$ for $\pi(n)$. So the previous sum can be written $\sum_{i=1}^{i=n} d(\pi(i), \pi(i+1)) \leq K$. From now on, we restrict our attention to those cases where $d(i, j) \leq 2$. The problem remains NP-complete (it corresponds to the reduction from Hamiltonian Circuit).

We will distinguish between the traditional TSP as it has been described above and a variant that we call the exact TSP in which the tour must be of length exactly K (see (GJ79)).

3.1 Intersection Optimization Problems

We present a first intersection optimisation problem, that we will enrich to get the second and third problems, which are more difficult. In the proofs, we do not use

[2] Actually we can restrict our attention to some more particular values for f and v. See (BGP04) for details.

automata with loops. So these problems can be stated with or without star languages. For an automaton A, *the size of* A that we write $|A|$ is the number of states of A. Every automaton is considered minimal, unless stated otherwise.

First Problem (IO1): Let $\mathcal{A}_n = (A_i)_{1\leq i\leq n}$ be a set of n finite state automata, $B \in \mathbb{N}^+$ a bound and K a target size. Is there an injective function $\pi : [1..j] \to [1..n]$ such that

- $|(\dots(A_{\pi(1)} \cap A_{\pi(2)}) \cap \cdots) \cap A_{\pi(j)}| = K$
- for all $k < j$, $|(\dots(A_{\pi(1)} \cap A_{\pi(2)}) \cap \cdots) \cap A_{\pi(k)}| \leq B$.

In other words, is there a subset $\mathcal{A} \subseteq \mathcal{A}_n$ such that $|\bigcap_{A\in\mathcal{A}} A| = K$ with all intermediate steps smaller than B? For disambiguation, this means that we are able to know the size of the final intersection.

Second Problem (IO2): Let $\mathcal{A}_n = (A_i)_{1\leq i\leq n}$ be a set of n finite state automata and $B \in \mathbb{N}$ a bound. Is there a bijection $\pi : [1..n] \to [1..n]$ such that for any $j \leq n$ we have $|(\dots(A_{\pi(1)} \cap A_{\pi(2)}) \cap \cdots) \cap A_{\pi(j)}| \leq B$? For disambiguation, this means that given a set of polarity automata we are able to know how to perform their intersection in order to bound the size of each intermediate intersection.

Third Problem (IO3): Let $\mathcal{A}_n = (A_i)_{1\leq i\leq n}$ be a set of n finite state automata and $B \in \mathbb{N}$ a bound. Is there a permutation π of $[1..n]$ such that $\sum_{1\leq j\leq n} |(\dots(A_{\pi(1)} \cap A_{\pi(2)}) \cap \cdots) \cap A_{\pi(j)}| \leq B$? This is the problem that we deal with in disambiguation: is there an order to perform intersection for which the total number of states that we create is bounded ?

3.2 NP Algorithms

These three problems are in NP. Each time we have to choose a permutation π and then:

- for (IO1), if an intermediate intersection is empty we stop and the answer to the problem is "no" (of course, if $K = 0$ it is "yes") if it has a size greater than B, the answer is no. Otherwise we proceed to the next intersection. When j intersections have been performed we compare the size of the resulting automaton to K. Observe that those intersections can be performed in time bounded by B^2 since all intermediate steps must have a size lesser than B. And so, we are polynomial with regards to B.
- for (IO2), if an intermediate intersection is empty then the answer is "yes" else if it is greater than B (again, we may need to consider B^2 states before

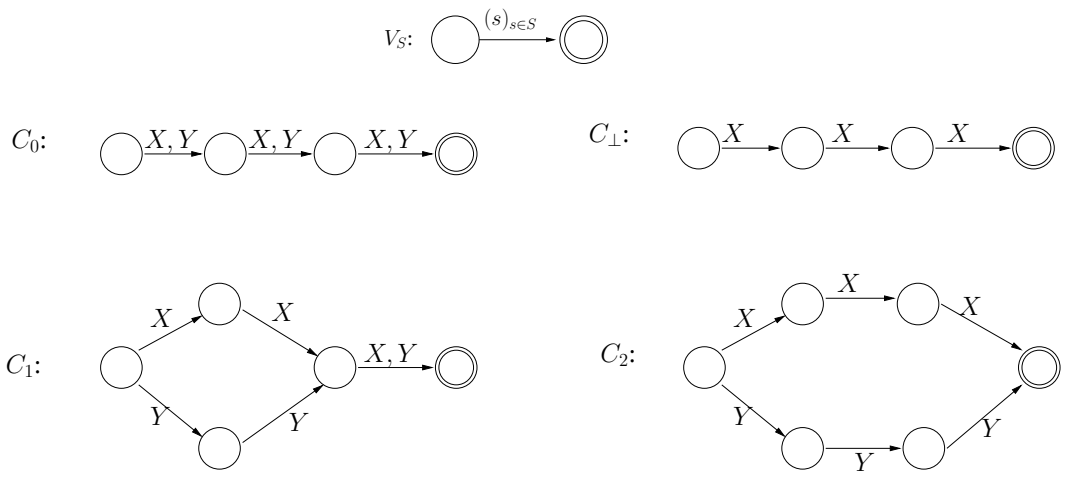

Figure 1: some brick automata

minimization) the answer to the problem is "no" else we proceed to the next intersection.

- for (IO3), we need to sum the sizes of the intermediate intersections and check that this sum is never greater than B. If an intersection is empty or if a partial sum exceeds B then we can stop immediately.

3.3 NP-Completeness

Theorem 1. ***(IO1)*** *is NP-complete.*

Proof. We consider some cells, that we will associate to build automata. They are given by Figure 1.

We can note that for $i \in \{0, 1, 2\}$ we have $|C_i| = |C_0| + i$. In other words, these automata encode the distance between two cities. Observe also that $C_i \cap C_0 = C_i$. So that C_0 is the "neutral" element for the intersection. Finally, if A is some automaton, A' denotes the same automaton, but with primed letters. We suppose we are given an arbitrary (but minimal) automaton D of size $6 \times n + 3$.

Now, given an instance of the exact TSP (V, d, k), we consider a set of automata $\mathcal{A}_{i,j,m}$ with $i, j \in V$ and $m \leq n$ where n is the number of cities in V. To fix the intuition, the automaton $\mathcal{A}_{i,j,m}$ corresponds to the choice of going from city i to city j at step m of a tour. In other words, it corresponds to the choice $\pi(m) = i$ and $\pi(m + 1) = j$. The m^{th} distance is set to $l = d(i, j)$ by cell C_l, between letters V_i and V_j. Moreover, if i is the initial city, it is also the last one. We define:

$$\begin{array}{lcl}
\mathcal{A}_{i,j,1} & = & V_i C_{d(i,j)} V_j (C_0 V_{V\setminus\{i,j\}})^{n-2} C_0 V_i + V'_{V\setminus\{1\}} D \\
\mathcal{A}_{i,j,m} & = & (V_{V\setminus\{i,j\}} C_0)^{m-1} V_i C_{d(i,j)} V_j (C_0 V_{V\setminus\{i,j\}})^{n-m} + V'_{V\setminus\{m\}} D \quad \text{for } n > m > 1 \\
\mathcal{A}_{i,j,n} & = & V_j C_0 (V_{V\setminus\{i,j\}} C_0)^{n-2} V_i C_{d(i,j)} V_j + V'_{V\setminus\{n\}} D
\end{array}$$

Let us consider the "witness" automaton $\mathcal{A} = (V_V C_0)^n V_V$ where no distance is set. Remark that $|\mathcal{A}_{i,j,m}| = |\mathcal{A}| + d(i,j) + |D|$. The (polynomial) reduction is then $(V, d, K) \mapsto ((\mathcal{A}_{i,j,m})_{i,j,m}, 2|D|, |\mathcal{A}| + K)$.

Correctness If there is a tour defined by π of length exactly K, observe that:

$$\bigcap_{1\leq m\leq n} \mathcal{A}_{\pi(m),\pi(m+1),m} = V_{\pi(1)} C_{d(\pi(1),\pi(2))} V_{\pi(2)} C_{d(\pi(2),\pi(3))} \cdots C_{d(\pi(n),\pi(1))} V_{\pi(1)}$$

which has a size $|\bigcap_{1\leq m\leq n} \mathcal{A}_{\pi(m),\pi(m+1),m}| = |\mathcal{A}| + \sum_{1\leq m\leq n} d(\pi(m), \pi(m+1))$. The bound on intermediate automata is discussed widely in the next proof. So, if the TSP has a solution, then its encoding has a solution for (IO1).

Completeness For the converse part, we consider the set $\mathfrak{A}$ of automata $(\mathcal{A}_{i,j,m})_{i,j,m}$ closed by intersection. Any non empty automata $A \in \mathfrak{A}$ has the following properties (proved by successive inductions):

(i) $A = A_1 + A_2$ with

- $A_1 = \emptyset$ or
- $A_1 = V_{\alpha_1} C_{\beta_1} V_{\alpha_2} C_{\beta_2} \cdots V_{\alpha_n} C_{\beta_n} V_{\alpha_{n+1}}$, $\alpha_i \subseteq V$, $\beta_i \in \{0,1,2\}$,

and $A_2 = V'_S D$ with $S \subseteq \{1..n\}$;

(ii) In (i), if $\alpha_j = \{k\}$ for some j, then no other α_ℓ contains k for $\ell \leq n$,

(iii) In (i), $\beta_i = 0$ iff $i \in S$,

(iv) In (i), if $\beta_i \neq 0$, then $\alpha_i = \{k\}, \alpha_{i+1} = \{\ell\}$ are singleton sets and $\beta_i = d(k,\ell)$.

(v) In (i), $\alpha_1 = \alpha_{n+1}$

From (i), we can say that $|A| \leq |A_1| + |A_2|$. So, in the worst case, $|A| \leq 2 + \sum_{i=1}^{n} |C_{\beta_i}| + |D| < 2 \times |D|$, and the bound on intermediate steps is always respected. From (iii), we learn that $V'_S D$ is empty iff $\forall j : \beta_j \neq 0$. So that (iv) with (ii,v) gives us the fact (F) that for all i, the set $\alpha_i = \{k_i\}$ is a singleton set and $\pi : [1..n] \to [1..n]$ which sends $i \mapsto k_i$ is a bijection and $k_1 = k_{n+1}$. Since, $|D| > |\mathcal{A}| + k$, $|A| = |\mathcal{A}| + k$ iff S is empty. The fact (F) above shows that it corresponds to an acceptable tour. □

Theorem 2. ***(IO2)*** *is NP-complete.*

Proof. We reduce the TSP to IO2. Let (V, d, k) be an instance of the TSP, let 2 be the maximal distance between two cities and $n = |V|$. Again, for each pair of cities (i, j) with distance $d(i, j)$ we build n automata according to the possible positions of these cities in a tour. That is to say we build n^3 automata $\mathcal{A}_{i,j,p}$. Technically speaking, with regards to (IO1), we must have a stronger control on the order in which the intersection is performed. This is due to the fact that we have a weaker condition that applies to every intermediate automaton. That is also why the proof is much more complex.

We can decompose automata in three components:

1. The first one detailed in Fig. 2 (that we call $\mathcal{C}_{1,i,j,p}$) is responsible for computing the total distance of the tour, like in (IO1) but without indexing V by a set of cities. The end states of the C_0 sub-components are connected to the initial state of $\mathcal{C}_{2,i,j,p}$ by a dummy letter X. Hence, if all the distances are instantiated (as in IO1) then only the last V will connect this first component to the second component.

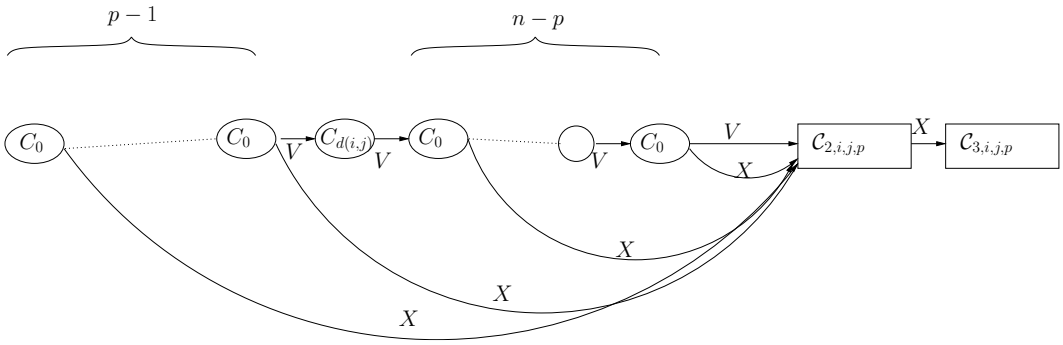

Figure 2: automaton $\mathcal{A}_{i,j,p}$ with detailed first component

2. The second one ($\mathcal{C}_{2,i,j,p}$) is responsible for chaining the edges correctly to make a valid tour. This component is shown on Figure 3. It should be observed that if it is intersected with $\mathcal{C}_{2,j,k,p+1}$ then the resulting automaton is of the same size. Otherwise (if city indices do not match) then it grows by $2n$ states.

3. The third one ($\mathcal{C}_{3,i,j,p}$), presented in Fig. 4, *forbids* (by making any unwanted intersection too big) the use of a position more than once and the use of position p without first considering positions $1, \ldots, p-1$. Otherwise it grows by $4n$ states.

Finally we need an additional automaton T, shown on Fig. 5. Its role is to end the intersection process.

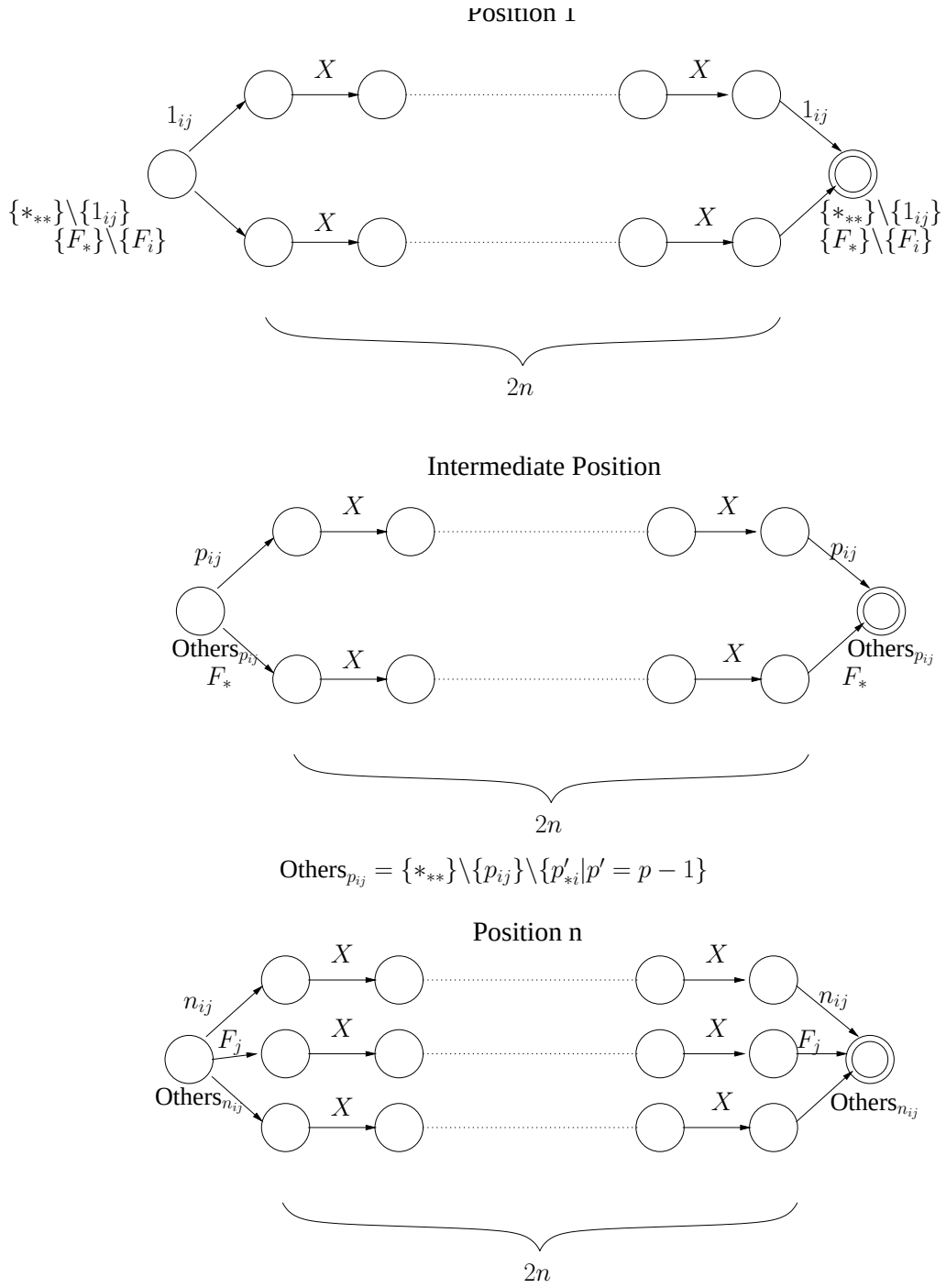

Figure 3: the second component for the automaton $\mathcal{A}_{i,j,p}$

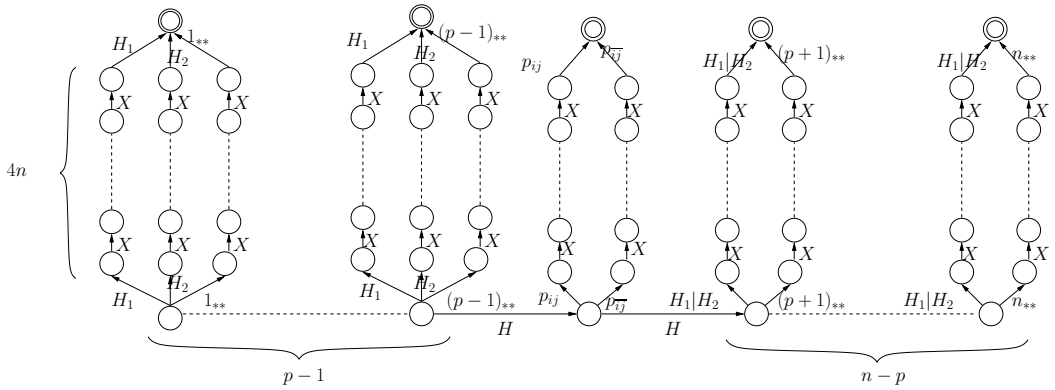

Figure 4: the third component for the automaton $\mathcal{A}_{i,j,p}$

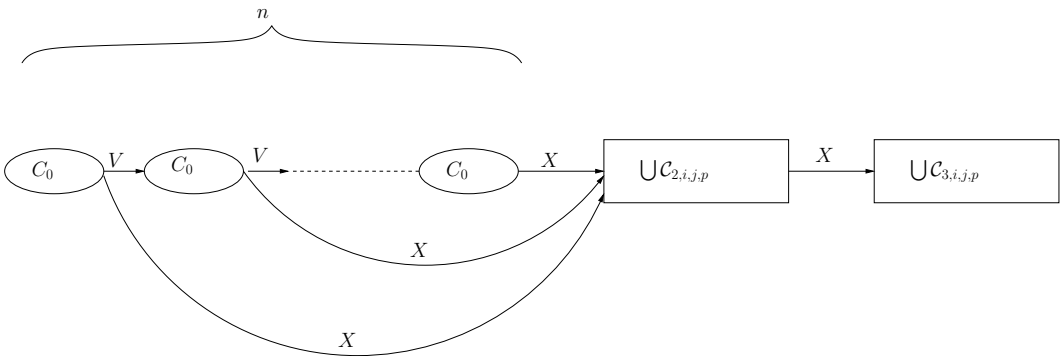

Figure 5: the automaton T

The size of $\mathcal{A}_{i,j,p}$ is $|\mathcal{A}_{i,j,p}| = |\mathcal{C}_{1,i,j,p}| + |\mathcal{C}_{2,i,j,p}| + |\mathcal{C}_{3,i,j,p}|$ where

$$|\mathcal{C}_{1,i,j,p}| = 2n + d(i,j)$$
$$|\mathcal{C}_{2,i,j,p}| = \begin{cases} 6n+2 & \text{if } p = n \\ 4n+2 & \text{otherwise} \end{cases}$$
$$|\mathcal{C}_{3,i,j,p}| = 3(p-1)(4n) + 2(n-p+1)(4n) + 2n = 2n(4n+2p-1)$$

$$|\mathcal{A}_{i,j,p}| = \begin{cases} 2 + d(i,j) + 4n(2+p+2n) & \text{if } 1 \leq p < n \\ 2 + d(i,j) + 2n + 4n(2+3n) & \text{otherwise} \end{cases}$$

We want to prove that $i_1, i_2, \ldots, i_n, i_1$ is a tour for the TSP with length lesser than k if and only if every intermediate automaton of the intersection

$$\bigcap_{1 \leq i \leq m, \alpha(i) \in I \subset [1..n]^3} \mathcal{A}_{\alpha(i)} \cap T \bigcap_{m+1 \leq j \leq n^3, \beta(j) \in [1..n]^3 \setminus I} \mathcal{A}_{\beta(j)}$$

is an automaton whose size is lesser than $B = 2 + K + 4n(1+2n)$. So the reduction is $(V = \{1 \ldots n\}, d, K) \mapsto ((\mathcal{A}_{i,j,p})_{1 \leq i,j,p \leq n} \cup T, B)$

Preliminary observations. Without loss of generality, we can suppose that $K \leq 2n$. Otherwise the TSP is trivial. This entails that, among all the automata $(\mathcal{A}_{i,j,p})_{i,j,p}$, only the automata $(\mathcal{A}_{i,j,1})_{i,j}$ are smaller than B:

i) $|\mathcal{A}_{i,j,1}| = 2 + d(i,j) + 4n(1+2n) < B$

ii) otherwise,

$$\begin{aligned} |\mathcal{A}_{i,j,p}| &\geq 2 + d(i,j) + 4n(1+p+2n) \\ &\geq 2 + d(i,j) + 4n(2+2n) + 4n(p-1) \\ &> 2 + d(i,j) + 4n(1+2n) + K > B \end{aligned}$$

Then, notice that:

iii) $|\mathcal{C}_{1,i,j,p}| = 2n + d(i,j)$

iv)
$$|\mathcal{C}_{1,i,j,p} \cap \mathcal{C}_{1,k,l,q}| = \begin{cases} 2n + d(i,j) + d(k,l) & \text{if } p \neq q \\ 2n + \max(d(i,j), d(k,l)) & \text{otherwise} \end{cases}$$

v)
$$|\mathcal{C}_{2,i,j,p} \cap \mathcal{C}_{2,k,l,q}| = \begin{cases} |\mathcal{C}_{2,i,j,p}| & \text{if } q = p+1 \text{ and } j = k \\ |\mathcal{C}_{2,i,j,p}| + 2n & \text{otherwise} \end{cases}$$

vi)

$$|\mathcal{C}_{3,i,j,p} \cap \mathcal{C}_{3,k,l,q}| = \begin{cases} |\mathcal{C}_{3,i,j,p}| & \text{if } q = p+1 \\ |\mathcal{C}_{3,i,j,p}| + 4n|p-q| & \text{if } q > p+1 \\ |\mathcal{C}_{3,i,j,p}| + 4n & \text{if } q \leq p \end{cases}$$

(v) and (vi) mean that there is a way to preserve the size of the second and third components: it is to perform the intersection with respect to the order of a tour (v) and by considering each position once in ascending order (vi). Following these remarks, for any sequence prefix of a tour $i_1, i_2, \ldots, i_k$ with $k \leq n+1$ (if $k = n+1$ we force $i_k = i_1$) in our instance of the TSP, we have

$$\begin{array}{rcl} |\bigcap_{1\leq p\leq k} \mathcal{A}_{i_p,i_{p+1},p}| & = & |\bigcap_{1\leq p\leq k} \mathcal{C}_{1,i_p,i_{p+1},p}| + |\bigcap_{1\leq p\leq k} \mathcal{C}_{2,i_p,i_{p+1},p}| + |\bigcap_{1\leq p\leq k} \mathcal{C}_{3,i_p,i_{p+1},p}| \\ & = & |\bigcap_{1\leq p\leq k} \mathcal{C}_{1,i_p,i_{p+1},p}| + |\mathcal{C}_{2,i_1,i_2,1}| + |\mathcal{C}_{3,i_1,i_2,1}| \\ & = & 2n + (\sum_{1\leq p\leq k} d(i_p, i_{p+1})) + 4n + 2 + 2n(4n-1) \\ & = & 2 + (\sum_{1\leq p\leq k} d(i_p, i_{p+1})) + 4n(1+2n) \end{array}$$

Correctness of the reduction. We first show that if there exists a tour $i_1, i_2, \ldots, i_n, i_1$ with length lesser than k then all intermediate intersections of $\mathcal{A}_{i_1,i_2,1} \cap \ldots \cap \mathcal{A}_{i_j,i_{j+1},j}$ for j ranging from 1 to n have a size lesser than B. We do this by induction on the steps of the intersection process.

As stated earlier, the initial automaton must be $\mathcal{A}_{i_1,i_2,1}$ because every other $\mathcal{A}_{i_1,i_2,p}$ would be too large. Then, by application of the equality defined above:

$$\begin{array}{rcl} |A = \bigcap_{1\leq p\leq n} \mathcal{A}_{i_p,j_{p+1},p}| & = & 2 + (\sum_{1\leq p\leq n} d(i_p, i_{p+1})) + 4n(1+2n) \\ & \leq & 2 + K + 4n(1+2n) = B \end{array}$$

So these first n intersections straightforwardly encode the tour in the TSP instance. Now, observe that $A \cap T = \emptyset$ because every C_i from its first component is different from C_0. Consequently if the instance of the TSP has a solution, the sequence $\mathcal{A}_{i_1,i_2,1}, \ldots, \mathcal{A}_{i_n,i_1,n}, T, S$ where S is a sequence over $\{(\mathcal{A}_{i,j,p})_{i,j,p}\} \backslash \{\mathcal{A}_{i_k,i_{k+1},k} : k \leq n\}$ is a solution to IO2.

Completeness. Consider an intersection of the form

$$A = (\bigcap_{(\alpha_i)_i \in I} \mathcal{A}_{\alpha_i}) \cap T \cap (\bigcap_{(\alpha_j)_j \in [1..n]^3 \backslash I} \mathcal{A}_{\alpha_j})$$

where no intermediate automaton has a size greater than B. In particular this is true for $A' = (\bigcap_{(\alpha_i)_i \in I} \mathcal{A}_{\alpha_i})$. We note $m = |I|$ and we can deduce that:

- α_1 is of the form $(x_1, y_1, 1)$; if $\alpha_i = (x_i, y_i, p)$ then $\alpha_{i+1} = (x_{i+1}, y_{i+1}, p+1)$ and $m \leq n$, otherwise component 3 would make $|A'| > B$. This implies that α_i is of the form (x_i, y_i, i)
- if $\alpha_i = (x_i, y_i, i)$ and $\alpha_{i+1} = (x_{i+1}, y_{i+1}, i+1)$ then $y_i = x_{i+1}$ and $m = n$ implies that α_m is of the form (x_m, x_1, m). Otherwise component 2 would make $|A'| > B$
- Finally, $m \geq n$ otherwise $|A' \cap T| > B$. This implies that $m = n$. Remark that this also implies that $|A' \cap T| = 0$.

In other words A' encodes a tour $i_1, i_2, \ldots, i_n, i_1$ in our instance of the TSP. Furthermore the size of A' if we follow its construction as stated above is

$$\begin{array}{rcl} |A'| = |\mathcal{C}_1| + |\mathcal{C}_2| + |\mathcal{C}_3| & \leq & B \\ |\mathcal{C}_1| + |\mathcal{C}_{2,i_1,i_2,1}| + |\mathcal{C}_{3,i_1,i_2,1}| & \leq & B \\ |\mathcal{C}_1| + 4n + 2 + 2n(4n-1) & \leq & B \\ |\mathcal{C}_1| + 2 + 2n(1+4n) & \leq & 2 + K + 4n(1+2n) \\ |\mathcal{C}_1| & \leq & K + 2n \\ 2n + d(i_1, i_2) + \cdots + d(i_n, i_1) & \leq & K + 2n \\ d(i_1, i_2) + \cdots + d(i_n, i_1) & \leq & K \end{array}$$

And so the tour is actually a solution for our instance of the TSP. □

Theorem 3. ***(IO3)*** *is NP-complete.*

Proof. The encoding remains the same that the one for (IO2) except for the first component. The non-instantiated distances before position p are erased by intersection with $C_\perp$. (Notice that $C\perp \cap C_{i\in\{0,1,2\}} = C_\perp$)

$$\mathcal{A}_{i,j,p} = (V(C_\perp + X\mathcal{C}_{2,i,j,p}X\mathcal{C}_{3,i,j,p}))^{p-1} V C_{d(i,j)} (V(C_0 + X\mathcal{C}_{2,i,j,p}X\mathcal{C}_{3,i,j,p}))^{n-p} V(\mathcal{C}_{2,i,j,p}X\mathcal{C}_{3,i,j,p})$$

The bound for this problem is $B = K + n(2 + 2n + |\mathcal{C}_2| + |\mathcal{C}_3|) = K + n(2 + 8n + 8n^2)$ which corresponds to K and the part of the first automaton of the tour that does not disappear by intersection before the intersection with T.

□

4 Conclusion

We showed that determining the best way to intersect a set of automata is an intractable problem. This compels finite state applications to look for clever heuristics. In our own implementation we choose to perform intersections according to the ascending size of the automata. (Tap97) gives several other heuristics. Another possibility is to approximate the intersection, see (YJ04).

References

G. Bonfante, B. Guillaume, and G. Perrier. Polarization and abstraction of grammatical formalisms as methods for lexical disambiguation. 2004.

T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. MIT Press, 1990.

M. R. Garey and D. S. Johnson. *Computers and Intractability*. Freeman, San Francisco, 1979.

Sylvain Kahane. Polarized unification grammars. In *ACL*. The Association for Computer Linguistics, 2006.

George Karakostas, Richard J. Lipton, and Anastasios Viglas. On the complexity of intersecting finite state automata. In *IEEE Conference on Computational Complexity*, pages 229–234, 2000.

M. Moortgart. Categorial Type Logics. In J. van Benthem and A. ter Meulen, editors, *Handbook of Logic and Language*, chapter 2. Elsevier, 1996.

G. Perrier. La sémantique dans les grammaires d'interaction. *Traitement Automatique des Langues*, 45(3):123–144, 2004.

Pasi Tapanainen. *Finite-State Language Processing*, chapter Applying a Finite-State Intersection Grammar. MIT, 1997.

Anssi Mikael Yli-Jyrä. Simplifications of intermediate results during intersection of multiple weighted automata. In M. Droste and H. Vogler, editors, *Weighted Automata: Theory and Applications*, 2004.

Sheng Yu. On the state complexity of combined operations. In Oscar H. Ibarra and Hsu-Chun Yen, editors, *CIAA*, volume 4094 of *Lecture Notes in Computer Science*, pages 11–22. Springer, 2006.

Sheng Yu, Qingyu Zhuang, and Kai Salomaa. The state complexities of some basic operations on regular languages. *Theor. Comput. Sci.*, 125(2):315–328, 1994.