Description of the Auto-Alignment Algorithm

Tim Sauerwein / 2020-Jan-01

Table of Contents

[1 Purpose and Scope 2](#_Toc60409497)

[2 Overview 2](#_Toc60409498)

[3 Concepts 3](#_Toc60409499)

[3.1 Source and Target Words 3](#_Toc60409500)

[3.2 Source and Target Positions 4](#_Toc60409501)

[3.3 Zones 4](#_Toc60409502)

[3.4 Alignment 5](#_Toc60409503)

[3.5 Translation Model 5](#_Toc60409504)

[3.6 Estimated Alignment Model 6](#_Toc60409505)

[3.7 The Syntax Tree 6](#_Toc60409506)

[3.8 Descriptive Collateral 7](#_Toc60409507)

[3.9 Strongs Codes 7](#_Toc60409508)

[3.10 Link Checking Experience Database 8](#_Toc60409509)

[4 Tree-Based Auto Alignment 8](#_Toc60409510)

[5 Aligning a Zone 11](#_Toc60409511)

[6 Identifying Source Point Candidates 12](#_Toc60409512)

[7 Identifying Tree-Node Candidates 13](#_Toc60409513)

[8 Finishing the Auto-Alignment 15](#_Toc60409514)

[9 Further Details 16](#_Toc60409515)

# Purpose and Scope

This document describes the auto-alignment algorithm that is implemented in the Clear3 prototype at the time of writing. This algorithm is essentially unchanged from Clear2.

# Overview

Diagram

Description automatically generated

The Clear3 auto-alignment algorithm analyzes a translated target text. The algorithm compares the target text to syntax trees for the original-language source text of the Bible. The algorithm uses the verses in the target text to match up the target text to the syntax trees. The algorithm also uses some auxiliary inputs, such as optional information about a former alignment and a specification of which source words are content words.

The analysis produces its best estimate of the links between source words and target words, where each link is accompanied by a confidence score. The result also contains descriptive information about the source and target words.

Here is a simplified overview of the steps in the Clear3 Auto Alignment:

Diagram

Description automatically generated

The steps are as follows:

* Analyze the verses in the target text to find corresponding source words. The result is a set of verse pairs, where each pair has some source words, and also some target words that together translate the source words. (As part of this step, it is also necessary to break the target text down into target words.)
* Train a statistical-translation-model using the verse pairs. (At present, the model is one implemented from research published by IBM, but we are considering alternatives.) The result of training is an estimated translation model (which gives the probable meanings of source words in terms of target words) and an estimated alignment model (which gives the probable links between source positions and target positions).
* Create a final alignment using the estimates from the previous step along with the syntax trees. This final step is the heart of the Clear3 algorithm and the principal focus of this paper.

# Concepts

This section presents some concepts that form a basis for explaining the auto-alignment algorithm in more detail.

## Source and Target Words

The translated text and the original-language source text are both regarded as a sequence of *words*. Usually the term *word* is used to denote the smallest linguistic unit of meaning. But note that in some languages (such as Hebrew), the smallest unit can actually be a portion of what is usually thought of as a word. So sometimes the term *segment* is used instead, and for Hebrew we speak of a segment within a word. For a language like Greek, a segment and a word are the same thing.

For the original-language sources we know the words and segments, but for the translated text we might not. The process that breaks the translated text down into words is called *segmentation* (and also *tokenization*).

Clear3 contains the segmentation algorithm from Clear2. The addition of other segmentation algorithms is a possible future activity.

Each word has a *surface form*, which is how the word appears in context and reflects the linguistic rules for using the word. The word also has a *lemma*, which is its dictionary form. There can be a variety of different surface forms with the same lemma.

We know the lemmas for the source words, but not for the target words. Clear3 uses a strategy of converting target words to lower case as a partial substitute for knowledge of target lemma forms.

## Source and Target Positions

Each word that occurs in the source and target does so in a certain position. These positions are described by special identification data, as follows.

A *source ID* is a datum that identifies a position in the source manuscript by giving its book number, chapter number, verse number, word number, and segment number. (In the case of a Greek word, the segment number is always 1.)

A *target ID* is a datum that identifies a position in the translated text by giving its book number, chapter number, verse number, and word number. (Clear3 looks at the translated text as having words but not subsegments.)

## Zones

The overview in Section 2 above spoke of verse pairs, but really a more general concept is needed. This need arises because sometimes a translation joins multiple verses together or splits verses apart. Clear3 introduces the more general concept of *zone* to mean some portion of the source (a *source zone*) or the target (a *target zone*). So, the overview could have been more precise by speaking of a zone pair instead of a verse pair.

In most cases, we expect that a zone corresponds to a verse, but it need not be so.

The general problem of finding the zones in the source and the target and matching them up properly is called *versification*. Our experience has been that simple versification algorithms are often inadequate, so this area is a likely one for future development.

Clear3 reproduces the simple approach to versification found in Clear2. This approach involves consulting a versification data file (conventionally called “Versification.xml”), and in the end results in source zones that are always contiguous verses.

A *source point* is a source word when regarded as a member of a particular source zone. The source points of a source zone are regarded as ordered by the way they occur in the source manuscript. So, we can think of a source zone as an ordered sequence of source points. We think of a source point as having a source ID (that gives its position in the source manuscript), a lemma, a surface form, an index number within the source zone, and so on.

In the same way, a *target point* is a target word when regarded as a member of a particular target zone, and we think of a target point as having a target ID, a surface form, a lower-cased form, an index number within the target zone, and so on.

There is an alternate way of identifying a particular source or target point called the *alternate ID*, which is a string datum of the form “xxx-nn”. In this representation, the “xxx” part of the string is the surface form of the point, and the “nn” part of the string is a sequence of decimal digits that numbers the occurrence of the surface form. For example, an alternate ID of “λόγος-2” could mean the second occurrence of the surface form “λόγος” within a source zone.

The advantage of using an alternate ID is that it has a greater chance of remaining meaningful if the other points of a zone are altered. Andi got the idea of using alternate IDs for certain purposes from unfoldingWord.

## Alignment

A *link* is a datum that has a set of source points and a set of target points, and stands for the concept that the target points are the translation of the source points. Equivalently, the link can be regarded as a relationship between some source positions and some target positions.

Two links are said to *conflict* if there is some source point that is in both links or there is some target point that is in both links. Otherwise the links are *nonconflicting* or *disjoint*.

A link type that must have exactly one source point and one target point is called a *monolink*. If the link is a more general type that can have more than one source point and more than one target point, then it is called a *multilink*.

A *zone* *alignment* is something that has a source zone, a target zone, and a disjoint set of links that all go between the source and target zones. A zone alignment that uses monolinks is called a *zone monoalignment*, and a more general zone alignment that uses multilinks is called a *zone multialignment.*

Clear3 defines a datatype (copied from Clear2) called a *legacy persistent alignment*. Its purpose is to be serialized as JSON in order to store alignments in files. This datatype expresses a collection of zone multialignments, where each of the zone pairs is also accompanied by descriptive information about each of its source points and target points.

When the output of auto alignment is converted to a legacy persistent alignment and serialized, the output file is conventionally called “alignment.json”. When a legacy persistent alignment is deserialized to obtain a former alignment that should influence a new auto alignment, the input file is conventionally named “oldAlignment.json”.

## Translation Model

A *translation model* means a mapping from a source lemma to a set of possible translations, where each possibility is a lower-cased target word with also a probability attached.

In Clear3 the step that trains an SMT model produces an estimated translation model. This result represents the guesses produced by the SMT training, based on the zone pairs that were provided as input.

Clear3 also provides for a so-called “manual translation model” as input, which is another translation model that takes priority over the estimated model during the auto alignment algorithm. The name “manual” is because in some workflows the manual model comes from manual alignment activities, but the source does not matter to Clear3. (The Clear2 legacy file formats for the estimated model and manual model differ; but Clear3 converts the information to a common form before use.)

## Estimated Alignment Model

The Clear3 SMT training step produces an *estimated alignment model*, which is a set of links, each accompanied by a probability. Unlike the zone alignments described in Section 3.4 above, the links in the estimated alignment model need not be disjoint, and represent different and perhaps conflicting possibilities found by the SMT training.

## The Syntax Tree

In Clear3 a *syntax tree* is a particular XML structure that represents the source words and their grammatical relationships. For example, here is a syntax tree for John 1:1:

Chart

Description automatically generated

In this diagram, the circles represent XML elements, and the lines show how one element is included inside of another element. (The numbers are to make discussion convenient, and do not appear in the XML.) The Greek-word labels, such as ἀρχῇ for element 8, represent that the Greek word occurs as the textual content of the XML node.

Elements at the fringe of the tree, such as 6, 8, 11, 14, and so on, are called *terminals*, and always represent source words. Other elements are called *non-terminals*.

The structure of the tree represents a grammatical analysis of the source text. The non-terminal elements have attributes that further express this analysis. The Clear3 auto alignment algorithm does not use the grammatical analysis, but only uses the structure of the tree. The terminal nodes have XML attributes that describe their associated source words. Clear3 uses some of the terminal-node attributes to discover information about the source words.

The order in which the source words occur along the fringe of the tree is called *tree order*, and might be different from the order of the source words in the source manuscript, which is called *manuscript order*. It is the manuscript order that determines the order of the source points within a source zone, but the source points also remember information about their tree order.

An element (such as 1, 4, or 7) with only one child is said to be *stacked* on top of its child, and a sequence of elements where each is stacked on the next (such as 9, 10, and 11) are said to form a *stack*. It turns out that the stacks, rather than the individual elements, are important when the auto-alignment algorithm traverses the syntax tree.

## Descriptive Collateral

The Clear3 algorithm requires some auxiliary descriptive information as input, including the following:

*Punctuation* is a set of strings that should be regarded as punctuation when they appear as the output of target text segmentation.

*Stop Words* are lemmas and lower-cased target texts that should never be aligned with anything.

Clear3 classifies both source words and target words as either *content words* (which are regarded as filled with meaning) and *function words* (which are regarded as part of the mechanics of the language). The function words are specified as input, and other words are all regarded as content words. Nouns, verbs, and adjectives are typically content words; prepositions and articles are typically function words.

## Strongs Codes

The syntax trees contain an *extended Strongs code* for each source word. In our current syntax trees, this code was assigned by Marumori, and is based on the well known Strongs code derived from Strong’s Concordance of the Bible. Often the extended code is the same as the ordinary Strongs code, but sometimes it discriminates among finer shades of meaning.

Clear3 has a usually empty *strongs database* input (conventionally from a file named “strongs.txt”) that maps a Strongs code to the target IDs that translate that Strongs code. Any such information conveys a great deal of knowledge about the target text, and has a large influence on the auto alignment.

## Link Checking Experience Database

Clear1 was used in combination with manual checking of alignments. The manual checking kept a database that counted when a translation was accepted (a *good link*) or rejected (a *bad link*). This data was then fed back to Clear1 to influence future auto alignments.

This feature was carried forward to Clear2 and Clear3. There are auxiliary inputs (conventionally from files named “goodLinks.txt” and “badLinks.txt”) which map from particular translations to counts of approval and rejection. Each particular translation is a pair consisting of a lemma and the lower-cased target text that should or should not translate the lemma. (A better naming might have been “good translations” and “bad translations.”)

In Clear3 the good links and bad links inputs have a limited influence on the auto alignment, as explained further below.

# Tree-Based Auto Alignment

The remainder of this document focuses on the tree-based auto alignment step. This step is the heart of Clear3 and a principal place where Andi invented new algorithms.

The overall context of this step is as follows:

Diagram

Description automatically generated

Tree-based auto alignment is driven by a sequence of *zone alignment problems*, which each consist of a portion of the source text (given by identifying the first and last verse) and a sequence of target words (each with its target ID) that are known to translate the source portion. The zone alignment problems were computed by the versification algorithm.

Tree-based auto alignment uses the syntax trees in an essential way, and produces a result that is a zone multialignment and is described further below.

Tree-based auto alignment is also conditioned by assumptions, which combine the results of previous computations with the specifications of certain parameters and the use of certain auxiliary inputs to Clear3. These assumptions are as follows:

* the estimated translation model and estimated alignment model, as computed by a previous step that trained a statistical-machine-translation model;
* the so-called “manual translation model,” which is an additional translation model supplied as input to Clear3 that takes priority over the estimated translation model;
* inputs to Clear3 that specify the punctuation, stop words, good links, bad links, source function words, and target function words, and having the meanings explained above;
* an old alignment input to Clear3, which is a zone multialignment that describes a former result that should influence the new alignment to be computed;
* a strongs database, with the meaning as explained above;
* Boolean flags that affect certain choices about content words and about use of the estimated alignment model within the algorithm, as explained further below;
* count thresholds for allowing the good links and bad links information to influence the algorithm, as explained further below; and
* a *maximum paths* parameter that is used to limit the number of alternatives to be considered, in order to guard against a combinatorial explosion of possibilities, as explained further below.

In these assumptions, each of the manual translation model, good links, bad links, old alignment, and strongs database may be empty.

The following diagram shows how the tree-based auto-alignment step is realized by assembling Clear3 building blocks:

Diagram

Description automatically generated

In this diagram:

* AlignZone, ConvertToZoneMultiAlignment, and GetLpaLine are the names of Clear3 client APIs.
* LpaLine is a Clear3 datatype for describing one zone multialignment, and occurs as a constituent of the LegacyPersistentAlignment datatype that was described above.
* The zone alignment problems are considered one at a time by AlignZone, which produces a zone monoalignment for each one.

Although we have been speaking of tree-based auto-alignment as something that Clear3 does, it would be more accurate to say that the algorithms we have been describing are one possible way of assembling the Clear3 building blocks. Our description has been of the principal workflow from Clear2, as realized by the regression tests currently in the Clear3 prototype, which assemble the Clear3 building blocks in certain ways.

# Aligning a Zone

The rest of this paper describes the algorithm inside of the AlignZone entry point.

Diagram

Description automatically generated

The steps in AlignZone are as follows:

Diagram

Description automatically generated

In the diagram above:

* The first step is to analyze the zone alignment problem.
  + Analysis identifies the source points (by consulting the syntax trees) and the target points.
  + This step also constructs a syntax tree for the zone, perhaps by combining several parts of the syntax trees and making new tree nodes as necessary.
* Once the source points are known, the algorithm identifies the possible candidates for each source point by consulting the assumptions. Each of these candidates is either a link between one source point and one target point, or else the certainty of linking a source point to nothing. When the analysis of source point candidates is completed, each source point has a list of candidates, each with a probability, and arranged in order of decreasing probability.
* Next, the algorithm identifies candidates for each node of the syntax tree for the zone, working from the terminals back toward the root. In this analysis, each candidate is one of the types described above for a source point, or else the union of sub-candidates. When this analysis is completed, each node of the syntax tree has a list of alternative candidates, each with its probability. The algorithm only keeps the best candidates for each node, and the alternatives for a given node all have the same probability.
* After the tree-based analysis is completed, the first candidate for the top node in the syntax tree for the zone is converted into a zone monoalignment.
* The final step examines the zone monoalignment to find source points that were not linked to anything, and attempts to link those points to target points. This step uses the assumptions and the syntax tree for the zone.
* The zone monoalignment produced by the final auxiliary alignment combines with the zone context to produce the overall result.

# Identifying Source Point Candidates

The following rules are applied in order to identify the candidate alignments for a source point.

1. If there is a link for the source word in the old alignment, then there is one candidate that links to the target word with certainty. (The process that compares the old alignment with the current zone uses the alternate IDs of the source and target words, to be more robust if the translated text has changed since the old alignment was performed.)
2. If the source word is not a content word, then there is one candidate that is the certainty of linking with no target word.
3. If the strongs database has target IDs for the strongs code of the source point (as obtained from the syntax tree), then
   1. there is a candidate for each target ID that occurs in the current zone, with a probability of 1,
   2. but if none of the target IDs occur in the current zone, then there is one candidate that is the certainty of linking with no target word.
4. If the source point is a stop word, then there is one candidate that is the certainty of linking with no target word.
5. If the manual translation model has any definitions for the lemma of the source point:
   1. There is a candidate for each target point in the current zone whose lower-cased text occurs as a possible translation for the lemma.
   2. The probability of a candidate is as given in the translation model, but is adjusted to be always at least 0.2.
   3. Only those candidates of maximal probability are kept.
   4. However, if no target points have lower-cased text that occurs as a possible translation for the lemma, then there is one candidate that is the certainty of linking with no target word.
6. If the estimated translation model has any definitions for the lemma of the source point:
   1. There is a candidate for each target point in the current zone whose lower-cased text occurs as a possible translation for the lemma, except that:
      1. If the combination of source lemma and lower-cased target text occurs in the bad links database with a count of at least the bad links threshold, then that target point is rejected.
      2. If the lower-cased target text is punctuation, then that target point is rejected.
      3. If the lower-cased target text is a stop word, then that target point is rejected.
   2. The probability of the candidate is computed from the probability given in the translation model as follows:
      1. If the flag to use the estimated alignment model is true:
         1. If the combination of lemma and lower-cased target text is mentioned in the estimated alignment model, then the probability is p1 + ((1.0 – p1) \* p2), where p1 is the probability from the estimated translation model and p2 is the probability from the estimated alignment model.
         2. Otherwise the probability is 0.6 times the score from the estimated translation model.
      2. Otherwise the probability is as in the estimated translation model.
   3. However, if no target points have lower-cased text that occurs as a possible translation for the lemma, then there is one candidate that is the certainty of linking the source point to nothing.
7. If none of the rules above apply, then there is one candidate that is the certainty of linking the target point to nothing.

The process that identifies candidates for source points also tries to cull the candidates. This culling algorithm examines those candidates that are less probable to find candidates that are competing with one another for the same target point, and tries to remove some of them.

# Identifying Tree-Node Candidates

Once the candidates for source points have been identified, then candidates are computed for each node of the syntax tree of the zone. This process starts at the terminal nodes of the syntax tree and works upward toward the root of the tree.

As an example, consider the following portion of the syntax tree that was given above for John 1:1.

A picture containing chart

Description automatically generated

The candidates for nodes 6, 8, 11, 17, and 34 are the candidates for their associated source points with surface text Ἐν, ἀρχῇ, ἦν, καὶ, and καὶ, respectively. The candidates for node 7 are the same as those for node 8. The candidates for node 5 are found by considering all possible unions of one candidate from node 6 and one candidate from node 7, using an algorithm that is discussed further below.

As with node 7, the candidates for nodes 9 and 10 are the same as for node 11. In fact, whenever there is a stack of tree nodes, such as 9, 10, and 11, every node in the stack has the same candidates. For this reason, the mapping from tree node to its candidates is actually a mapping from stacks of nodes to candidates.

Whenever a node has more than one child, such as nodes 2, 3, and 5 in the example above, the algorithm finds the candidates for the node by examining the candidates for its children, according to the following algorithm.

1. Compute all possible combinations that take one of the candidates from each child and forming the union of those sub-candidates. The *unadjusted probability* of such a union of sub-candidates is the product of the sub-candidate probabilities.
2. However, limit the exploration in such a way as to avoid considering more than approximately the number of possibilities given by the *maximum paths* parameter.
3. From these possibilities, eliminate those that conflict, in the sense of linking two different source points to the same target point.
4. However, always keep at least one candidate, even if it conflicts.
5. Compute an *adjusted probability* for each candidate, which depends on the relative arrangement of target points when lined up in the order of their corresponding source points. This computation is an important part of what Andi has invented based on his research, and a place where the local context is influencing the auto-alignment algorithm.
6. Sort the candidates in order of their adjusted probabilities, and then take the prefix of this sequence that has the same unadjusted probabilities.
7. This prefix is the desired set of candidates for the node. The unadjusted probability is the one that is used for the candidate as it travels up to the next level of the tree.

# Finishing the Auto-Alignment

Once the candidates for each node-stack in the tree have been computed, the zone alignment algorithm proceeds as follows to compute an initial zone monoalignment that is then refined into a final zone monoalignment:

1. Convert a most probable candidate for the top of the tree into a provisional alignment for the zone.
2. The links in the provisional alignment might not be disjoint, because the tree-traversal is forced to keep a conflicting candidate when it has no alternatives. So, apply an algorithm to resolve conflicts by removing some links, resulting in an initial zone monoalignment.
3. Attempt to improve the alignment by finding links for source points that are not linked in the provisional alignment. (Note that Andi no longer includes this step in his latest research.)
4. The improvement attempt might again lead to conflicts, so apply the algorithm for resolving conflicts again.
5. Apply an algorithm to interchange links so as to avoid crossings, where a *crossing* is what occurs when two links have their source points and target points in opposite orders, so that the links would cross each other in a picture.

The algorithm in Step 3 above considers the unlinked source points one at a time, and attempts to find a link for each one according to the following rules, applied in order:

1. If the source point is a stop word, the attempt fails.
2. If the Boolean flag for content words only is true and the source point is a function word, then the attempt fails.
3. If the Boolean flag that calls for using the estimated alignment model is true, and the source ID of the source point is mentioned in the estimated alignment model:
   1. Choose one of the estimated alignments that mentions the source ID as a proposal.
   2. If the proposed target point has already been linked, the attempt fails.
   3. If the source point is a stop word, then the attempt fails unless the proposal occurs in the good links database with a count at least equal to the good links threshold.
   4. If the proposal occurs in the bad links database with a count at least equal to the bad links threshold, then the attempt fails.
   5. If the target point is punctuation or a stop word, the attempt fails.
   6. Otherwise, add a link from the source point to the proposed target point.
4. Otherwise use the zone’s syntax tree together with the alignment so far to find a suitable unlinked target point that is “near” the source point:
   1. If the Boolean flag for content words only is true, then only content words are suitable.
   2. Target points that are punctuation are not suitable.
   3. If a suitable target point is found, add a link from the source point to the target point.
   4. Otherwise the attempt fails.

# Further Details

To understand the Clear3 tree-based auto-alignment algorithm at the next level of detail, please consult the source code of the Clear3 prototype. The code is heavily commented and organized in an attempt to promote understanding.

The following table suggests possible starting points in the code for more details about some of the topics discussed in this paper.

|  |  |
| --- | --- |
| **Topic** | **Possible Starting Points** |
| Concepts of Section 3 | DataModel\_Alignment.cs  IImportExportService |
| Assumptions | IAutoAlignAssumptions |
| LegacyPersistentAlignment | DataModel\_Persistence.cs  Persistence.cs |
| Identifying Source Point Candidates | TerminalCandidates.GetTerminalCandidateForWord |
| Culling Source Point Candidates | TerminalCandidates.GetTerminalCandidates |
| Syntax Tree Traversal to Find Candidates for Nodes | TreeBasedAlignment.AlignTree |
| Combining Child Candidates During Tree Traversal and Adjusted Probabilities | TreeBasedAlignment.ComputeTopCandidates |
| Attempting to Find Links for Unlinked Source Points | AuxiliaryAlignment.ImproveAlignment |
| Removing Conflicting Links from a Provisional Alignment | ZoneAlignment.ResolveConflicts |