NATURAL
LANGUAGE ENGINEERING

CAMBRIDGE
UNIVERSITY PRESS

Sarf: Efficient and Application Customizable Arabic Morphological Analyzer

Journal:	*Natural Language Engineering*
Manuscript ID	NLE-ARTC-15-0011.R1
Manuscript Type:	Article
Date Submitted by the Author:	n/a
Complete List of Authors:	zaraket, fadi; American University of Beirut, Electrical and Computer Engineering Jaber, Ameen; American University of Beirut, ECE Makhlouta, Jad; American University of Beirut, Electrical and Computer Engineering Safieddine, Maya; American University of Beirut, Electrical and Computer Engineering
Keywords:	Morphology, Language Resources

Natural Language Engineering **1** (1): 1–32. Printed in the United Kingdom

Sarf: Fast and Application Customizable Arabic Morphological Analyzer

Fadi A. Zaraket

Ameen Jaber

Jad Makhlouta

Maya H. Safieddine

Electrical and Computer Engineering,
American university of Beirut

(*Received 20 March 2014; revised 30 September 2014*)

Abstract

The rich nature of Arabic morphology makes morphological analysis key for Arabic natural language processing applications. Arabic morphological analyzers return several morphological solutions for a given Arabic word. Each solution consists of several morphological features such as part of speech and gloss description tags. Often times, applications need only few of those features. This paper presents Sarf, an application customizable morphological analyzer for Arabic. Sarf provides an interface that allows application developers to (1) control and prioritize the analysis, (2) refine solution features, and (3) define categories and associate them with existing morphemes.

Sarf uses agglutinative and fusional morphemes for affix representation, and extends and refines the morpheme lexicons of SAMA and BAMA. This reduces redundant morphemes, and subsequently inconsistent morpheme tags in the lexicons. It also solves the segmentation correspondence problem between an input word and the several parts of the associated morphological solution. It uses diacritics to refine solutions, and solves the 'run-on words' problem. The implementation of Sarf efficiently encodes the morpheme lexicons. Sarf was used in several NLP applications for information extraction and provided more accurate solutions than existing solvers with faster running time.

1 Introduction

Natural language processing (NLP) applications such as *machine translation* (MT) and *information extraction* (IE) require *morphological analysis* to preprocess Arabic text due to the rich morphological nature of the Arabic language (Benajiba, Rosso, & Benedíruiz, 2007; Habash & Sadat, 2006). Arabic morphological analyzers return the internal structure of a given Arabic word composed of several *morphemes* including *affixes* (*prefixes* and *suffixes*), *clitics* (*proclitics* and *enclitics*), and *stems* (Al-Sughaiyer & Al-Kharashi, 2004). The morphological solution consists also of several *morphological features* (tags) associated with the word and its constituent morphemes such as *part of speech* (POS), transliteration, *gloss*, and *vocalized morpheme form* (VMF) tags (diacriticized form of the morpheme). The prefix and suffix attach before and after the stem, respectively. Clitics are

special affixes that attach to the stem to form a word, and differ from regular affixes in that they play a syntactic role of another word (often omitted) (Habash, 2010).

This work presents Sarf, a *fast and application customizable morphological analyzer* for Arabic that is used in several applications for information extraction from Arabic text (Jaber & Zaraket, 2013; Makhlouta, Zaraket, & Harkous, 2012; Zaraket & Makhlouta, 2012a, 2012c). Sarf provides NLP application developers with an application programming interface (API) to control and refine morphological analysis on the fly. The developer implements the interfaces in the application. Sarf calls the interfaces on control points such as prefix, stem, suffix, and full solution matches. The Sarf API allows the application to (1) control and prioritize the analysis, (2) refine the solution features, and (3) define developer categories and associate them with existing morphemes.

Sarf is a significant extension of the work in (Zaraket & Makhlouta, 2012b). It represents Arabic affixes as agglutinative affix morphemes with fusional affix concatenation rules. Simpler agglutinative affix morphemes can be concatenated to form a more complex affix (Vajda, 2001). Fusional affix concatenation rules specify affix pairs and use regular expressions as substitution rules to compose the resulting orthographic and semantic tags from the tags of the original morphemes (Spencer, 1991). The Sarf substitution rules are in sync with rules and examples on morpheme concatenative properties from Arabic morphology textbooks (AlRajehi, 2000a, 2000b). This representation resolves consistency, maintenance, and segmentation issues of the current approaches in BAMA and SAMA.

In this paper, we make several additional contributions including the following.

- Sarf provides an application customizable morphological analyzer where the developer can control and refine the analysis.
- Sarf structures and computations are engineered to reduce computation cost as follows.
  - — The organization of the lexical structures into root index tries during runtime.
  - — The traversal structures that allows multiple solutions to be considered at once.
  - — The computation of the solution features in a tree based structure instead of computing all features and filtering later on
  - — The manual optimization of the traversal of the lexicon tries to reduce the typical excessive non-determinism exhibited in the finite state machine based solutions.
- Sarf is a novel Arabic morphological analyzer with agglutinative affixes and fusional affix concatenation rules based on textbook Arabic morphological rules and on the concatenation rules of existing analyzers.
- Sarf solves the "run-on" words problem and optionally uses partial diacritics for solution disambiguation.
- Sarf solves inconsistencies in existing affix lexicons of BAMA and SAMA. In this paper we list and discuss the inconsistencies.
- Sarf solves the correspondence between the morphological solution and the morphological segmentation of the original text problem.
- Sarf is fully implemented and available online as an open source tool.[1]

[1] http://research-fadi.aub.edu.lb/carla/doku.php?id=sarf

	suffix ونها	stem لعبـ	prefix وسيـ
POS	IVSUFF_SUBJ:MP_MOOD:I+IVSUFF_DO:3FS	VERB_IMPERFECT	CONJ+FUT+IV3MP
Transliteration	uwnahA	loEab	wa+sa+ya
Gloss	[MASC.PL.]+it/them/her.	play	and they will

Table 1. Morphological solution for وسيلعبونها*wsylʕbwnhā* (and they will play it)

We evaluated Sarf for segmentation correspondence, lexicon size, lexicon consistency, accuracy, and runtime efficiency. Our results show that Sarf lexicons are smaller than lexicons of existing analyzers and provide coverage for more morphological solutions. Sarf simplifies the lexicon maintenance task, provides better accuracy and improves run time efficiency as compared to existing analyzers.

We evaluated Sarf also within applications that use its API (Jaber & Zaraket, 2013; Makhlouta et al., 2012; Zaraket & Makhlouta, 2012a, 2012c). The carried case studies show that Sarf performs better than existing Arabic morphological analyzers in terms of running time. They also demonstrate the efficiency and the utility of the Sarf application customizable API.

The rest of this paper is structured as follows. In Section 2, we motivate Sarf and discuss how it deals with existing morphological analysis challenges. In Section 3, we compare Sarf to related work. In Section 4, we present the structre of Sarf. In Section 5, we present the interface provided by Sarf for the application developer to control and refine the morphological analysis. In Section 6, we present our method to build agglutinative affix morphemes with fusional affix concatenation rules. In Section 7, we present our method of using partial diacritics to reduce the morphological ambiguity. In Section 8, we discuss the use of Sarf with several NLP tasks and present the results of comparing Sarf to other analyzers in terms of speed, accuracy, and consistency. Finally, we conclude and discuss future work in Section 9.

2 Motivation

In this section, we discuss issues that currently challenge morphological analysis and explain how Sarf resolves them.

2.1 Exhaustive enumeration and performance:

Current morphological analyzers such as BAMA (Buckwalter, 2002), SAMA (Kulick, Bies, & Maamouri, 2010a; Maamouri, Graff, Bouziri, Krouna, Bies, & Kulick, 2010), Beesley (Beesley, 2001), MADAMIRA (Pasha et al., 2014), and ElixirFM (Smrž, 2007) take as input white space delimited tokens, consider them as words, and enumerate all possible morphological solutions for each word.

For example, given the word وسيلعبونها*wsylʕbwnhā* [2] (and they will play it), an

[2] The paper uses ArabTex package (Legally, 2004) for Arabic text editing and transliteration. ArabTex package is recommended and used by (Nizar Y., 2009).

analyzer may return the solution presented in Table 1 with a prefix, a stem, a suffix, and their corresponding POS, transliteration, and gloss tags. The prefix وسيـ can be further segmented into (1) the proclitic و with POS tag CONJ and gloss tag and, (2) the proclitic سـ with POS tag FUT and gloss tag will, and (3) the prefix يـ with POS tag IV3MP and gloss tag they(people). Similarly, the suffix ونها can be segmented into (1) the suffix ـون, forming a circumfix with يـ, with POS tag IVSUFF_SUBJ:MP_MOOD:I and gloss tag [MASC.PL.], and (2) the enclitic ـها with POS tag IVSUFF_DO:3FS and gloss tag it/them/her.

The exhaustive enumeration of all solutions may hurt performance and may not be necessary or appropriate in some applications (Maamouri, Bies, Kulick, Zaghouani, et al., 2010). Thus the need of a customizable morphological analyzer that adapts to application specific requirements.

2.2 Diacritics and accuracy

The accuracy of morphological analysis suffers from inherent difficulties of the Arabic language such as omitted diacritics and position dependent letter forms. Diacritics, i.e. short vowels, such as fatha (ـَ *a*), damma (ـُ *u*), kasra (ـِ *i*), tanween (i.e. doubled diacritic including ـً *an* , ـٌ *un* , ـٍ *in*), and sokun (ـْ) are almost always omitted in written Arabic text as they can be inferred by human readers. The mark shadda (ـّ) denotes the repetition of the marked character and is also often omitted. Partial diacritics can help disambiguate solutions. Consider the unvocalized word أكل *akl* with nine morphological solutions. Its partially vocalized version أَكِل *akil* has only two solutions; VMF أَكِل *aakil* with gloss I+trust/put in charge, and VMF أُكِّل *aukil* with gloss I+make tired/wear out.

Analyzers such as BAMA and SAMA ignore partial diacritics while other analyzers such as (Attia & Elaraby Ahmed, 2000; Beesley, 2001; Chaâben Kammoun, Hadrich Belguith, & Ben Hamadou, 2010) make use of the partial diacritics to reduce ambiguity. Sarf provides an option that enables the use of existing diacritics for disambiguation, and considers the diacritics at morpheme boundaries to generate only the diacritic matching solutions, rather than generating all morphological solutions then filtering them.

2.3 Problem of 'run-on' words:

Arabic letters have up to four different forms corresponding to their position in a word, i.e, beginning, middle, end, and separate word forms. This allows the phrase الىالمدرسة (to the school) to be visually recognizable as two separate words الى *ilā* (to) and المدرسة *almdrsh* (the school) without the need of a delimiter space in between. The reason is the first word الى *ilā* ends with ى *ā* a non-connecting letter. These words, referred to as 'run-on' words (Buckwalter, 2004), occur regularly, and greatly increase the difficulty of tokenization. None of existing morphological analyzers, up to our knowledge, resolve this issue.

Prefix	Vocalized	Category	Gloss	POS data
ف*f*	فَ*fa*	Pref-Wa	and/so	fa/CONJ+
ي*y*	يَ*ya*	IVPref-hw-ya	he/it	ya/IV3MS+
في*fy*	فَيَ*faya*	IVPref-hw-ya	and/so + he/it	fa/CONJ+ya/IV3MS+
سي*sy*	سَيَ*saya*	IVPref-hw-ya	will + he/it	sa/FUT+ya/IV3MS+
فسي*fsy*	فَسَيَ*fasaya*	IVPref-hw-ya	and/so + will + he/it	fa/CONJ+sa/FUT+ya/IV3MS+
ي*y*	يَ*ya*	IVPref-hmA-ya	they (both)	ya/IV3MD+
في*fy*	فَيَ*faya*	IVPref-hmA-ya	and/so + they (both)	fa/CONJ+ya/IV3MD+
سي*sy*	سَيَ*saya*	IVPref-hmA-ya	will + they (both)	sa/FUT+ya/IV3MD+
فسي*fsy*	فَسَيَ*fasaya*	IVPref-hmA-ya	and/so + will + they (both)	fa/CONJ+sa/FUT+ya/IV3MD+
و*w*	وَ*wa*	Pref-Wa	and	wa/CONJ+
وي*wy*	وَيَ*waya*	IVPref-hw-ya	and + he/it	wa/CONJ+ya/IV3MS+
وسي*wsy*	وَسَيَ*wasaya*	IVPref-hw-ya	and + will + he/it	wa/CONJ+sa/FUT+ya/IV3MS+
وي*wy*	وَيَ*waya*	IVPref-hmA-ya	and + they (both)	wa/CONJ+ya/IV3MD+
وسي*wsy*	وَسَيَ*wasaya*	IVPref-hmA-ya	and + will + they (both)	wa/CONJ+sa/FUT+ya/IV3MD+

Table 2. Partial BAMA v1.2 prefix lexicon

Sarf on the other hand does by the structures and rules it introduces, which are further discussed in the next issue.

2.4 Structure of existing analyzers and related problems:

Concatenative morphological analyzers (Buckwalter, 2002; Kulick et al., 2010a) are based on lexicons of prefixes L_p, stems L_s, and suffixes L_x. As shown in Table 2, each entry in a lexicon includes the morpheme, its vocalized form with diacritics, a *concatenation compatibility category* (CCC) rule, a gloss tag, and a POS tag. Separate CCC rules specify the compatibility of prefix-stem R_{ps}, stem-suffix R_{sx}, and prefix-suffix (circumfix) R_{px} concatenations. The affixes و and يـ in the before mentioned example are valid standalone prefixes, and can be concatenated to the stem لعب (he plays) to form ولعب (and he plays) and يلعب (he is playing), respectively. The L_p and L_x lexicons contain also all final forms of concatenated affixes as shown in Table 2 for sample morphemes.

This is the source of several problems:

- L_p and L_x contain redundant entries which result in maintenance and consistency issues (Kulick, Bies, & Maamouri, 2010b; Maamouri, Kulick, & Bies, 2008).
- Augmenting L_p and L_x with additional morphemes, such as *ʔaa* أَ (the question glottal hamza), may result in a quadratic increase in the size of the lexicons (*Hunspell Manual Page.*, 2012). The additional morpheme may attach to exiting morphemes. Currently, this results in adding all the resulting morphemes to the BAMA and SAMA lexicons.
- The L_p and L_x lexicons are larger than needed especially that they have to account for several forms of a morpheme with varying diacritics.
- The concatenated forms in L_p and L_x contain concatenated POS and other tags. The alignment and correspondence between the original word and its morphemes with the tags of its morphological solution are essential to the success of NLP tasks such as MT and IE (Lee, Haghighi, & Barzila, 2011; Nasredine, Laib, & Fluhr, 2008). The analysis of the example للقضاء*llqḍāʾ* (for/to the justice/judiciary) using SAMA, generates the morphological solution `li/PREP + Al/DET + qaDA'/NOUN`. The issue of the alignment and correspondence between the original word للقضاء and its morphemes arises as the concatenation of the unvocalized prefix morphemes `li` and `Al` is not the same as the corresponding prefix segment لل in the original word. This is not a problem for SAMA, but rather for the applications that use the SAMA output such as treebanking that is interested in `li` be present as a separate leaf in the syntactic tree. Sarf provides a general solution for the segmentation correspondence problem since the valid compound affixes preserve the input text segmentation. In particular, a partial affix لAl/DET connects to the atomic affix لli/PREP and resolves the problem.

Alternatively, Sarf extends and refines the lexicons and compatibility category rules of BAMA and SAMA such that:

- it represents only atomic affix morphemes in the lexicons,
- it adds prefix-prefix R_{pp} and suffix-suffix R_{xx} agglutinative and fusional rules,
- it generates compound affixes from the atomic ones using R_{pp} and R_{xx}, and
- it extends the stem lexicon with additional words necessary for the NLP applications it considers.

Considering the entries of Table 2 as example, Sarf represents them using only five atomic affix morphemes and five prefix-prefix rules.

3 Related work

In this section, we review work related to Arabic morphological analyzers, segmentation correspondence, partial diacritics, and application specific analyzers and compare it to Sarf.

Table 3 summarizes the comparison between Sarf and related Arabic morphological analyzers. Only ElixirFM (Smrž, 2007), Beesley (Beesley & Karttunen, 2003), and Fassieh (Attia, Rashwan, & Al-Badrashiny, 2009) provide root-pattern analysis of the stem. ElixirFM, MADAMIRA (Pasha et al., 2014), and MADA+TOKAN (Habash, Rambow, & Roth, 2009) are based on BAMA and SAMA and use functional and statistical techniques

	Sarf	SAMA	ElixirFM	MADA+ TOKAN	MADAMIRA	Beesley	Fassieh
Application customizable	✓	-	-	-	-	-	-
Feature selection	✓	-	-	-	✓	-	-
Run-on words	✓	-	-	-	-	-	-
Partial diacritics	✓	-	-	-	-	✓	-
Affix segmentation	✓	-	functional	tokenization schemes	statistical	-	-
Root-Pattern	-	-	✓	-	-	✓	✓
Automated disambiguation	-	-	-	SVM	SVM	-	maximum aposteriori

Table 3. Comparison of Sarf with SAMA, ElixirFM, MADA+TOKAN, MADAMIRA, Beesley, and Fassieh

to address the segmentation problem by reverse engineering the multiple tags of the affixes. Sarf differs in that the segmentation is an output of the morphological analysis and not a reverse engineering of the multi-tag affixes. Sarf is the only analyzer that addresses the 'run-on words' problem and solves it while performing the analysis. MADA+TOKAN, MADAMIRA, and Fassieh apply morphological disambiguation using support vector machines (SVM), and *maximum a posteriori* (MAP) estimation, respectively. Beesley and Sarf consider partial diacritics to eliminate morphological solutions that are not in agreement with the partial diacritization. Sarf provides an application customizable analyzer that enables the developer to control and refine the analysis on the fly and filter the solution features. MADA+TOKAN and MADAMIRA provide partial control over the output, and not the analysis, where MADA+TOKAN allows the user to select from several segmentation schemes and MADAMIRA enables the user to select solution features.

Sarf builds upon the lexicon of Buckwalter(Buckwalter, 2002). SAMA is an updated version of BAMA with increased lexicon coverage and additional POS tags (Maamouri, Graff, Bouziri, Krouna, & Kulick, 2010). Sarf differs from Buckwalter and SAMA in that it defines agglutinative and fusional affixes using a shorter list of affixes and a list of concatenation compatibility rules that allow prefix-prefix and suffix-suffix concatenations. This allows Sarf to better maintain the morphological tags associated with the affixes.

Buckwalter(Buckwalter, 2002) and SAMA (Maamouri, Bies, Kulick, Zaghouani, et al., 2010) produce a set of segmentation solutions for a word, compute the morphological solutions for each segment, compute the product of the solutions, eliminate the incompatible solutions, and then report the valid solutions. Sarf traverses the affix and stem structures with the input word character by character and keeps a stack of morpheme nodes. When a morpheme node in a structure is met, it is checked for compatibility with the stack of nodes. Consequently, Sarf generates only the solutions with valid segmentation, and reports only those with compatible stem and affix concatenation.

SAMA was refined to interact with the Arabic Treebank(ATB) (Maamouri & Bies, 2004) project after the addition of a large new corpus. The algorithmic changes in SAMA were

done manually to integrate with the ATB format. Sarf API allows for customizable refinements and allows Sarf to interact with any application on the fly without the modification of the morphological engine itself as was reportedly required with SAMA (Maamouri, Bies, Kulick, Zaghouani, et al., 2010)

Like ElixirFM (Smrž, 2007), Sarf builds on the lexicon of the Buckwalter analyzer. Sarf also uses deterministic parsing with tries and DAGs to implement the affix and stem structures. We think that the inferential-realizational approach of ElixirFM that is highly compatible with the Arabic linguistic description (Badawi, Carter, & Gully, 2004) can benefit from many features unique to the Arabic language. Sarf leaves implementing that to the developer customization through the API since in several cases the NLP application that uses the morphological analyzer needs only a partial linguistic model of Arabic.

MADA+TOKAN (Habash et al., 2009) is a toolkit for Arabic tokenization, diacritization, morphological disambiguation, POS tagging, stemming, and lemmatization. Sarf performs all those tasks except for morphological disambiguation where MADA uses SVM. Sarf keeps a stack of the positions that partition text into morphemes as part of the solution construction process. Therefore, the text segments corresponding to the solution features is preserved with no need for further post-processing.

MADAMIRA is a tool for Arabic morphological analysis and disambiguation that is based on the general design of MADA, an Arabic morphological analyzer and disambiguator, with additional components inspired by AMIRA (Pasha et al., 2014), a language independent SVM based analyzer. MADAMIRA improves upon the two systems and returns information selectively upon the request of the user. Sarf provides means for the developer to control and adapt the morphological analysis according to application needs. Moreover, the API enables the developer to implement high level applications such as NER which is provided by AMIRA.

Beesley (Beesley & Karttunen, 2003) compiles Xerox rules into specialized finite state machine (FSM) based morphological analyzer. The number of machines generated by a compiler for Xerox rules cannot be controlled by the developer of the analyzer, and the composition of the FSMs into a single framework is a difficult task (Beesley, 2001). Consequently the efficiency of the resulting analyzer depends on the way the Xerox rules are written. Writing application specific Xerox grammars and rules, or modifying the existing ones, requires deep knowledge and insight from the NLP application developer in compilation techniques, context free grammars, and morphological analysis. Sarf constructs a framework of efficient structures that encode the stems and the agglutinative and fusional affixes, respectively. The structures are traversed in a manner similar to the Xerox finite state machines, however, they are manually optimized to reduce the number of states and the number of non-deterministic transitions. Control on non-determinism is left to the compilers with Xerox. Sarf also provides an application customizable API that allows the developer to control the analysis. Doing the equivalent with Beesley requires the modification of the Xerox rules and the recompilation of the analyzer. Unlike Sarf, Beesley provides a root-pattern analysis of the stem.

Fassieh is a commercial Arabic text annotation tool that enables the production of large Arabic text corpora (Attia et al., 2009). The tool supports Arabic text factorization including morphological analysis, POS tagging, full phonetic transcription, and lexical semantics analysis in an automatic mode. Unlike Sarf, Fassieh provides morphological disambigua-

tion and root-pattern analysis. However, Fassieh does not provide segmentation of the affix and reports it as a whole unit. This tool is not directly accessible to the research community and requires commercial licensing. Sarf differs in that it is an open-source application customizable tool that solves the affix segmentation and 'run-on words' problems.

The work in (Attia, Toral, Tounsi, Pecina, & van Genabith, 2010) addresses the detection of Arabic Multi-word Expressions (MWE). They define MWEs as 'idiosyncratic interpretations that cross word boundaries or spaces'. Sarf adopts a similar approach for specific entities such as person names and place names.

Several researchers stress the importance of correspondence between the input string and the tokens of the morphological solutions. Some work uses POS tags and a syntactic morphological agreement hypothesis to refine syntactic boundaries within words (Lee et al., 2011). The work in (Grefenstette, Semmar, & Elkateb-Gara, 2005)(Nasredine et al., 2008) uses an extensive lexicon with 3,164,000 stems, stem rewrite rules (Darwish, 2002), syntax analysis, proclitics, and enclitics to address the same problem. Parallel traversal of the input string and the tokens of the morphological solution, while accounting for all possible SAMA normalizations, partially solves the problem as reported in (Kulick et al., 2010b; Maamouri et al., 2008). Later notes in the documentation of the ATB (Maamouri, Bies, Kulick, Krouna, et al., 2010) indicate that extensive manual work is still required and that later versions may drop the input tokens. (Lee et al., 2011) uses syntactic analysis to resolve the same problem.

The survey in (Al-Sughaiyer & Al-Kharashi, 2004) compares several morphological analyzers. Analyzers such as (Khoja, 2001)(Darwish, 2002) target specific applications in the analyzer itself or use a specific set of POS tags as their reference. Sarf differs in that it is a general morphological analyzer that reports all possible solutions. It is application customizable in the sense that the API is used to control and prioritize the analysis, refine the solution features, and associate morphemes with developer-defined categories.

The work in (Attia & Elaraby Ahmed, 2000; Beesley, 2001; Chaâben Kammoun et al., 2010) considers partial diacritics and performs morphological disambiguation by filtering the full morphological solutions and excluding inconsistent ones. This approach constructs several solutions that will be excluded later. Sarf considers diacritic consistency at the morpheme level instead of the final solution level. It checks for diacritic consistency between the input morpheme and the candidate VMF features at every accept node during the traversal of Sarf structures. Sarf analysis proceeds with the consistent VMFs and terminates the inconsistent ones.

(Beesley, 2001), (Chaâben Kammoun et al., 2010), and (Attia & Elaraby Ahmed, 2000) present analyzers that consider partial diacritics for morphological disambiguation. They filter the output morphological analysis based on compatibility with input diacritics if found. Sarf differs in that it considers the diacritics at morpheme boundaries to generate only the diacritic matching solutions, rather than generating all morphological solutions then filtering them.

4 Sarf

The flow diagram in Figure 1 illustrates the components of Sarf. The stem lexicon of Sarf L_s extends the lexicon of Buckwalter (Buckwalter, 2002) with proper and location names

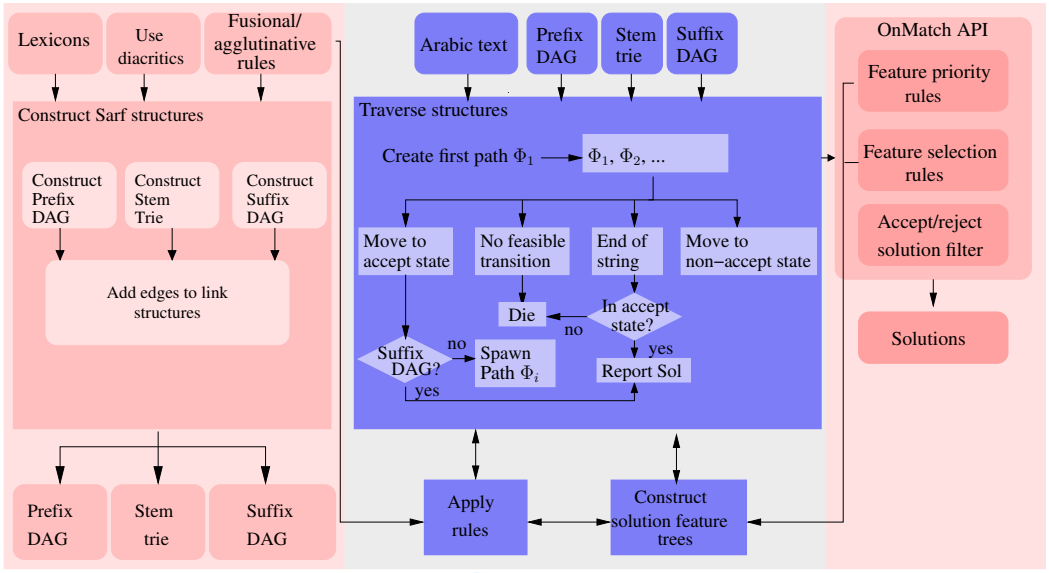

Fig. 1. Sarf flow diagrams: construction and traversal of morpheme structures, and generation of solutions.

extracted from different online sources as well as biblical sources. [3] The fusional and agglutinative affix rules encode affix-stem compatibility rules which are also taken from Buckwalter (Buckwalter, 2002) Additionally, they encode morpheme-morpheme concatenation rules to implement agglutinative morphemes and they are associated with fusional rules expressed in regular expressions where necessary. The use of diacritics to disambiguate morphological solutions is optional.

The *construct Sarf structures* process takes as input the lexicons, the fusional and agglutinative rules, and the "use diacritic" option and constructs directed acyclic graph (DAG) structures that encode the affixes, and a root index trie structure that encodes the stems (Aoe, 1989).

The *traverse structures* process reads the user-provided Arabic text in question one character at a time and traverses the Sarf structures accordingly. The traversal produces a sequence of morphemes each with its morpheme features, applies the fusional and agglutinative rules on each morpheme, checks for concatenation compatibility, and constructs morphological solution feature trees. The construction of the solution trees calls the `OnMatch API` at every morpheme match (control point) and takes into account the feature priority and selection rules defined by the NLP application developer. A `feature priority rule` is an order on features that is followed when constructing the morphological solution tree. Features with higher priority appear at higher levels in the solution tree. `A feature selection rule` defines the morphological features to be included in the morphological solution tree; other features are ignored in the construction of the solution. The traversal reports the con-

[3] http://alasmaa.net/ , http://ar.wikipedia.org/, Genesis 4:17-23; 5:1-32; 9:28-10:32; 11:10-32; 25:1-4, 12-18; 36:1-37:2; Exodus 6:14-25; Ruth 4:18-22; 1 Samuel 14:49-51; 1 Chronicles 1:1-9:44; 14:3-7; 24:1; 25:1-27:22; Nehemiah 12:8-26; Matthew 1:1-16; Luke 3:23-38

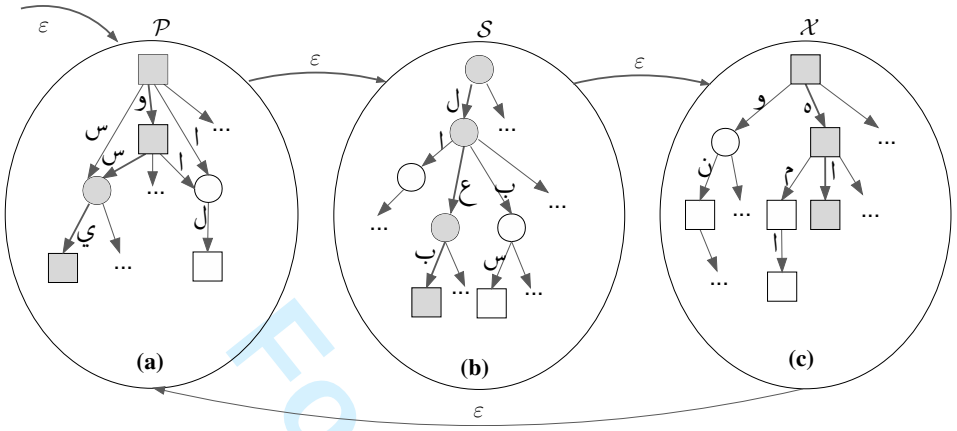

Fig. 2. Example affix DAGs and stem trie. The traversal of the shaded nodes corresponds to the final solution of وسيلعبهَا.

structed solution trees to a `filter` specified by the developer that either accepts or rejects the reported solutions.

4.1 Sarf structures

Sarf represents affix lexicons and rules using directed acyclic graphs as shown in Figure 2. This provides compactness as well as linear time traversal with respect to the input text. Algorithm `ConstructSarfDAGs` in Figure 3 describes how the structures are constructed from the lexicons and the rules. First, the algorithm creates efficient double array trie structures (Aoe, 1989) to represent the lexicons and benefit from the common sub-strings. A morpheme is represented as an accept-node in the trie structure. Then for each relation tuple in R_{pp}, R_{ps}, R_{sx}, and R_{xx} Sarf adds corresponding edges between the corresponding morphemes. Note that this transforms the affix trie structures into DAGs and keeps the stem structure as a trie since no stem-stem relations exist.

The `AddEdges` algorithm iterates over all relation tuples in R, and for each relation tuple $r = \langle c_1, c_2 \rangle$ it creates edges for all morphemes $\langle m_1, m_2 \rangle$ with compatibility categories c_1 and c_2, respectively.

In contrast, the Buckwalter analyzer considers all possible sub-strings and looks them up in affix hash tables, and performs several hash lookups in the stem hash tables in the order of all possible partitions of the input string. Sarf saves a binary version of the affix and stem structures which allows a fast loading time and only regenerates them if one of the lexicons and agglutinative and fusional rules are modified.

The diagram in Figure 2 illustrates the Sarf data structures. Subfigures (a), (b), and (c) in Figure 2 represent $\mathcal{P}$ the prefix DAG, $\mathcal{S}$ the stem trie, and $\mathcal{X}$ the suffix DAG, respectively. Boxes and circles denote morpheme versus non-morpheme nodes, respectively. The edges between nodes are labeled with input letters. Morpheme nodes contain the morphological features of the matching morpheme. These features include the VMF, gloss, POS, and compatibility information for morpheme concatenation.

```
AddEdges(R, T1, T2)
 foreach r = (c1, c2) ∈ R,
  foreach m1 ∈ T1 such that category(m1)= c1
   foreach m2 ∈ T2 such that category(m2)= c2
    createEdge(m1, m2);
   end
  end
 end
```

```
ConstructSarfDAGs(Lp, Ls, Lx,
        Rpp, Rps, Rsx, Rxx)
Tp = makeTrie(Lp)
Tx = makeTrie(Ls)
Ts = makeTrie(Lx)
AddEdges(Rpp, Tp, Tp)
AddEdges(Rps, Tp, Ts)
AddEdges(Rsx, Ts, Tx)
AddEdges(Rxx, Tx, Tx)
```

Fig. 3. Algorithm to construct Sarf structures.

When we reach a morpheme node in $\mathcal{P}$, $\mathcal{S}$, or $\mathcal{X}$, we proceed with the traversal in the next data structure. We use the symbol ϵ to refer to this transition. Invalid moves from a current node given an input letter denote the absence of a valid solution through this traversal path.

4.2 Running example: Sarf traversal

In this section, we illustrate how Sarf constructs morphological solutions from affix DAGs and stem trie using an example string وسيلعبهَا اللَّاعبون*wsylʿb-h-ā 'l-lāʿbwn* [4] (and they will play it) . The solution construction process also reveals how Sarf implements agglutinative and fusional affixes and how it handles 'run-on' words. Figure 2 highlights the node sequences of morphemes in وسيلعبهَا اللَّاعبون*wsylʿb-h-ā 'l-lāʿbwn* in the affix DAGs and stem trie. A square node (box) signifies reaching a complete morpheme and is referred to as a morpheme node, and an ϵ-edge is a transition move from one tree structure to another or a starting move to $\mathcal{P}$. It connects any morpheme node in the source tree to the root node in the destination tree.

In our example, Sarf parses the string وسيلعبهَا اللَّاعبون*wsylʿb-h-ā 'l-lāʿbwn* and traverses the tree structures using a traversal path Φ. Φ first uses the starting ϵ-edge and moves to the root square node in $\mathcal{P}$. A square node implies that an ϵ-edge from $\mathcal{P}$ to $\mathcal{S}$ is a valid transition, i.e. the morpheme sequence can start as a stem without a prefix . All edges in $\mathcal{P}$ starting from the root node with the current parsed character و*w* are also valid edges. Therefore, there are two valid moves, و*w* and ϵ. Φ hence spawns an exact copy of itself Ψ. Φ proceeds with و*w* in $\mathcal{P}$, and Ψ moves to $\mathcal{S}$ through ϵ. Each of Φ and Ψ represents a valid analysis path so far. A traversal path (Φ or Ψ) dies when it reaches a node of no valid moves. In our example, if there were no stems that start with the letter و*w* , Ψ dies. With the complete stem trie, Ψ actually dies when the input is at وسيلع*wsylʿ* as no stem word in the Arabic language is of such morpheme sequence. The Sarf DAGs allow agglutinative and fusional affixes. In our example, after the و*w* move, Φ reaches a square morpheme node. A copy of Φ gets created and proceeds with ϵ, and Φ proceeds with س*s* , in $\mathcal{P}$, being

[4] و س ي ل ع ب ه أ ا ل ل أ ع ب و ن in separate form to ease following the example

the next parsed character. The move of Φ leads to a round node signifying that ϵ move is invalid. Φ proceeds with يy and reaches a square node corresponding to compound prefix وسيwsy .

Now Φ transitions along ϵ, being the only valid move, into the root node of $\mathcal{S}$. Similar to its traversal in $\mathcal{P}$, Φ traverses moves corresponding to العl in $\mathcal{S}$. Before proceeding with بb and reaching a morpheme node, Φ checks if stem العبlb is compatible with the prefix وسيwsy . If the category of العبlb is compatible with the category of وسيwsy then Φ moves to a morpheme node. Otherwise, it moves to a non-morpheme node or dies. Sarf enables this by keeping compatibility category values as part of the morpheme nodes.Thus a morpheme node represents the whole morpheme sequence path leading to it within its tree.
Since Φ is now in a morpheme node in $\mathcal{S}$, it continues to traverse $\mathcal{S}$ and spawns a copy Ξ that moves to root node of $\mathcal{X}$. Φ dies when هـh is parsed since no هـ-edge from the current node exists in $\mathcal{S}$. On the other hand Ξ moves along هـh in $\mathcal{X}$ and reaches a valid analysis morpheme node.

A Sarf traversal considers a full word at any morpheme node in $\mathcal{X}$ and continues the traversal using an ϵ transition to the root node of $\mathcal{P}$. This solves the 'run-on' words problem. Consider there was no space between words اَللَّاعبون and وسيلعبهَا. The traversal will transition to the root of $\mathcal{P}$ when the word وسيلعبهَا is fully consumed, and then the traversal of اَللَّاعبون will resume. As for the other transitions from $\mathcal{X}$ morpheme nodes to the root of $\mathcal{P}$ before the completion of وسيلعبهَا, they will result in dead traversals and they will not be reported. Sarf reaches a solution and reports a valid traversal, including the morphological solutions of the 'run-on' words, when it reaches text delimiters, such as white space and punctuation, with valid segmentation of the input string.
In case of string with only stem sequences, empty transitions ϵ between $\mathcal{S}$ and $\mathcal{X}$, $\mathcal{X}$ and $\mathcal{P}$, and $\mathcal{P}$ and $\mathcal{X}$ to accept nodes allow parsing the string.

4.3 Morphological solution construction

inline,color=yellow]R2.4.This section describes how Sarf constructs the morphological solutions of an input string. We use وأكلهw-klh as an example input for illustration throughout the section. As a first step, Sarf processes the string input by the traversal algorithm

which results in two sequences of morpheme nodes where each sequence refers to a valid segmentation. The first sequence is ⟨ وw ,أ, كkl , هh ⟩ and the second sequence is ⟨وw , أكkl , هh ⟩. Figures 4 (a) to (d) show the first sequence and figures 4 (a),(e),(d) show the second sequence. Below, we describe how the morphological solution tree of every sequence is generated.

inline,color=yellow]R2.11 For every sequence, Sarf calls the developer-defined API at each morpheme node and constructs its solution tree according to the selected features and priority rules. Each path from the root of a morpheme solution tree to one of its leaves forms a morpheme solution path. Figures 4(a) to (e) show the constructed solution feature trees of the morphemes وw , أ , كkl , هh , and أكkl , respectively. The figures show that the API selects the gloss, POS, and VMF features with decreasing priority. Hence, the gloss is at first level of the tree, then POS follows, and finally VMF is at leaf level. Figure 4(f) illustrates an alternative solution feature tree of the morpheme أكkl with POS at highest priority followed by VMF and gloss. The comparison between the solution trees (e) and (f) shows how the priority rules can lead to the construction of smaller trees depending on the application at hand. For example, the solution feature tree (f) is more efficient if the developer is interested in the POS first. In such case, if Noun POS is required, only the right hand side branch of أكkl solution tree (f) is considered. This would be much simpler than the case of (e) where every single branch has to be checked to determine whether to include it or not in the solution. More details on the API are provided in Section 5.

Finally, Sarf composes the set of valid morphological solutions for the input string from solution paths that match the prefix-prefix, prefix-stem, stem-suffix, suffix-suffix, and prefix-suffix compatibility rules. For example, the first paths (paths to the leftmost leaves) of each of the solution trees in Figures 4(a), (b), (c), and (d) are compatible. The resulting morphological solution has the prefix وw with POS tag CONJ+, gloss tag and, and VMF tag وwa , the prefix أ with POS tag IVIS+, gloss tag I, and VMF tag أa , the stem كل with POS tag VERB_IMPERFECT, gloss tag trust/put in charge, and VMF tag كِkil , and the suffix ه with POS tag PVSUFF:DO:3MS, gloss tag him/it, and VMF هُho .

5 Application specific API

Sarf provides the developer with an application programming interface (API) that allows to (1) define developer categories and associate them with specific morphemes, (2) provide rules that prioritize and filter the solution features, and (3) control and refine the morphological analysis on the fly at solution *control points*. The control points are (1) agglutinative prefix matches, (2) stem matches, (3) agglutinative suffix matches, and (4) full solution matches.

To use the API, an NLP application developer is required to inherit from one or more of the prefix, suffix, stem and general stemmer C++ classes and implement the *OnMatch* interface. The developer configures the feature priority and selection rules by using the *setSolutionSetting* method to push the desired features into a feature configuration vector.

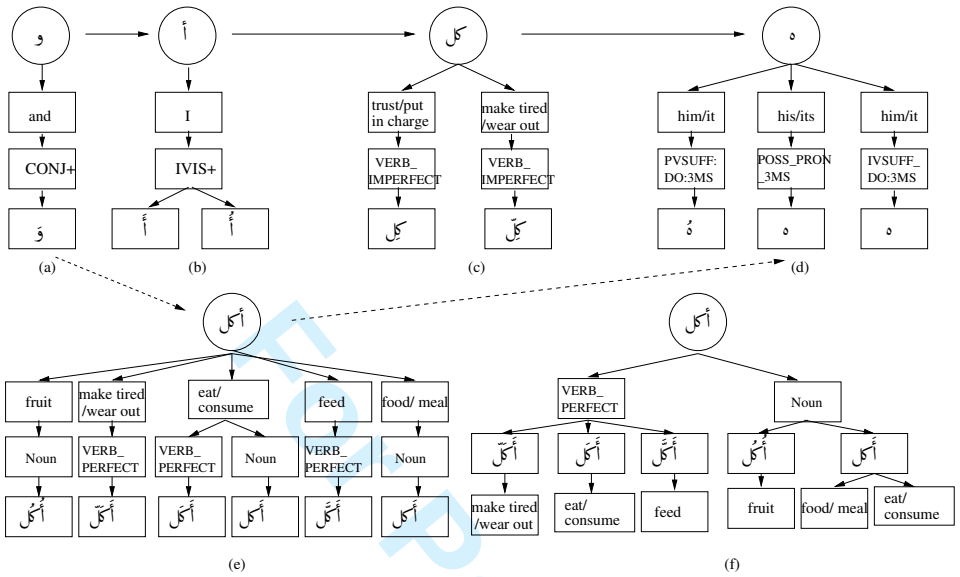

Fig. 4. Sample solution feature trees.

The computation returns the features present in vector in the solutions ordered by the same order in the vector. If the vector is empty, all features are considered.

The developer implements the *OnMatch* interface that Sarf calls at the control points. The developer processes the solution features provided at the control point and returns a value that instructs Sarf to (1) proceed with the analysis ignoring the current solution, (2) accept the solution and continue considering other solutions, and (3) accept the solution and stop the analysis. The developer can inspect at each control point the agglutinative morphemes and their compatibility category tags, the VMF, gloss, POS, lemma, and the developer-defined category tags.

Consider the task that aims to detect words with possible VERB POS tags. The API can be implemented to reject the analysis at the stem control point if the POS tag is not a VERB. This prevents the analyzer from computing insignificant full morphological solutions at early stages of the analysis. The developer can also use the feature selection filter API to disregard features such as VMF, gloss, and categories. This reduces the size of the solution trees and their corresponding traversal time. For example, given the word أكل, with gloss tags such as 'he/it eat' and 'fruit', Sarf typically returns nine possible morphological solutions with different VMF, POS tags, and gloss tags. With the feature selection filter API,

Sarf considers only two solutions with two solutions with the VERB_PERFECT and NOUN POS tags.

Sarf also provides the developer with the ability to alter the structure of the morphological solutions so that the traversal of the solutions is application specific. The developer can provide a factorization order of the solution features using the API priority rules. Sarf uses the priority rules to build the structures. This allows the developer to dismiss analysis earlier at the control points. Consider the task with an aim is to detect adjectives with positive sentiment. Since the POS feature is more limited than the gloss feature, it might give priority to the POS over the gloss. Hence, it can filter invalid solutions early in the analysis.

Moreover, Sarf enables the application developer to define categories and associate them with existing morphemes. Consider the task of detecting words that indicate family relations such as إبن (son), أب (father), and أم (mother). The developer can define a category called 'family connections' and associate it with the stems إبن, أب, أم or with their relevant glosses. The user defined categories are attached to the tags in the morpheme nodes as auxiliary tags that can be looked up in constant time.

6 Agglutinative and fusional morphemes

Sarf considers three types of affixes:

- *Atomic affix morphemes* such as يـ can be affixes on their own and can directly connect to stems using the R_{ps} and R_{sx} rules.
- *Partial affix morphemes* such as سـ can not be affixes on their own and need to connect to other affixes before they connect to a stem.
- *Compound affixes* are concatenations of atomic and partial affix morphemes as well as other smaller compound affixes. They can connect to stems according to the R_{ps} and R_{sx} rules.

Sarf forms compound affixes from atomic and partial affix morphemes using newly introduced prefix-prefix R_{pp} and suffix-suffix R_{xx} concatenation rules. Sarf considers L_p and L_x to be lexicons of atomic and partial affix morphemes associated with their tags. Sarf forms agglutinative affixes using prefix-prefix R_{pp} and suffix-suffix R_{xx} concatenation or agglutination rules. A rule $r \in R_{pp} \cup R_{xx}$ takes the compatibility category tags of affixes a_1 and a_2 and checks whether they can be concatenated. If so, the rule takes a_1 and a_2 and their tags and generates the affix $a = r(a_1, a_2)$ with its associated tags according to substitution rules based on regular expressions. The rules are fusional in the sense that they modify the orthography and the semantics of the resulting affixes by more than simple concatenation.

Category 1	**Category 2**	**Resulting Category**	**Substitution rules**
NPref-Bi	NPref-Al	NPref-BiAl	
NPref-Li	NPref-Al	NPref-LiAl	
			*r//*ال*||*ال**
Pref-Wa	none of { Pref-0, NPref-La, PVPref-La}	$2	
IVPref-li-	IVPref-*-y*	IVPref-(@1)-liy(@2)	*d//he|him*, *d//they||them* . . . *d//(+2)|| to*

Table 4. Example rules from R_{pp}

We illustrate this with the example rules in Table 4. Row 1 presents a simple rule that allows the concatenation of prefixes with category `NPref-Bi` such as بِ and كَ to prefixes with category `NPref-Al` such as ال, the result is the compound prefix with category `NPref-BiAl`. Since no substitution rule is specified, the tags of the resulting prefix are simple concatenations.

Row 2 presents a rule that takes prefixes with category `NPref-Li` such as لِ*li-* and prefixes with category `NPref-Al` such as ال. The substitution rule replaces the ال with ل resulting in لِلـ. The syntax of the substitution rule for the affix form is `r//(substring)||(replacement)\\`.

The rule in the Row 3 states that prefixes of category `Pref-Wa` can be concatenated with prefixes with categories that are neither of `Pref-0`, `NPref-La`, and `PVPref-La` categories. The resulting category is denoted with $2 which means the category of the second prefix.

Row 4 illustrates the use of the wild character '⋆' to capture sub-strings of length zero or more in the second category, and refers to the captured sub-strings in the resulting category using the '`@`' operator. The '`@`' operator is always followed by a number that denotes the captured '⋆' expression. Row 4 has also an example of substitution rules for the gloss (description) tag that start with the letter d. The `+2` pattern in the last substitution rule means that the `to` partial gloss description should be appended after the gloss of the second affix. Substitution rules for POS tags start with the letter p.

The rule introduced in row 4 replaces 24 prefix category rules in BAMA. Given the word لِيَأكل *lyʔakl* (for him/it to eat/consume), Sarf returns the solution with the prefixes ل (for) and ي (him/it), and the stem أكل (eat/consume). The prefixes ل and ي have the categories `IVPref-li-` and `IVPref-hw-ya`, respectively. The category `IVPref-li-` is the same as category 1 of the rule in row 4 and the category `IVPref-hw-ya` matches the pattern of category 2 of the same rule. The first wild character within the pattern captures the sub-string `hw` while the second wild character captures the sub-string `a`. Sarf recognizes that

those two affixes are compatible using this rule, and applies the corresponding expression to get the resulting category `IVPref-hw-liya` for the compound affix لي. Then, Sarf matches the result category of the compound affix with the category `IV_no-Pref-A` of the stem أكل and returns a morphological solution.

6.1 Building R_{pp} and R_{xx}

Our method is in line with native Arabic textbooks on morphology and syntax (AlRajehi, 2000a, 2000b; Mosaad, 2009) where only atomic and partial affixes are introduced. The textbooks also list rules to concatenate the affixes and discuss the syntax, semantic, and phonological forms of the resulting affixes. For example, the fourth rule in Table 4 is derived from a textbook rule that states `IVPref-li-` prefixes connect to all imperfect verb prefixes and transform the subject pronoun in the gloss to an object pronoun.

The method built the rules in four steps:

1. In the first step, we encoded textbook morphological rules into patterns.
2. In the second step, we inspected the BAMA and SAMA affix lexicons and extracted the atomic and partial affixes from them.
3. Then, we grouped the rest of the BAMA and SAMA affixes into the rules we collected from the textbooks.
4. We refined the rules wherever necessary, and we grouped rules that shared the same patterns.

We validated our work by generating all possible affixes and compared them against the BAMA and SAMA affix lexicons. The comparison resulted in discovering the BAMA and SAMA inconsistencies listed in Tables 5 and 6.

6.2 Redundancy

Consider the partial lexicon of prefixes in Table 2. The first five rows can be replaced with two atomic affix morphemes (ف*f* and ي*y*) and one partial affix morpheme (س*s*) in L_p, and three rules to generate compound morphemes in R_{pp} (r (ف*f* , ي*y*), r (ف*f* , س*s*), r (س*s* , ي*y*)). Representing prefix يَ*ya* (them/both) required four entries, three of them only differ in their dependency on the added يَ*ya* . Representing prefix و*w* required the addition of five entries. With Sarf, the equivalent addition of يَ*ya* (them/both) requires only two rules in R_{pp} that relate the categories of ف*f* and س*s* to that of ي*y* . The addition of و*w* requires only one additional entry (و*w*) in L_p. The difference is much larger when we consider the full lexicon as will be shown in Section 8.

	Affix	Vocalized	Inconsistent tag
a) missing plus in **gloss** tag of **prefix**	ف*fl*	فَل*fali*	and/so [+] for/to + the
	و بال*wbāl*	وَ بِال*wabiāl*	and + with/by [+] the
b) missing alternative **gloss** in **prefix**	ف*f*	فَ*fa*	and/so
	ف ب*fb*	فَ بِ*fabi*	and [/so] + with/by
	ف ك*fk*	فَ كَ*faka*	and [/so] + like/such as
c) gender/number qualifier omitted in **gloss** of subject **suffix**	ن*n*	ن*n*	they [fem.pl.] <verb>
	نهم*nhm*	نَهُم*nahom*	they [fem.pl.] <verb> them
	ات*āt*	ات*āt*	[fem.pl.]
	اتك*ātk*	اتَكَ*ātka*	[fem.pl.] your
d) also in **gloss** of object **suffix**	انهما*ānhmā*	انِهِمَا*ānihimā*	them (both)
	تكم*tkm*	تُكُم*tukum*	it/they/she <verb> you (pl.)
e) different ways to express them (both) in **gloss** of **suffix**	تهما*thmā*	تُهُما*tuhumā*	it/they/she <verb> them (both)
	اهما*āhmā*	اهُما*āhumā*	we <verb> (both of) them (both)
	ناهما*nāhmā*	ناهُما*nāhumā*	we <verb> (both of) them (both)
f) '.' omitted after pl in **gloss**	م*m*	مُ*um*	you [masc.pl.] <verb>
	ونا*wnā*	ونا*uwnā*	you [masc.pl[.]] <verb> us
g) **POS** tag is not same as vocalized	ته*th*	تِه*thi*	+ti/PVSUFF_SUBJ:2FS + [hu/] [hi/] PVSUFF_DO:3MS

Table 5. Sample BAMA inconsistencies

6.3 Inconsistencies

The entries in Tables 5 and 6 list examples of the 197 and 208 inconsistencies detected in the affix lexicons of BAMA version 1.2 and SAMA version 3.1, respectively. We found a small number of these inconsistencies manually and we computed the full list via comparing L_p and L_x with their counterparts computed using our agglutinative affixes. Most of the inconsistencies are direct results of partially redundant entries with erroneous tags. Experiments described later in section 8.2 shows that most of the detected inconsistencies are specifically related to gloss (Table 8). We note that SAMA corrected several BAMA inconsistencies, but also introduced several new ones when modifying existing entries to meet new standards and when introducing new entries.

The following describes the BAMA inconsistencies illustrated in Table 5:

(a) L_p omits a plus (+) symbol that indicates boundaries in compound prefixes.

	Affix	Vocalized	Inconsistent tag
a) missing standalone alef with no hamza **prefix** forms	*lā* لا	*ĀlA* آلا	I
	sā سا	*sa'au* سَأُ	I
b) additional by in **gloss**	*w* و	*wa* وَ	with
	wā- وال	*waā-* وَالا	with /by + the
c) additional space in vocalized form	*bf-* بف	*bfa-* بِفَ	
d) wrong **prefix**	*wf-* وف	*wafa-* وَفَ	and + so/and
e) missing definite indicator in **suffix gloss**	*āt* آت	*āti* آتِ	[fem.pl.] + [def.acc.]
	ātk اتك	*ātiki* اتِكِ	[fem.pl.] + [def.acc.] + your [fem.sg.]
	ātk آتك	*ātiki* آتِكِ	[fem.pl.] + [def. acc.] + your [fem.sg.]
f) omitted gender/num qualifier in **gloss**	*āk* اك	*ātāki* اتاكِ	we [verb] + you [fem.sg.]
	nhm نهم	*nahum* نَهُم	they [fem.pl.] [verb] + them
g) different ways to express them (both) in **gloss** of **suffix**	*thmā* تهما	*athumā* أَتهُما	it/they/she [verb] them (both)
	āhmā اهما	*āhumā* اهُما	we [verb] (both of) them (both)
	nāhmā ناهما	*nāhomā* ناهما	we [verb] (both of) them (both)
h) leftover BAMA style tags in **gloss**	*km* كم	*kum* كُم	he/it [verb] + you [masc.pl.]
	tkm تكم	*tukum* تُكُم	I [verb] + you (pl.) [masc.pl.]
	kn كن	*tukuna* تُكُنَّ	I [verb] + you (women) [fem.pl.]
i) indicative **gloss** with jussive **POS**	*nā* نا	*nā* نا	IVSUFF_MOOD:J, [ind.] [jus.] + us
	ny ني	*niy* نِي	IVSUFF_MOOD:J, [ind.] [jus.] + me
		u ُ	[def.nom.]
j) omitted '.' in **gloss**	*h* ه	*uhu* ُهُ	[def.nom .]
	tkn تكن	*atukuna* تَتُكُنَّ	[fem.sg.] + [def.nom .] + your [fem.pl .]
k) shadda inconsistent in **POS**	*y* ي	*ya* يَ	ya/POSS_PRON_1S
	y ي	*iyya* يِّ	. . . + ~a/ ya/ POSS_PRON_1S

Table 6. Sample SAMA inconsistencies

(b) L_p omits the (so) alternative gloss that corresponds to *bf-* بف in several compound prefixes.

(c) L_x omits gender and number qualifiers that appear within within square brackets from several glosses of subject suffixes.

(d) L_x omits gender and number qualifiers from several glosses of subject suffixes that appear within parenthesis.

(e) L_x expresses the dual quantifier as 'them (both)' in the majority of the entries, and as '(both of) them' in several entries.

(f) L_x omits the dot ('.') symbol from the gloss abbreviation of plural.

(g) L_x contains POS tags that are not consistent with the semantics of the vocalized tags for compound affixes.

The following describes the SAMA inconsistencies illustrated in Table 6:

(a) L_p misses entries for Alef prefixes with omitted hamza or madda due to relaxed writing standards which are common in many documents. This is resolved in SAMA for

standalone Alef prefixes via preprocessing tokens and flipping all forms of Alef into one form. We report it here since compound prefix entries with Alef are all listed, e.g. سَا*sā* , سأ*sʾa* , سآ*sā* , and the standalone prefixes are available but commented out. Sarf addresses this concern by the separation of prefixes into partial and atomic variants, and by introducing agglutinative and fusional rules. The rules construct compound prefixes from partial and atomic prefixes. This, therefore, enables the introduction of all prefixes that include Alef, with any of its alternative spellings, by only adding the Alef alternatives into the atomic prefix set.

(b) L_p contains an additional erroneous alternative gloss for *wa-* in only one compound prefix *wa-* ; while correctly not included elsewhere.

(c) L_p contains stray spaces in the vocalized tags of one of the فَ *fa-* alternatives.

(d) L_p contains an entry that supports the concatenation of *wa-* and فَ *fa-* conjunctions. This entry is erroneous and is illegal in Standard Arabic.

(e) L_x omits the definite indicator in the gloss of several suffixes.

(f) L_x omits gender and number qualifiers that appear within square brackets from the gloss tags of several suffixes.

(g) L_x expresses the dual quantifier as 'them (both)' in the majority of the entries, and as '(both of) them' in several other entries.

(h) L_x contains number indicators in the gloss tags still expressed in BAMA style.

(i) L_x contains entries with an *indicative* gloss mood and a *jussive* POS mood.

(j) L_x omits dot ('.') for the abbreviation of plural in several gloss tags.

(k) L_x represents a repeated consonant by a shadda in the POS tag where it should not. SAMA POS tags should spell out the repeated consonants if each belongs to one. In SAMA, the repeated consonant (of the shadda) is spelled out whenever the consonants has its separated partial POS tag.

In addition, 53 BAMA and 27 SAMA minor differences exist between L_p and L_x of BAMA and SAMA and their counterparts computed using our agglutinative affixes. For example, the BAMA gloss tags for prefixes that contain 'bi/PREP' report 'with/by' in some entries and its reverse 'by/with' in others. In addition, we detected several entries in L_p of SAMA with no category compatibility rules in R_{ps}, R_{sx}, and R_{px}.

7 Diacritics

Diacritics are short vowels that are often omitted in Arabic text and inferred by readers from context. Their omission adds to the ambiguity problem of Arabic morphological analysis. The diacritics ـَ *a* (*fatha*), ـُ *u* (*damma*), and ـْ represent and appears above the letter. 'a' vowel, 'o' vowel, and consonant, respectively, and appear above the letter. The diacritic ـِ *i* represents a 'y' vowel and appears below the letter. The diacritics ـً *an* , ـٌ *un* , and ـٍ *in* represent the 'a', 'o', and 'y' vowels followed by a phonetically stressed نْ *n* consonant.

The shadda ـّ mark is not a diacritic but is treated typographically as one, and is also often omitted in Arabic text. It denotes a repeated letter, first as consonant, and second as vocalized. Arabic forbids two letters with a consonant diacritic to follow each other.

Analyzers such as BAMA and SAMA ignore input partial diacritics because they consider them to be (1) rare in common corpora, and (2) unreliable because of dialect diversity and human errors (Attia, 2006; Elkateb et al., 2006). The diacritics can be miss-placed for several reasons including: (1) writers use a local Arabic dialect pronunciation of the word, (2) font effects such as ligatures compensate for miss-placing diacritics, and (3) writers sometimes use diacritics to decorate their text. However, the work in (Attia & Elaraby Ahmed, 2000; Beesley, 2001; Chaâben Kammoun et al., 2010) considers partial diacritics to decrease morphological ambiguity. The work in Maamouri, Bies, and Kulick (2006) inspected the ATB v3.2 corpus (Maamouri & Bies, 2004) for diacritics and found that 1.364% of the words were partially diacriticized and those diacritics eventually and *effectively* reduced morphological ambiguity. Most of the diacritics we found were tanween marks which indicate indefiniteness, followed in frequency by shadda, and then by dhamma that often indicates passive tense when associated with verbs. Hence, we decided not to force the use of partial diacritics to disambiguate the solutions and to provide it as an option.

Key to partial diacritic analysis is a diacritic-aware consistency check that replaces standard string matching checks. The `Diacritic-aware consistency check` algorithm takes as input two text chunks c_1 and c_2. It checks if the sequence of letters in $c1$ is the same as that in $c2$. It also checks the consistency of the sequence of diacritics in $c1$ and in $c2$.

Two sequences of diacritics are consistent iff:

1. Both are equal, or
2. One of the sequences is empty, or
3. If one has a shadda, then the other has no sukoun, or
4. If one has a shadda and the other has no shadda, then the rest of the diacritics are compared recursively.

If both checks were positive, then $c1$ and $c2$ are said to be equal with diacritic-aware consistency.

	word	string comparison					diacritic-aware consistency check				
(a)	أَكّلaakl	أ	ـَa	ك k	ـّ	ل l	أ	ـَa	ك k	ـّ	ل l
	↕	↕	↕	↕	↕	?	↕	↗	↗	↗	
	أَكَلakal	أ	ك k	ـَa	ل l		أ	ك k	ـَa	ل l	
(b)	أُكِيلaukil	أ	ـُu	ك k	ـِi	ل l	أ	ـُu	ك k	ـِi	ل l
	↕	↕	↕	↕	↕	?	↕	↗	↗	↗	
	أَكَلakal	أ	ك k	ـَa	ل l		أ	ك k	ـَa	ل l	

Table 7. Arabic string comparison with consideration of partial diacritics

Table 7 illustrates the `diacritic-aware consistency check` as compared to the standard string comparison with an example. Part (a) shows two diacritic-consistent words أَكّلaakl and أَكَلakal as a fatha ـَa is compatible with an empty diacritic and a shadda ـّ is compatible with a fatha ـَa . Part (b) illustrates two inconsistent diacritizations since ـِi is incompatible with ـَa next to the letter ك k .

Sarf takes an Arabic word as input with possible partial diacritics. Then, it traverses $\mathcal{P}$, $\mathcal{S}$, and $\mathcal{X}$ structures to compute the morphological solutions. For each matching morpheme in the structures, Sarf uses the `diacritic-aware consistency check` algorithm to check for the consistency of the fully diacriticized morpheme with the corresponding partially diacriticized chunk of the input word. The traversal path corresponding to the morpheme dies if the consistency check is negative (inconsistent).

8 Results

In this section we present and discuss the results of evaluating Sarf and compare it to existing morphological analyzers such as BAMA, SAMA, MADA+TOKAN, and ElixirFM.

We compared the segmentation capabilities of Sarf to that of SAMA and MADA+TOKAN under the ATB v3.2. We also evaluated the presence of the BAMA and SAMA inconsistencies of Tables 5 and 6 in the ATB v3.2 Part 3. Results show that the annotations of Sarf nearly perfectly agree with the manual annotations of the ATB v3.2. The results also show that Sarf can automatically correct 0.76% of the ATB annotations due to lexicon inconsistencies.

Moreover, we evaluated the efficiency of Sarf by measuring the size of the lexicons, lexicon augmentation cost, and the runtime and accuracy performance of the analyzer.

We compared the cost of augmenting Sarf with the question clitic (hamza أ) to that of BAMA and SAMA. We also conducted two experiments to evaluate the performance of Sarf compared to SAMA and ElixirFM. The experiments show the advantage of Sarf over the other analyzers in terms of performance.

8.1 Segmentation correspondence

We evaluated the segmentation correspondence capabilities of Sarf under the segmentation guidelines of the ATB v3.2. For each entry in the 'before' section of ATB v3.2 Part 3 with a correct SAMA solution, we automatically computed the segmentation using Sarf and compared the result to the segmentation in the 'after' entries that are manually validated by the LDC.

SAMA had a correct morphological solution for 273,618 words out of 393,201 ATB words. Those required later segmentation and manual validation. Our automatically generated segmentation agree with 99.991% with the oracle ATB segmentation for the 273,618 words. We inspected the 25 entries for which our segmentation disagreed with ATB and found that both segmentations were valid. For example, the entry مِنّا*mnā* is formed of من `min/PREP` (from) and نا `nA/PRON_1P` (us). Our segmentation is مِن+نَّا while that of ATB is مِنّا. When the morphemes are concatenated, the two نـ consonants can be fused into a single one with a shadda (ـّ). Since it is common to omit the shadda, we are left with a single consonant at the boundary, which can correspond to either morpheme.

Since Sarf preserves correspondence when performing segmentation, it is capable of generating a vocalized tag in the ATB 'after' dataset which carries more information in 15.47% of the time than the counterpart POS derived vocalized entry. The POS derived ATB vocalization processes, in some cases, the source token by splitting it into multiple unvocalized POS tokens which are then converted into vocalized ATB tokens. For example, the word للقضاء*llqḍāʾ* (for/to the justice/judiciary) is split into `li/PREP + Al/DET + qaDA'/NOUN` tokens. In such cases, the POS tokens differ from the source token. This issue does not exist in Sarf and the vocalized entry is maintained.

8.2 Lexicon consistency

We evaluated the presence of the inconsistencies of Tables 5 and 6 in the ATB v3.2 Part 3. The first experiment considered the ATB entries that adopted the SAMA solution. The rest of the entries have manually entered solutions. The gloss inconsistencies affect 0.76% of those entries.

The second experiment considered all tokens in the ATB with a SAMA solution. The Δ_{word} column of Table 8 reports the ratio of the affected ATB words, and the $\Delta_{analysis}$ columns reports the ratio of the conflicting morphological analyses. One word might have several morphological solutions which explains the difference. The rows report the effect of the POS and gloss tags, and that of the wrong prefix entry d of Table 6. In total 8.774% of the words and 3.264% of the morphological solutions are affected. Sarf automatically solves all these conflicts.

Conflict	Δ_{word}	$\Delta_{analysis}$
POS	0.206%	0.016%
Gloss	8.317%	3.226%
WaFa	0.251%	0.022%
Total	**8.774%**	**3.264%**

Table 8. Effect of lexicon inconsistencies.

	$\|L_p\|$	$\|R_{pp}\|$	$\|L_x\|$	$\|R_{xx}\|$	$\Delta_L^{\text{ء}}$	$\Delta_R^{\text{ء}}$
BAMA	299	–	618	–	295	–
Sarf: Agglutinative only	70	89	181	123	1	32
Sarf: Aggl. & fusional only	43	89	146	128	1	32
Sarf: Aggl., fusional, & grouping	41	7	146	32	1	1
SAMA	1325	–	945	–	1,296	–
Sarf: Agglutinative only	107	129	221	188	1	38
Sarf: Aggl. & fusional only	56	129	188	194	1	38
Sarf: Aggl., fusional, & grouping	53	18	188	64	1	1

Table 9. Comparison of lexicon size between BAMA, SAMA, and Sarf. Sarf is represented as BAMA or SAMA with Agglutinative, fusional, & grouping.

8.3 Lexicon size.

The $|L_p|$, $|L_x|$, $|R_{pp}|$, and $|R_{xx}|$ entries in Table 9 report the size of the prefix lexicon, the size of the suffix lexicon, the number of prefix-prefix concatenation rules , and the number of suffix-suffix concatenation rules , respectively for BAMA and SAMA. The entries also report the effect of using agglutinative affixes, fusional rules, and grouping of rules with similar patterns using wildcards on reducing the size of the lexicon.

The size of the affix lexicons of Sarf is obtained by applying agglutinative affixes, fusional rules, and grouping of rules on BAMA and SAMA affix lexicons. Sarf only requires 226 entries to represent the 917 entries of BAMA affixes with inconsistencies corrected. Sarf entries include 41 prefixes, seven prefix-prefix rules, 146 suffixes, and 32 suffix-suffix rules while BAMA affixes include 299 prefixes and 618 suffixes. Similarly, Sarf only requires 323 entries to represent 2,270 entries of SAMA affixes with inconsistencies corrected.

The transition from BAMA to SAMA required the addition of 1,353 entries to the lexicons of BAMA. Sarf only required the addition of one order of magnitude less entries to

accommodate an equivalent change. The 136 entries consisted of 12 more entries in L_p, 42 in L_x, 18 rules in R_{pp}, and 64 in R_{xx}.

Augmentation. The question clitic, denoted by the glottal sign (hamza أ *a*), is missing in BAMA and SAMA as noted by (Attia, 2006). The Δ_L and Δ_R entries in Table 9 show the difference in the number of additional affixes and rules needed to accommodate for the addition of the question clitic. Our method only requires the addition of one atomic affix and one fusional concatenation rule. Whereas BAMA and SAMA need 295 and 1,296 additional entries to their lexicons, respectively, Moreover, the process requires manual intervention with a possibility of inducing inconsistencies in the process. This is evidence that our method is better for the consistency and the maintenance of the lexicons.

8.4 Performance

We evaluated the speed of Sarf and compared it to that of SAMA and ElixirFM. We selected a random article from an Arabic newspaper containing 1,122 words and computed the morphological solutions of those words using the three analyzers. Sarf with no application specific API implemented generates all the solutions of all words in 0.72 seconds while SAMA does the task in 2.46 seconds. However, ElixirFM requires much more time with 34 minutes and 19 seconds.

We also evaluated the accuracy and runtime efficiency of Sarf and compared that to SAMA and ElixirFM with one of the applications (Zaraket & Makhlouta, 2012a) that used Sarf as a back-end morphological analyzer. The hadith extraction application (Zaraket & Makhlouta, 2012a) is concerned with analyzing a book of traditions related to prophet Mohammad through a chain of narrators. Narrators are identified by composite proper person names that are connected with family connectors. For example, the chain of narrators حدّثنَا قتيبة بن سعيد عن جرير *ḥddṯnā qtybh bn sʿyd ʿn ǧryr* , starts with the first narrator قتيبة بن سعيد *qtybh bn sʿyd* where قتيبة *qtybh* and سعيد *sʿyd* are `proper person names`. The word بن *bn* (son of) indicates a parental relation that we will refer to as `family-connectors`. The name جرير *ǧryr* is a proper person name denoting the second narrator, and the words حدّثنَا *ḥddṯnā* (told us) and عن *ʿn* (from/about) indicate a narration relation and we refer to them as `tell-connectors`. Key to narrator detection are morphological features that point to places such as names and location prepositions.

Table 10 reports the results of detecting morphological features that define proper names, tell-connectors, and family connectors and compares the results with SAMA and ElixirFM in terms of accuracy and running time. The table considers three books of hadith selected arbitrarily (Al Kulayni, 1996; Al Tousi, 1995; Ibn Hanbal, 2005)[5]. All experiments used a Linux operating system running on a dual core 2.66 Ghz 64-bit processor with 4GB of memory.

Sarf scored higher recall for all the features and approximately similar precision across the three books. The precision and recall measures of the family connectors in Sarf and SAMA are close, unlike ElixirFM which reports lower accuracy measures. The precision for proper names is low since some proper names exist in the books but outside the context

[5] We obtained the digitized books from online sources such as http://www.yasoob.com/ and http://www.al-eman.com/.

		Al Kafi			Al Istibsar			Ibn Hanbal		
		Sarf	SAMA	ElixirFM	Sarf	SAMA	ElixirFM	Sarf	SAMA	ElixirFM
Proper Names	precision	0.36	0.36	0.42	0.38	0.35	0.37	0.53	0.55	0.64
	recall	0.95	0.83	0.77	0.96	0.81	0.73	0.98	0.79	0.76
Tell connectors	precision	0.86	0.85	0.84	0.91	0.9	0.92	0.95	0.93	0.95
	recall	0.99	0.99	0.99	0.99	1	1	1	1	1
Family connectors	precision	0.91	0.90	0.78	0.91	0.91	0.77	1	1	0.97
	recall	1	1	0.41	1	1	0.42	1	1	0.69
Total	precision	0.51	0.52	0.56	0.57	0.56	0.55	0.69	0.72	0.78
	recall	0.97	0.92	0.77	0.98	0.92	0.74	0.99	0.90	0.83
Time	(secs)	1.32	6.65	2.78×60^2	1.31	4.55	2.3×60^2	0.096	0.66	29.2×60

Table 10. Comparison of Sarf to SAMA and ElixirFM using the hadith application

		Temporal	Hadith	Biography	Genealogy
	Words	125,010	18,047,732	14,710,064	21,385
Without API	Solutions	4.33	5.41	5.61	4.63
	Time (secs)	12.45	45.21×60	133.17×60	2.89
With API	Solutions	1.74	1.82	2.12	2.44
	Time (secs)	2.39	22.64×60	31.77×60	0.41

Table 11. Morphological solutions per word ratio and runtime gains with the customizable Sarf API utility.

of a chain of narrator names and are hence not wanted. After analyzing the results, it turned out that ElixirFM misses some gloss tags such as the 'son' tag associated with the stem '*bn*بن'. Sarf produces significantly higher proper name recall measure compared to SAMA and ElixirFM. This mainly due to augmenting the stem lexicon of Sarf with proper names as explained in Section 4.

Sarf outperformed both SAMA and ElixirFM in running time even without the use of the feature priority and the feature selection API. SAMA performed better than ElixirFM.

8.5 Sarf API

We evaluated the application customizable Sarf API with several applications that used Sarf. Table 11 reports the number of morphological solutions per word and the runtime of Sarf for several applications before and after using the application customizable Sarf API. The numbers show the utility of the Sarf API at improving the runtime and the efficiency of NLP applications by an order or magnitude across the four applications.

In what follows, we shortly describe the applications. Detailed results including accuracy measures can be found in the corresponding papers.

The temporal entity extraction application uses finite state machines driven by morphological features that indicate temporal units, intervals, quantities, and prepositions as input (Zaraket & Makhlouta, 2012c). The application processed 43 articles arbitrarily selected from local newspapers. The average number of solutions that Sarf reported per word without the use of the API was 4.33 solutions per word. The application specific refinements of the analysis implemented using the Sarf API eliminated solutions that the application is not interested in and thus the number of solutions per word that Sarf ended up reporting was 1.74 and that in turn resulted in a substantial improvement in runtime.

The hadith segmentation and extraction application extracts chains of narrators from hadith books using finite state machines that take morphological features such as gloss and POS tags that indicate proper names, family relations, tell-connections, places, and possessive nouns as input (Zaraket & Makhlouta, 2012a). The application processed a total of 41 books with a total of 196,171 narrations to build a graph where narrators are nodes and their relation to each other are edges. Similarly to above, the number of solutions per word improved from 5.41 to 1.82 and the runtime improved by more than half.

The biography application matches a narrator extracted from the hadith application to corresponding biographies in biography books and extracts entities such as birth and death dates, location, students, professors, and authentication qualifications (Zaraket & Makhlouta, 2012a). The biography application uses narrator extraction and temporal extraction, and in addition it uses morphological features that indicate qualifying adjectives that relate to authenticity. The application used the graph generated from the hadith application and processed 15 books of biographies with a total of 79,946 biographies to (1) segment the biographies, (2) extract the narrators in the biographies, and (3) annotate the narrator nodes in the graph with qualifiers extracted from the corresponding biographies. The use of the Sarf API improved the solutions per word ratio from 5.61 to 2.12 and improved the run time by almost a factor of 4.

The genealogy extraction application extracts a family tree and learns words that indicate family relations from biblical texts. It uses morphological features that indicate proper names, family relations, places, and professions (Makhlouta et al., 2012). The application processed the book of Genesis with fifty verses. 21,385 words. The use of Sarf API improved the solutions per word ratio from 4.63 to 2.44 and improved run time substantially.

9 Conclusion

This paper presents Sarf, an application customizable Arabic morphological analyzer. NLP applications can implement the Sarf API to to refine the morphological analysis on the fly by (1) selecting the interesting features, (2) prioritizing the features, and (3) accepting and rejecting solutions on the fly based on partial features reported so far. Sarf extends the SAMA and BAMA lexicons and represents affixes using agglutinative and fusional morphemes which (1) significantly reduces the size of the lexicons needed to represent Arabic morphology,(2) fixes inconsistencies in morphological features corresponding to morphemes, (3) simplifies the maintenance and augmentation of the affix morpheme lexicons, and (4) solves the segmentation correspondence problem between the morphological solution and the original text. Sarf also allows the NLP application to use partial diacritics for morphological solution disambiguation, and solves the ‘run-on words’ problem. Sarf

is available online as an open source tool. We evaluated Sarf in terms of effectiveness and performance with respect to other tools such as SAMA, BAMA and ElixirFM. Sarf outperforms other analyzers in run-time and slightly increases recall rates in morphological analysis due to the extended lexicons. We used Sarf application specific API in several NLP applications for information extraction from Arabic documents and the applications provided more accurate results than existing analyzers with faster running time.

In the future, we plan to improve Sarf by allowing root analysis of stems, supporting inflectional stems, and providing a graphical user interface to allow the users to edit the affix and stem morpheme lexicons of Sarf.

References

Al Kulayni, M. (1996). *Kitab al-kafi*. Taaruf.

Al Tousi, M. (1995). *Al istibsar*. Taaruf.

AlRajehi, A. (2000a). *The morphological practice/at-tatbiq assarfi* (first ed.). Renaissance (An-nahda).

AlRajehi, A. (2000b). *Semantical practice/at-tatbiq alnahawi* (first ed.). Renaissance (nahda).

Al-Sughaiyer, I., & Al-Kharashi, I. (2004). Arabic morphological analysis techniques: a comprehensive survey. *American Society for Information Science and Technology*, *55*(3), 189–213.

Aoe, J.-i. (1989). An efficient digital search algorithm by using a double-array structure. *IEEE Transactions on Software Engineering*, *15*(9), 1066–1077.

Attia, M. (2006). An ambiguity-controlled morphological analyzer for modern standard Arabic modelling finite state networks. In *The challenge of Arabic for nlp/mt conference.* The British Computer Society.

Attia, M., & Elaraby Ahmed, M. (2000). *A large-scale computational processor of the Arabic morphology* (Unpublished master's thesis). Faculty of Engineering.

Attia, M., Rashwan, M., & Al-Badrashiny, M. (2009). Fassieh, a semi-automatic visual interactive tool for morphological, pos-tags, phonetic, and semantic annotation of Arabic text corpora. *Audio, Speech, and Language Processing, IEEE Transactions on*, *17*(5), 916–925.

Attia, M., Toral, A., Tounsi, L., Pecina, P., & van Genabith, J. (2010). Automatic extraction of Arabic multiword expressions. In *Proceedings of the workshop on multiword expressions: from theory to applications (mwe 2010)* (pp. 18–26). Beijing, China: Association for Computational Linguistics.

Badawi, E., Carter, M., & Gully, A. (2004). *Modern written Arabic: A comprehensive grammar*. New York: Routledge.

Beesley, K. (2001). Finite-state morphological analysis and generation of Arabic at xerox research: Status and plans. In *Workshop proceedings on Arabic language processing: Status and prospects* (pp. 1–8). Toulouse, France.

Beesley, K., & Karttunen, L. (2003). *Finite-state morphology: Xerox tools and techniques*. Stanford: CSLI.

Benajiba, Y., Rosso, P., & Benedíruiz, J. (2007). Anersys: An Arabic named entity recognition system based on maximum entropy. In *Computational linguistics and intelligent text processing* (pp. 143–153). Springer.

Buckwalter, T. (2002). *Buckwalter Arabic morphological analyzer version 1.0* (Tech. Rep.). LDC catalog number LDC2002L49.

Buckwalter, T. (2004). Issues in Arabic orthography and morphology analysis. In *Semitic '04: Proceedings of the workshop on computational approaches to Arabic script-based languages* (pp. 31–4). Morristown, NJ, USA.

Chaâben Kammoun, N., Hadrich Belguith, L., & Ben Hamadou, A. (2010). The morph2 new version: A robust morphological analyzer for Arabic texts. In *Jadt 2010: 10th international conference on statistical analysis of textual data.*

Darwish, K. (2002). Building a shallow Arabic morphological analyzer in one day. In *Proceedings of the acl-02 workshop on computational approaches to semitic languages.*

Elkateb, S., Black, W., Vossen, P., Farwell, D., Rodríguez, H., Pease, A., & Alkhalifa, M. (2006). Arabic wordnet and the challenges of Arabic. In *Proceedings of Arabic nlp/mt conference, london, uk.*

Grefenstette, G., Semmar, N., & Elkateb-Gara, F. (2005). Modifying a natural language processing system for european languages to treat Arabic in information processing and information retrieval applications. In *Acl workshop on computational approaches to semitic languages* (pp. 31–37).

Habash, N. (2010). Introduction to Arabic natural language processing. *Synthesis Lectures on Human Language Technologies*, *3*(1), 1–187.

Habash, N., Rambow, O., & Roth, R. (2009). Mada+tokan: A toolkit for Arabic tokenization, diacritization, morphological disambiguation, POS tagging, stemming and lemmatization. In K. Choukri & B. Maegaard (Eds.), *Proceedings of the second international conference on Arabic language resources and tools.* Cairo, Egypt: The MEDAR Consortium.

Habash, N., & Sadat, F. (2006). Arabic preprocessing schemes for statistical machine translation. In *Proceedings of the north american chapter of the association for computational linguistics(naacl)* (pp. 49–52).

Hunspell manual page. (2012). `http://www.linuxcertif.com/man/4/hunspell`.

Ibn Hanbal, A. (2005). *Musnad ahmad*. Noor Foundation.

Jaber, A., & Zaraket, F. (2013). MATAr: Morphology-based tagger for Arabic. In *Aiccsa.* Fes, Morocco.

Khoja, S. (2001). Apt: Arabic part of speech tagger. In *Naacl student research workshop.*

Kulick, S., Bies, A., & Maamouri, M. (2010a). Consistent and flexible integration of morphological annotation in the Arabic treebank. In *Proceedings of the seventh conference on international language resources and evaluation (lrec'10).* Valletta, Malta.

Kulick, S., Bies, A., & Maamouri, M. (2010b). Consistent and flexible integration of morphological annotation in the Arabic treebank. In *International conference on language resources and evaluation.* European Language Resources Association.

Lee, Y. K., Haghighi, A., & Barzila, R. (2011). Modeling Syntactic Context Improves Morphological Segmentation. In *Conference on computational natural language learning (conll).*

Legally, K. (2004, March). *Arabtex: Typesetting Arabic and hebrew, user manual version 4.00* (Technical Report 2004/03). Fakultat Informatik, Universitat Stuttgart.

Maamouri, M., & Bies, A. (2004). Developing an Arabic treebank: methods, guidelines, procedures, and tools. In *Semitic '04: Proceedings of the workshop on computational approaches to Arabic script-based languages* (pp. 2–9).

Maamouri, M., Bies, A., & Kulick, S. (2006). Diacritization: A challenge to Arabic treebank annotation and parsing. In *The british computer society arabic nlp/mt conference.*

Maamouri, M., Bies, A., Kulick, S., Krouna, S., Gaddeche, F., & Zaghouani, W. (2010). Arabic treebank: Part 3 version 3.2. In *Linguistic data consortium,ldc2010t08.*

Maamouri, M., Bies, A., Kulick, S., Zaghouani, W., Graff, D., & Ciul, M. (2010). From speech to trees: Applying treebank annotation to Arabic broadcast news. In *Proceedings of the seventh conference on international language resources and evaluation (lrec'10).* Valletta, Malta.

Maamouri, M., Graff, D., Bouziri, B., Krouna, S., Bies, A., & Kulick, S. (2010). Ldc standard Arabic morphological analyzer (sama) version 3.1. In *Ldc2010l01. web download.*

philadelphia: Linguistic data consortium.

Maamouri, M., Graff, D., Bouziri, B., Krouna, S., & Kulick, S. (2010). Ldc standard Arabic morphological analyzer (sama) v. 3.1. *LDC Catalog No. LDC2010L01. ISBN*.

Maamouri, M., Kulick, S., & Bies, A. (2008). Diacritic annotation in the Arabic treebank and its impact on parser evaluation. In *International conference on language resources and evaluation.*

Makhlouta, J., Zaraket, F., & Harkous, H. (2012). Arabic entity graph extraction using morphology, finite state machines, and graph transformations. In *Cicling.*

Mosaad, Z. (2009). *The briefing of morphology* (first ed.). As-Sahwa.

Nasredine, S., Laib, M., & Fluhr, C. (2008). Evaluating a natural language processing approach in Arabic information retrieval. In *Elra workshop on evaluation.*

Nizar Y., H. (2009). *Introduction to Arabic natural language processing*. Morgan & Claypool.

Pasha, A., Al-Badrashiny, M., El Kholy, A., Eskander, R., Diab, M., Habash, N., Pooleery, M., Rambow, O., & Roth, R. (2014). Madamira: A fast, comprehensive tool for morphological analysis and disambiguation of Arabic. In *Proceedings of the ninth international conference on language resources and evaluation.* Reykjavik, Iceland: European Language Resources Association.

Smrž, O. (2007). Elixirfm: implementation of functional Arabic morphology. In *Semitic '07: Proceedings of the 2007 workshop on computational approaches to semitic languages* (pp. 1–8). Prague, Czech Republic.

Spencer, A. (1991). *Morphological theory: An introduction to word structure in generative grammar* (Vol. 2). Basil Blackwell Oxford.

Vajda, E. J. (2001). *Typology.* `http://pandora.cii.wwu.edu/vajda/ling201/test1materials/typology.htm`. Western Washington University.

Zaraket, F., & Makhlouta, J. (2012a). Arabic cross-document NLP for the hadith and biography literature. In *Flairs.*

Zaraket, F., & Makhlouta, J. (2012b). Arabic morphological analyzer with agglutinative affix morphemes and fusional concatenation rules. In *Coling.* Mumbai, India.

Zaraket, F., & Makhlouta, J. (2012c). Arabic temporal entity extraction using morphological analysis. *IJCLA*, *3*, 121-136.