A Graph-Based Approach to String Regeneration

Matic Horvat
Churchill College

A dissertation submitted to the University of Cambridge in partial fulfilment of the requirements for the degree of Master of Philosophy in Advanced Computer Science (Option B)

University of Cambridge
Computer Laboratory
William Gates Building
15 JJ Thomson Avenue
Cambridge CB3 0FD
UNITED KINGDOM

Email: mh693@cl.cam.ac.uk

June 14, 2013
Declaration

I Matic Horvat of Churchill College, being a candidate for the M.Phil in Advanced Computer Science, hereby declare that this report and the work described in it are my own work, unaided except as may be specified below, and that the report does not contain material that has already been used to any substantial extent for a comparable purpose.

Total word count: 14554

The word count excludes appendices which contain non-essential content.

Signed:

Date:

This dissertation is copyright ©2013 Matic Horvat.
All trademarks used in this dissertation are hereby acknowledged.
Abstract

The string regeneration problem is the problem of generating a fluent sentence from an unordered list of words. It can be considered as the most unconstrained and shallowest meaning representation provided to a Natural Language Generation system. The purpose of investigating and developing approaches to solving the string regeneration problem is grammaticality and fluency improvement of machine generated text. It is relevant to popular natural language applications such as Statistical Machine Translation and Summarisation. We investigated a graph-based approach to the string regeneration problem that finds a permutation of words with the highest probability. The evaluation of the approach on a number of datasets yielded promising results. We confirmed the results by conducting a manual evaluation study. We extended the basic graph-based approach to include phrases instead of just words. The extension requires additional work in order to fulfill its potential.
Acknowledgements

Foremost, I would like to express my sincere gratitude to my supervisor, Dr. Bill Byrne, for his guidance, support, and many fruitful discussions. I could not have hoped for a better supervisor.

I would like to sincerely thank Juan Pino of the Department of Engineering for his help with practical aspects of the dissertation. Your help and patience were invaluable.

I thank the University of Cambridge Department of Engineering for welcoming me as a visitor and generously offering their facilities.

My sincere thanks also go to Prof. Ann Copestake for her advice and guidance throughout my stay in Cambridge.

Many thanks to the Computer Laboratory course administrators Lise Gough and Katherine Cisek for their support throughout the course. Your positive attitude made everything seem easier than it was.

I thank all of the professors and lecturers of the Computer Laboratory and Department of Engineering that have taught me so many new things.

Finally, I would like to thank my friends in Cambridge and my family back home for their continuous support and happy moments. I couldn’t have done it without you.
Resources

I have based my work on the following resources:

1. Language models, corpora, and evaluation datasets were created and prepared by Juan Pino of Department of Engineering.

2. The Concorde TSP Solver was created by Applegate et al. (2006b).

3. The LKH heuristic TSP solver was created by Keld Helsgaun.

4. The language model server is a part of the SRILM toolkit created by the SRI Speech Technology and Research Laboratory.

5. The BLEU evaluation script was created by NIST.

6. Computing resources were provided by the University of Cambridge Department of Engineering.
# Contents

1 Introduction 1

2 Background 5
   2.1 Graph Theory Background 5
   2.2 The Travelling Salesman Problem 8
   2.3 Variations of the Traveling Salesman Problem 10
   2.4 BLEU Evaluation Metric 12

3 Graph-Based Approach to String Regeneration 17
   3.1 N-gram Language Models 18
   3.2 The Naive Approach 20
   3.3 Initial Work on Constraint Satisfaction 21
   3.4 Bigram Graph-Based Approach 24
   3.5 Higher N-gram Order Graph-Based Approach 27

4 Implementation 33
   4.1 Overview 33
   4.2 Language Models 36
   4.3 TSP Solvers 37

5 Evaluation 41
   5.1 Evaluation metric 41
   5.2 Datasets and Preprocessing 42
   5.3 Automatic Evaluation 45
   5.4 Manual Evaluation 48

6 Graph-Based Approach with Phrase Constraints 53
   6.1 Graph-Based Approach with Phrase Constraints 54
   6.2 Phrase Selection 57
   6.3 Evaluation 59
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>Related Work</td>
<td>63</td>
</tr>
<tr>
<td>8</td>
<td>Conclusions and Future Work</td>
<td>65</td>
</tr>
<tr>
<td></td>
<td>8.1 Summary</td>
<td>65</td>
</tr>
<tr>
<td></td>
<td>8.2 Future Work</td>
<td>66</td>
</tr>
<tr>
<td>A</td>
<td>Graph Transformations</td>
<td>75</td>
</tr>
<tr>
<td></td>
<td>A.1 Asymmetric to Symmetric TSP Graph Transformation</td>
<td>75</td>
</tr>
<tr>
<td></td>
<td>A.2 Generalized ATSP to ATSP Graph Transformation</td>
<td>78</td>
</tr>
<tr>
<td>B</td>
<td>TSP Solvers</td>
<td>83</td>
</tr>
<tr>
<td></td>
<td>B.1 Concorde TSP Solver</td>
<td>83</td>
</tr>
<tr>
<td></td>
<td>B.2 LKH</td>
<td>84</td>
</tr>
</tbody>
</table>
## List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>An example of an undirected graph with a simple path</td>
<td>6</td>
</tr>
<tr>
<td>2.2</td>
<td>An example of a directed graph with a simple cycle</td>
<td>6</td>
</tr>
<tr>
<td>2.3</td>
<td>An example of a tour of a graph</td>
<td>7</td>
</tr>
<tr>
<td>2.4</td>
<td>An example of the shortest tour of a graph</td>
<td>7</td>
</tr>
<tr>
<td>2.5</td>
<td>The increase of the largest optimally solved TSP instance over the years</td>
<td>9</td>
</tr>
<tr>
<td>2.6</td>
<td>An example GTSP instance</td>
<td>11</td>
</tr>
<tr>
<td>3.1</td>
<td>An example of the naive approach to finding the best permutation</td>
<td>22</td>
</tr>
<tr>
<td>3.2</td>
<td>A general graph modelling the bigram approach for a 3 word sentence</td>
<td>25</td>
</tr>
<tr>
<td>3.3</td>
<td>A graph modelling the bigram approach for a specific 3 word sentence</td>
<td>26</td>
</tr>
<tr>
<td>3.4</td>
<td>A general graph modelling the trigram approach for a 3 word sentence</td>
<td>31</td>
</tr>
<tr>
<td>3.5</td>
<td>A graph modelling the trigram approach for a specific 3 word sentence</td>
<td>32</td>
</tr>
<tr>
<td>5.1</td>
<td>The distribution of sentence length of the three datasets</td>
<td>44</td>
</tr>
<tr>
<td>5.2</td>
<td>The distribution of chopped sentence length of the three datasets</td>
<td>44</td>
</tr>
<tr>
<td>5.3</td>
<td>Performance of 3 versions of the approach at increasing sentence length</td>
<td>47</td>
</tr>
<tr>
<td>5.4</td>
<td>Output examples of 3 versions of the graph-based approach</td>
<td>49</td>
</tr>
<tr>
<td>6.1</td>
<td>An example graph modelling the bigram approach with phrase constraints</td>
<td>55</td>
</tr>
<tr>
<td>6.2</td>
<td>An example graph modelling the trigram approach with phrase constraints</td>
<td>56</td>
</tr>
<tr>
<td>6.3</td>
<td>A continued example of the trigram approach with phrase constraints</td>
<td>56</td>
</tr>
<tr>
<td>6.4</td>
<td>Example output of the approach with phrase constraints</td>
<td>62</td>
</tr>
</tbody>
</table>
List of Tables

2.1 BLEU score computation for two example translations. . . . . 13
3.1 Illustration of problem size growth for $O(n^2 2^n)$ and $O(n \cdot n!)$ . 27
4.1 The growth of the vertex set size for increasing length of sen-
tences . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 35
5.1 Evaluation results of 4 versions of the graph-based approach . 47
5.2 Manual evaluation results . . . . . . . . . . . . . . . . . . . . 51
5.3 Raw numbers of the manual evaluation . . . . . . . . . . . . . 51
6.1 Evaluation results of the approach with phrase constraints . . 60
6.2 Phrase selection statistics . . . . . . . . . . . . . . . . . . . . 60
Chapter 1

Introduction

The string regeneration problem can be stated as: given a set of unordered words taken from a fluent grammatical sentence, recover the original sentence. As it is often difficult to recover the exact original sentence based solely on a set of words, the problem is relaxed to generating a fluent version of the original sentence (Zhang and Clark, 2011).

The string regeneration problem can generally be considered a difficult problem even for humans. Consider the following set of words, \{ Iraq, list, in, a, third, joins, the, of, Bush’s, of, ., critics, policy, senator, republican \}, and try to recover the original sentence or at least a fluent grammatical sentence. The original sentence was: a third republican senator joins the list of critics of Bush’s policy in Iraq.

The string regeneration problem is most often investigated in the field of Natural Language Generation. The basic task of all Natural Language Generation approaches is to take some sort of meaning representation as input and generate human-readable output. The approaches differ on how much information is required from the meaning representation. The representation can range in depth from semantically annotated dependency graphs to shallow syntactic dependency trees. Taking an unordered set of words, which is the string regeneration problem we investigate in this thesis, can be con-
considered as the most unconstrained and shallowest meaning representation provided to a natural language generation system.

The purpose of investigating and developing approaches to solving the string regeneration problem is grammaticality and fluency improvement of machine generated text. Knight (2007) characterised the output of Statistical Machine Translation systems as ‘gobbledygook’. That is, the output of SMT systems often lacks grammaticality and fluency. Other systems that generate text, such as abstract-like text Summarisation systems, Question Answering systems, and Dialogue Systems, suffer from the same problem (Soricut and Marcu, 2005). The string regeneration problem is therefore often used as an application-independent method of evaluating approaches for improving grammaticality and fluency of the systems generating text.

In the thesis we investigated an approach to the string regeneration problem that works under the assumption that the best permutation of the input words is the one with the highest probability under a language model. We explored different approaches to finding the highest probability permutation and decided on a graph-based approach. The graph-based approach models the problem as a set of vertices containing words and a set of edges between the vertices, whose cost equals language model probabilities. Finding the permutation with the highest probability in the graph formulation is equal to finding the shortest tour in the graph or, equally, solving the Travelling Salesman Problem. Despite the TSP being an NP-hard problem, state-of-the-art approaches exist to solving large problem instances. This enabled us to find optimal permutations of words under a given language model and consequently solve the string regeneration problem. Additionally, we extended the graph-based approach to enable the use of phrases as input instead of just words.

The remainder of the thesis is structured as follows. In Chapter 2 we describe the concepts from various fields that are important for understanding the graph-based approach and its implementation. In Chapter 3 we define the graph-based approach and describe it in detail. In Chapter 4 we discuss the practical issues with the implementation of the approach. In Chapter 5 we
describe the setup and report the results of the experimental evaluation of the graph-based approach. In Chapter 6 we extend the graph-based approach to include phrases instead of just words. In Chapter 7 we summarize the related work on the string regeneration problem. Finally, in Chapter 8 we summarize our work and discuss the possible future work.
Chapter 2

Background

In the following sections we describe the concepts from various fields that are important for understanding the graph-based approach to the string regeneration problem. In Section 2.1 we introduce some of the basic concepts from Graph Theory that are necessary for defining and understanding the graph-based approach in later chapters. In Sections 2.2 and 2.3 we describe the Travelling Salesman Problem and its variations, which are vital to our approach to the string regeneration problem. Finally, in Section 2.4 we describe BLEU, an automatic machine translation metric that was used to evaluate the output quality of our approach.

2.1 Graph Theory Background

The following description of basic graph theory concepts draws from Bollobás (1998). A graph is a structure consisting of vertices and edges. It is defined as $G = (V, E)$, where $V$ is the set of vertices and $E$ is the set of edges. In an undirected graph the edges connect pairs of vertices with no orientation. On the other hand, in a directed graph an edge is directed from an origin vertex to a destination vertex. Examples of undirected and directed graphs are shown in Figures 2.1 and 2.2.
Figure 2.1: An example undirected graph. A simple path starting in vertex A and ending in D is shown in bold.

Figure 2.2: An example directed graph. A simple directed cycle including vertices B, C, and D is shown in bold.

The edges in a graph can have weights associated with them. The edge weight often represents the cost that is incurred by moving from one vertex to the other using that edge. With weighted graphs we can model a variety of problems, for example, computer communication networks. In a communication network graph, vertices represent network devices and edges represent connections between them, while the edge weights represent the connection delay. Standard graph theory algorithms can then be used to solve problems such as routing by finding the shortest path between two vertices.

A path in a graph is a sequence of edges and vertices beginning in a start vertex and ending in an end vertex. A simple path is a path with no repeated vertices, that is, no vertex contained in the path is visited twice. An example simple path is shown in Figure 2.1 in bold. A cycle is a path with identical start and end vertex. Similar to a simple path, a simple cycle contains no repeated vertices with the exception of repetition of the start and end vertex. An example simple cycle is shown in Figure 2.2. A $k$-path is a path consisting of $k$ edges. Similarly, a $k$-cycle is a cycle consisting of $k$ edges. The path in Figure 2.1 is a 2-path and the cycle in Figure 2.2 is a 3-cycle.

A (vertex) tour or a Hamiltonian cycle is a simple cycle that includes every
vertex in a graph. Not every graph contains a Hamiltonian cycle. Determining if such cycles exist in a graph is an NP-complete problem referred to as Hamiltonian cycle problem. The example graphs shown in Figures 2.1 and 2.2 do not contain any tours. However, if we add a single edge between vertices A and E to the example in Figure 2.1, the graph contains a single Hamiltonian cycle. The cycle is shown in Figure 2.3. We will refer to a Hamiltonian cycle simply as a tour throughout the thesis.

A $k$-partite graph is a graph that can be decomposed into $k$ mutually disjoint sets of vertices, such that no edge connects two vertices in the same set. A graph is complete if there exists an edge between any pair of vertices in the graph. A directed graph is complete if there exists a pair of edges with different orientation between any pair of vertices in the graph. A $k$-partite graph is complete if there exists an edge between any pair of vertices belonging to two different mutually disjoint sets of vertices in the graph.
2.2 The Travelling Salesman Problem

The Travelling Salesman Problem (TSP) is a problem stated as: given a list of cities and distances between them, find the shortest route visiting all cities exactly once and ending in the original city. It is an NP-hard problem and often used as an example of such problems. TSP is one of the most intensely studied problems in computational mathematics and was applied to problems in numerous areas, including logistics, genetics, telecommunications, and neuroscience (Applegate et al., 2006a). Due to its prominence it is often used as a benchmark for new optimization algorithms.

The Travelling Salesman Problem is most often modelled as an undirected weighted graph $G = (V,E)$. Each vertex is a single city and the weight of an edge between two vertices is the distance between the two cities. Finding the shortest route visiting all cities exactly once and ending in the original city therefore equals finding a tour of the graph, such that the sum of weights of visited edges is minimal. An example TSP and its solution are shown in Figure 2.4.

As the Travelling Salesman Problem is an NP-hard problem, no known polynomial time algorithm exists for solving general problem instances. A naive approach checking all permutations of $n$ vertices has the time complexity of $O(n!)$. In 1962 Held and Karp developed a dynamic programming algorithm for solving TSP with time complexity of $O(n^2 2^n)$. This presented a marked improvement on the naive approach for large $n$. To this day the Held and Karp algorithm remains the algorithm with the best-known running time guarantee for solving general instances of the TSP (Applegate et al., 2006a).

However, the poor time complexity guarantee did not stop researchers from developing better approaches to solving TSP that do not have such guarantees. As the new algorithms could not be compared using time complexity guarantees, this led to an alternative method of measuring progress using the size of the largest solved TSP instance. Algorithms are often evaluated on problem instances in TSPLIB, a TSP test suite introduced by Reinelt (1991).
One of the earliest successful approaches to solving the TSP was the cutting plane method using linear programming developed by Dantzig, Fulkerson, and Johnson (1954). It solved a problem instance with 49 cities, the largest at the time. In the following years approaches using branch-and-bound method gained popularity but the largest solved TSP instance remained unchanged (Applegate et al., 2006a). In 1970 Held and Karp developed a method for solving TSP instances using linear programming relaxation and increased the number of cities of the largest solved instance to 64. In 1970s and 1980s Martin Grötschel, Manfred Padberg, and Giovanni Rinaldi continued to increase the largest solved TSP instance, solving a 2,392 city problem in 1987 (Applegate et al., 2006a). The size of the largest solved instance increased rapidly through 1990s and 2000s due to improved approaches and advanced hardware platforms. The largest TSP instance of the TSPLIB containing 85,900 cities was optimally solved in 2006 (Applegate et al., 2009) using the Concorde TSP Solver. The computation took over 136 CPU years. The increase in the size of the largest optimally solved TSP instance over the years on logarithmic scale (image taken from Applegate et al. (2006a, p53)). The datapoints after the solution of a 2,392 city instance by Padberg and Rinaldi in 1987 (PR) were all solved using the Concorde TSP Solver and are marked with the names of the specific TSPLIB problem instances.
TSP instance over the years is shown in Figure 2.5.

Finding an optimal tour in a large problem instance often requires long computation time. This is unacceptable for many practical applications. Many heuristic approaches that attempt to find a good solution quickly have instead been developed and applied to the TSP. These include both approaches developed specifically for solving the TSP and general heuristic algorithms, often referred to as metaheuristics. These include local-search algorithms (for example guided local search by Voudouris and Tsang (1999)), simulated annealing (Kirkpatrick et al., 1983), ant colonies (Dorigo and Gambardella, 1997), and genetic algorithms (a review of representations and operators appears in Larrañaga et al. (1999)). One of the most successful heuristic approaches to solving TSP to date remains the algorithm by Lin and Kernighan (1973) and its improvements.

2.3 Variations of the Traveling Salesman Problem

The Asymmetric Travelling Salesman Problem (ATSP) differs from (Symmetric) TSP in that the distance between two cities is not necessarily the same in both directions. This extension enables representation of a wider set of problems, for example including one-way streets in finding the shortest tour of $n$ cities. The ATSP is modelled as a directed weighted graph, such that each vertex represents a single city and the weight of a directed edge between two vertices equals the distance between the cities, travelling from origin to destination city.

The ATSP is not as widely studied as TSP and fewer approaches exist to solving it. A good overview of heuristic approaches to solving ATSP is provided by Johnson et al. (2002), including experimental evaluation and comparison of the approaches.

Alternatively, transformations from ATSP to TSP exist that at most double
the number of vertices. Although the transformations increase the problem instance significantly, they enable the use of state-of-the-art algorithms for solving the original asymmetric problem.

The Generalized Travelling Salesman Problem (GTSP) is a generalization of the TSP in which \( n \) cities are grouped into \( s \) districts with no overlap, each district containing \( m \) cities \( (n = s \cdot m) \). The GTSP task is to find the shortest tour of length \( s \) visiting exactly one city in each district. The GTSP is modelled as an \( s \)-partite directed weighted graph, such that each partition represents a single district and contains \( m \) vertices representing cities. Solving the GTSP therefore equals finding the shortest \( s \)-path in the graph. The TSP is a special case of the GTSP in which each district contains a single city and therefore \( s = n \). An example GTSP is shown in Figure 2.6.

Figure 2.6: An example GTSP instance with \( s = m = 3 \) and \( n = s \cdot m = 9 \). A possible tour visiting one vertex in each district is shown in bold. As exactly one city is visited in each district we do not model connections between cities in the same district.

GTSP has many applications, including routing, scheduling, and location-routing (Renaud and Boctor, 1998). Although specialized approaches exist for solving the GTSP (for example, Fischetti et al. (1997), Renaud and Boc-
tor (1998), and Snyder and Daskin (2006)), the GTSP is often instead transformed into TSP and solved using state-of-the-art algorithms for solving the original problem.

The Generalized Asymmetric Travelling Salesman Problem (GATSP) is the generalization of the ATSP. It is similar to GTSP, but with directed edges between cities instead of undirected edges. Like ATSP, GATSP is modelled as a directed weighted graph.

2.4 BLEU Evaluation Metric

BLEU is an automatic machine translation evaluation metric introduced by Papineni et al. (2002). The BLEU evaluation metric can be used as an inexpensive and fast method of measuring incremental progress of statistical machine translation systems. It is currently the most popular evaluation metric in the SMT community. One of the main reasons for its popularity is its high correlation with human judgement of translation quality.

Despite its popularity, it is widely acknowledged that it is a flawed metric. It has been shown by Callison-Burch et al. (2006) that it sometimes does not correlate well with human judgement of translation quality. They concluded that BLEU should not be used for comparisons of systems with radically different approaches to machine translation. Additionally, a BLEU score achieved by a translation system is meaningless by itself as it depends on the language pair, translation domain, number of reference translations, tokenization scheme, and other factors (Koehn, 2010). This makes direct comparison of different systems difficult. This weakness is alleviated by frequent evaluation competitions.

In order to ensure that changes in BLEU scores reflect changes in actual translation quality it is necessary to confirm them through manual evaluation.

---

1This section has previously been submitted with minor changes as part of the L102 Statistical Machine Translation module course work.
Additionally, when reporting evaluation results, example translations should be shown alongside the BLEU scores.

In the remaining part of this section we describe how the BLEU score is computed, using a machine translation example to guide our explanation.

BLEU measures translation closeness to a reference translation using N-gram precision. Consider the following example:

Reference: The South African company constructed a new children’s hospital

Translation A: South Africa company built new children’s hospital

Translation B: The South African company built a new hospital for children

The N-gram matches between translations and the reference are underlined. N-gram precision of a translation is computed by dividing the number of N-gram matches by the total number of N-grams in the translation. For example, 1-gram precision of translation A is 5/7. N-gram precisions of both translations up to 4-grams are shown in Table 2.1.

<table>
<thead>
<tr>
<th>Metric</th>
<th>Translation</th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-gram precision</td>
<td>5/7</td>
<td>7/10</td>
<td></td>
</tr>
<tr>
<td>2-gram precision</td>
<td>2/6</td>
<td>4/9</td>
<td></td>
</tr>
<tr>
<td>3-gram precision</td>
<td>1/5</td>
<td>2/8</td>
<td></td>
</tr>
<tr>
<td>4-gram precision</td>
<td>0/4</td>
<td>1/7</td>
<td></td>
</tr>
<tr>
<td>Brevity Penalty</td>
<td>0.75</td>
<td>1.00</td>
<td></td>
</tr>
<tr>
<td>BLEU</td>
<td>54.0</td>
<td>61.4</td>
<td></td>
</tr>
</tbody>
</table>

Table 2.1: BLEU score computation for the two example translations A and B, including the N-gram precision, brevity penalty, and the final BLEU score.

The number of N-gram matches is clipped to the maximum number of occurrences of the N-gram in the reference. For example, if translation C was The The The The, its 1-gram precision would be 1/4, because The only occurs once in the reference. If clipping was not performed, its precision would be
4/4, which is clearly undesirable.

The BLEU metric is computed using the following formula:

\[\text{BLEU} = \text{BP} \cdot \exp \left( \frac{1}{N} \sum_{n=1}^{N} \log p_n \right)\]

\(p_n\) is the clipped N-gram precision, while \(N\) is the highest order of N-grams used in the computation of BLEU score. It is often set to 4. \(\frac{1}{N} \sum_{n=1}^{N} \log p_n\) is therefore a geometric mean of \(N\) N-gram precisions.

BP refers to the brevity penalty. As the name implies, it penalizes short translations. Short translations receive high N-gram precision scores as it is easier to achieve high precision for a small number of N-grams compared to a large number of N-grams. The brevity penalty is computed using the following formula:

\[
\text{BP} = \begin{cases} 
1 & \text{if } \text{len(trans)} > \text{len(ref)} \\
 e^{(1-\text{len(ref)}/\text{len(trans)})} & \text{otherwise}
\end{cases}
\]

In our example, the reference is 9 words long, while translations A and B are 7 and 10 words long respectively. Their brevity penalties therefore equal 0.75 and 1.

BLEU score is often computed over the entire test set. This is accomplished by counting N-gram matches over all sentences in the test set. The counts are divided by the total number of N-grams in the test set to obtain the test set N-gram precision.

Finally, the BLEU metric can be modified to use multiple reference translations for each sentence. With multiple references an N-gram match occurs if the N-gram occurred in any of the references. The number of N-gram matches is clipped to the maximum number of occurrences of the N-gram in the reference with the most occurrences of the N-gram. The reference length, which is used in computation of the brevity penalty, is the length of
the closest reference in terms of length. If two references are equally close to
the translation in terms of length, the shorter one is taken as the reference
length. For example, given a translation length of 7 and reference lengths 5,
6, 8, and 12, the reference length is 6.

The final BLEU score for translations A and B evaluated against a single
reference with $N = 4$ is 54.0 and 61.4 respectively.
Chapter 3

Graph-Based Approach to String Regeneration

The string regeneration problem can be stated as: given a set of words, recover the original sentence. As it is often difficult to recover the exact original sentence based solely on a set of words, the problem is relaxed to generating a fluent version of the original sentence.

The basic idea we investigated in this thesis is using an N-gram language model to compute the probabilities of sentence permutations and pick the sentence with the highest probability as our solution.

In the following section we describe how an N-gram language model is used to compute the probability of a sentence permutation. In Section 3.2 we describe a naive approach to finding the sentence permutation with the highest probability by enumerating all sentence permutations and computing their probabilities. In Section 3.3 we describe our initial work on an approach to finding the sentence permutation with the highest probability using constraint satisfaction. In Section 3.4 we formulate our graph-based approach to finding the sentence permutation with the highest probability using a bi-gram language model. In Section 3.5 we extend the graph-based approach to higher order N-gram language models.
3.1 N-gram Language Models

The following discussion of N-gram language models is based on Jurafsky and Martin (2009, Chap. 4). N-gram language models are pervasive in Natural Language Processing and Statistical Machine Translation applications. Intuitively, N-gram language models attempt to capture the fact that some words are more likely to follow each other than others. For example, given a word sequence *what are you up*, it is more likely that the next word is *to* than *with*. This can be expressed as a conditional probability of a word given the context, \( P(to \mid \text{what are you up}) > P(with \mid \text{what are you up}) \). These conditional probabilities are provided by the N-gram language model. They are estimated by frequency counts on large text corpora. The probabilities are usually adjusted by procedures such as smoothing, interpolation, and backoff to account for data sparsity.

We often need to estimate the probability of a sequence of words, for example a probability of a sentence. Computing the probability of a sequence of words is indeed a vital part of our approach to solving the string regeneration problem. Using the chain rule, the probability of a sequence of \( n \) words \( w_1^n = w_1 \ldots w_n \) can be decomposed as:

\[
P(w_1^n) = P(w_1) \cdot P(w_2 \mid w_1) \cdot P(w_3 \mid w_2) \ldots P(w_n \mid w_1^{n-1})
\]

\[
= \prod_{k=1}^{n} P(w_k \mid w_1^{k-1}) \tag{3.1}
\]

This formulation requires the computation of conditional probability of a word given all previous words in the sequence. The final conditional probability is \( P(w_n \mid w_1^{n-1}) \). As the context \( w_1^{n-1} \) can be very long, it is unlikely that the language model probability estimates for \( P(w_n \mid w_1^{n-1}) \) are reliable. For a long sentence, it is unlikely that the particular sequence of words has occurred in the corpus at all. The probability \( P(w_n \mid w_1^{n-1}) \) is instead approximated with a shorter context of length \( N - 1 \), \( P(w_n \mid w_1^{n-1}) \approx P(w_n \mid w_{n-N+1}^{n-1}) \).
A bigram language model \((N = 2)\) computes the conditional probability of a word using only the preceding word, \(P(w_i|w_{i-N+1}^{i-1}) = P(w_i|w_{i-1})\). Referring to the initial example, instead of using the entire context, \(what\ are\ you\ up\), a bigram language model uses only the last word in the context, \(up\). The conditional probability of the next word being \(to\) is then computed as \(P(to | up)\). This formulation can be easily extended to higher order N-gram language models. Typical values of \(N\) are between 2 and 5.

Due to practical reasons (the danger of underflow when multiplying small probabilities) log probabilities are used instead of actual probabilities. The log probability of a sequence of \(n\) words is then computed as:

\[
\log P(w_1^n) = \sum_{k=1}^{n} \log P(w_k|w_{k-N+1}^{k-1})
\]  

(3.2)

using an N-gram language model. Note that the logarithm of a probability is a negative number.

Consider a short example sentence \(seize\ the\ day\). The bigram log probabilities are \(\log P(the \mid seize) = -0.52\) and \(\log P(day \mid the) = -2.82\), and the backed-off unigram probability of the first word in the sequence is \(\log P(seize) = -4.71\). The log probability of the sentence is then computed as:

\[
\log P(seize\ the\ day) = \sum_{k=1}^{3} \log P(w_k|w_{k-1})
\]
\[
= \log P(seize)
\]
\[
+ \log P(the \mid seize)
\]
\[
+ \log P(day \mid the)
\]
\[
= -8.05
\]  

(3.3)
3.2 The Naive Approach

Our approach to solving the string regeneration problem uses an N-gram language model to compute the probabilities of sentence permutations and pick the sentence with the highest probability under a given language model as our solution. The naive approach to finding the sentence permutation with the highest probability is to enumerate all permutations, compute their probabilities using Equation 3.2, and choose the permutation with the highest probability as the solution.

The example shown in Figure 3.1 illustrates the naive approach using the sentence \textit{seize the day} and a bigram language model to compute the conditional log probabilities. The permutation with the highest probability under the bigram language model is the original sentence \textit{seize the day} with log probability of $-8.05$.

The time complexity of the naive approach is $O(n \cdot n!)$ as we are enumerating all permutations of the $n$ word sentence and multiplying $n$ conditional probabilities for each permutation. This means that the naive approach is not viable for sentences of even moderate length. For example, there are 3,628,800 permutations of a 10 word sentence and the number grows to 355,687,428,096,000 for a 17 word sentence.

In the discussion and the examples we have considered so far we used only the content words for constructing the permutations of sentences. However, in many NLP and SMT applications two additional symbols marking the start and end of sentence are inserted before and after the sentence. They are most often denoted as $<$\texttt{s}$>$ and $<$\texttt{/s}$>$. Their function is to mark explicitly that a word is the first word in the sentence and that a word is the last word in the sentence. Language models include conditional probabilities that model these two symbols. The actual sentence log probability of the highest scoring
example demonstrated above is computed as:

$$\log P(<s\> \text{seize the day } </s>) = \log P(\text{seize } | <s>)$$

$$+ \log P(\text{the } | \text{seize})$$

$$+ \log P(\text{day } | \text{the})$$

$$+ \log P(</s> | \text{day})$$ (3.4)

3.3 Initial Work on Constraint Satisfaction

We initially developed an approach to solving the string regeneration problem using constraints. The original idea is based on the ideas described by Rush and Collins (2012). They give an overview of the possible uses of Lagrangian relaxation for inference in many Natural Language Processing applications, where a problem can be stated as a set of linear constraints and an objective function.

The string regeneration problem solved by finding the highest probability permutation of words can be described in the following manner. Given a set of \( n \) words \( S, |S| = n \), we would like to find the best permutation \( \rho \) of the words from the set \( S \) under some N-gram language model. For a bigram model, this is written as:

$$\arg \max_{\rho} \prod_{i=1}^{n} P(w_{\rho_i} | w_{\rho_{i-1}})$$ (3.5)

An equivalent statement is to find the best set of histories \( h_i, i = 1, \ldots, n \) so that \( \prod_{i=1}^{n} P(w_i | h_i) \) is maximized subject to the constraints that \( \sum_{i=1}^{m} 1\{h_i[m] = w_j\} = 1 \) for \( j = 1, \ldots, n \) and \( m = 1, \ldots, N - 1 \).

The constraints ensure that each word \( w_j \) appears exactly once at position \( m \) in any history. As proposed by Rush and Collins (2012), we can transform the objective function by relaxing the constraints and instead introducing them.
Unigram backed-off log probabilities:

\[
\begin{align*}
\log P(\text{day}) &= -3.77 \\
\log P(\text{seize}) &= -4.71 \\
\log P(\text{the}) &= -2.76
\end{align*}
\]

Bigram conditional log probabilities:

\[
\begin{align*}
\log P(\text{seize} \mid \text{day}) &= -5.30 \\
\log P(\text{the} \mid \text{day}) &= -2.33 \\
\log P(\text{day} \mid \text{seize}) &= -4.82 \\
\log P(\text{the} \mid \text{seize}) &= -0.52 \\
\log P(\text{day} \mid \text{the}) &= -2.82 \\
\log P(\text{seize} \mid \text{the}) &= -7.01
\end{align*}
\]

Log probabilities of $3! = 6$ permutations of the three word sentence \textit{seize the day}:

\[
\begin{align*}
\log P(\text{day seize the}) &= \log P(\text{day}) + \log P(\text{seize} \mid \text{day}) + \log P(\text{the} \mid \text{seize}) = -9.59 \\
\log P(\text{day the seize}) &= \log P(\text{day}) + \log P(\text{the} \mid \text{day}) + \log P(\text{seize} \mid \text{the}) = -13.11 \\
\log P(\text{seize day the}) &= \log P(\text{seize}) + \log P(\text{day} \mid \text{seize}) + \log P(\text{the} \mid \text{day}) = -12.35 \\
\log P(\text{seize the day}) &= \log P(\text{seize}) + \log P(\text{the} \mid \text{seize}) + \log P(\text{day} \mid \text{the}) = -8.05 \\
\log P(\text{the day seize}) &= \log P(\text{the}) + \log P(\text{day} \mid \text{the}) + \log P(\text{seize} \mid \text{day}) = -10.88 \\
\log P(\text{the seize day}) &= \log P(\text{the}) + \log P(\text{seize} \mid \text{the}) + \log P(\text{day} \mid \text{seize}) = -14.59
\end{align*}
\]

Figure 3.1: An example showing the naive approach of finding the permutation of the sentence \textit{seize the day} with the highest probability using a bigram language model. The sentence permutation with the highest log probability under the given bigram language model is the original sentence \textit{seize the day} (log probability of $-8.05$).
into the objective function with Lagrangian multipliers. A common approach to solving the transformed problem is to use a subgradient algorithm to minimize the new objective function. In each iteration the Lagrangian multipliers are updated to reflect whether their respective constraints are satisfied in the current best solution. The multipliers therefore affect the score of the objective function in such a way to force the constraints to be satisfied in order to achieve the objective function minimum.

However, the simple form of the constraints as given above do not ensure that the solution is a valid permutation of words. Consider the following example. Given a set of words $S = \{ \text{This}, \text{is}, \text{a}, \text{company} \}$, and a bigram language model, $N = 2$, the following solution is valid under the constraints defined above:

$$w_1 = \text{This}, \ h_1 = \text{is}$$

$$w_2 = \text{is}, \ h_2 = \text{This}$$

$$w_3 = \text{company}, \ h_3 = \text{a}$$

$$w_4 = \text{a}, \ h_4 = \text{company}$$

Each word from the set $S$ occurs exactly once in the history of another word. However, the permutation is clearly not valid, as the first two and last two words form a cycle and are not joined together with histories to form a permutation. To prevent such solutions from occurring, additional constraints need to be introduced. We have explored various approaches to formulate these constraints. However, we did not find an elegant solution to this problem as all of the approaches required introducing a large number of additional constraints that increased rapidly with the length of the sentence. Because of this, we decided to abandon the constraint satisfaction approach to the string regeneration problem in favor of the more intuitive graph-based approach presented in the following sections.
3.4 Bigram Graph-Based Approach

In this section we define the graph-based approach to finding the sentence permutation with the highest probability and consequently our solution to the string regeneration problem. For a bigram language model, $N = 2$, and a set of words $S$, we define a directed weighted graph $G = (V, E)$, such that given the set of symbols $X$, $X = S \cup \{<s>, </s>\}$, the set of vertices is defined as $V = \{w_i|w_i \in X\}$. Therefore, each symbol in $X$ is represented by a single vertex. Let the set of edges $E$ be a set of ordered pairs of vertices $(w_i, w_j)$, such that $E = \{(w_i, w_j)|w_i, w_j \in V\}$. The edge cost is then defined as:

$$
c_{ij} = \begin{cases} 
0 & \text{if } w_i = </s> \text{ and } w_j = <s>, \\
-\log P(w_j|w_i) & \text{if } w_i \neq w_j, \\
\infty & \text{otherwise.}
\end{cases}
$$

The conditional log probabilities of the form $\log P(w_j|w_i)$ are computed by a bigram language model. Consequently, finding the sentence permutation with the highest probability under the bigram language model equals finding the shortest tour in graph $G$ or equally, solving the Asymmetric Travelling Salesman Problem modelled by graph $G$. A general example graph for a sentence of length 3 is shown in Figure 3.2.

The individual cases of the edge cost function presented in Equation 3.6 ensure that the solution tour is a valid sentence permutation. The negation of log probabilities transforms the problem of finding the longest tour in graph $G$ to the common problem of finding the shortest tour.

Figure 3.3 shows an example graph for the sentence seize the day using a bigram language model. The shortest tour is shown in bold and represents the word sequence $<s> \text{seize the day </s>}. The shortest tour equals the sentence permutation with the highest probability under the bigram language model. The log probability of the sequence is $-10.98$. Note that this log probability differs from the log probability computed for the same sentence permutation.
Figure 3.2: A graph modelling the string regeneration problem for a general three word sentence. The edge cost equals the negated bigram conditional log probability of the destination vertex given origin vertex. Only edges with non-infinite edge cost are shown in the graph. Finding the shortest tour in the graph, or equally solving the ATSP modelled by the graph, equals finding the sentence permutation with the highest probability.
Figure 3.3: A graph modelling the string regeneration problem for a three word sentence *seize the day* using a bigram language model. The shortest tour is shown in bold and represents the word sequence `<s> seize the day </s>` with the log probability of $-10.98$.

in Section 3.2 because of the additional start and end of sentence symbols.

The number of vertices and edges in the graph grows with the length $n$ of the sentence $S$ being represented by the graph $G = (V, E)$. The size of the set of vertices $V$ in the graph is $|V| = n + 2$ and the size of the set of edges $E$ is $|E| = |V|^2 = n^2 + 4n + 4$. Note that in the graphs shown in Figures 3.2 and 3.3 we do not show edges with infinite cost, as defined by Equation 3.6, to improve readability.

The graph-based approach presents a notable improvement on the naive approach presented in Section 3.2. Because finding the sentence permutation with the highest probability in the graph equals the Travelling Salesman Problem we can draw several conclusions about the problem we are investigating. Firstly, we can observe that the problem of finding the sentence permutation with the highest probability, and therefore the investigated string regeneration approach, is an NP-hard problem. Secondly, modelling the problem as a TSP still presents a large improvement on the naive approach.
As shown in Section 3.2, the time complexity of the naive approach for a sentence of length $n$ equals $O(n \cdot n!)$. However, as discussed in Section 2.2, the algorithm for solving the TSP with the best-known running time guarantee has the time complexity of $O(n^{2n})$. Although the required time grows exponentially with the length of the sentence, it grows significantly slower than with the factorial time complexity. This is illustrated in Table 3.1.

<table>
<thead>
<tr>
<th>$n$</th>
<th>5</th>
<th>10</th>
<th>15</th>
<th>20</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n^22^n$</td>
<td>800</td>
<td>102,400</td>
<td>7,372,800</td>
<td>419,430,400</td>
</tr>
<tr>
<td>$n \cdot n!$</td>
<td>600</td>
<td>36,288,000</td>
<td>19,615,115,520,000</td>
<td>48,658,040,163,532,800,000</td>
</tr>
</tbody>
</table>

Table 3.1: Illustration of problem size growth at increasing values of $n$ for algorithms with time complexity of $O(n^{2n})$ and $O(n \cdot n!)$.

Finally, by modelling the problem as a TSP we are able to take advantage of the decades of research into the TSP and choose between hundreds of algorithms for solving it. Even though no algorithm with lower running time guarantee than $O(n^{2n})$ has been discovered since the dynamic programming algorithm described by Held and Karp (1962), many algorithms that have no guarantees have been developed since then and the size of the largest solved instance has increased significantly over the years.

### 3.5 Higher N-gram Order Graph-Based Approach

In the previous section we introduced the graph-based approach to solving the string regeneration problem. Finding the sentence permutation with the highest probability under a bigram language model equals solving the Asymmetric Travelling Salesman Problem modelled by the constructed graph. In this section we extend the graph-based approach to higher order N-gram language models.

Higher order N-gram language models use longer context compared to bigram language models when computing the conditional probability of the next
word. This usually results in improved probability estimates for sequences of words. Therefore, to improve our initial approach using bigrams, we extend it to higher order N-grams.

For a general N-gram language model, \( N > 2 \), and a set of \( n \) words \( S, |S| = n \), we define an \( n \)-partite directed weighted graph \( G = (V, E) \), such that given the set of symbols \( X \), \( X = S \cup \{<s>, </s>\} \), the set of vertices is defined as \( V = \{w_i|w_i[j] \in X \text{ for } 1 \leq j \leq N-1, w_i[j] \neq w_i[k] \text{ for } 1 \leq j < k < N\} \). Each vertex is therefore represented by a sequence of symbols \( w_i[1..N-1] \) of length \( N - 1 \) from the set \( X \), and the symbols occurring in the sequence do not repeat themselves. The set of vertices \( V \) is partitioned into \( n \) disjoint independent subsets, \( V_i = \{w_j|w_j \in V, w_j[1] = i\} \), based on the first word in the word sequence, \( w_j[1] \).

Let the set of edges \( E \) be a set of ordered pairs of vertices \((w_i, w_j)\), such that \( E = \{(w_i, w_j)|w_i \in V_k, w_j \in V_l, k \neq l, w_i[2..N-1] = w_j[1..N-2], w_i[1] \neq w_j[N-1]\} \). An edge therefore exists between two vertices if they are parts of two different subsets of \( V \) (have different first word in the sequence), have a matching subsequence, and the words outside the matching subsequence do not repeat between the two vertices.

The edge cost is then defined as:

\[
 c_{ij} = \begin{cases} 
 0 & \text{if } w_i[N-1] = </s> \text{ and } w_j[N-1] = <s>, \\
 -\log P(w_j[N-1]|w_i[1..N-1]) & \text{if } w_i[k] \in S, 2 \leq k \leq N-2 \text{ and } w_i[1] \neq </s> \text{ and } w_j[N-1] \neq <s>, \\
 -\log P(w_j[N-1]|w_i[x..N-1]) & \text{if } x \geq 2 \text{ and } w_i[x] = <s> \text{ and } w_i[x-1] = </s>, \\
 \infty & \text{otherwise.}
\end{cases}
\]

(3.7)

The conditional log probabilities of the form \( \log P(w_j[N-1]|w_i[1..N-1]) \) are computed by an N-gram language model. Consequently, finding the sentence permutation with the highest probability under the N-gram language
We described the Generalized Travelling Salesman Problem (GTSP) in Section 2.3 as finding the shortest tour of length \( s \), visiting exactly one city in each of the \( s \) districts. Each district contains \( m \) cities for a total of \( n = s \cdot m \) cities. In graph \( G \) the districts are defined by the first word in the word sequence associated with each vertex. This means that each word appears exactly once in the solution to the GTSP, ensuring that the solution is a valid permutation. A general example graph with districts for \( N = 3 \) and a sentence of length 3 is shown in Figure 3.4.

An important condition for an edge to exist between two vertices is that the subsequences associated with the vertices match. They match if the following is true: \( w_i[2..N-1] = w_j[1..N-2] \). If two vertices match, they form a word sequence \( w \) of length \( N \). The negative conditional log probability of the last word in the sequence given the previous \( N - 1 \) words equals the cost of the edge between the two vertices.

An additional condition for an edge to exist between two vertices is that the words outside of the required matching subsequence of the two vertices do not repeat between the vertices (condition \( w_i[1] \neq w_j[N-1] \)). For example, two sequences 1 2 3 4 and 2 3 4 1 match according to the condition described above, but outside the required matching subsequence (2 3 4), word 1 appears twice which produces an invalid sentence permutation.

The cases and conditions of the edge cost function presented in Equation 3.7 ensure that the solution is a valid sentence permutation, preventing the start and end of sentence symbols appearing anywhere but at the beginning and ending of the sentence. The third case in Equation 3.7 computes the probabilities of words occurring at the start of the sentence using a backed-off lower N-gram order probability.

Figure 3.5 shows an example graph for the sentence *seize the day* using a trigram language model \((N = 3)\). The shortest tour is shown in bold and represents the word sequence <s> seize the day </s>. The shortest tour equals
the sentence permutation with the highest probability under the trigram language model. The log probability of the permutation is $-11.21$. Note that this log probability differs from the log probability computed for the same sentence permutation in Section 3.4 because the conditional probabilities are computed by a trigram language model instead of a bigram language model.

The size of the vertex and edge set of the graph $G = (V, E)$ grows with the length $n$ of the sentence $S$ and the order $N$ of the $N$-gram language model. The size of the set of vertices $V$ in the graph (including the vertices with repeated words) is $|V| = (n + 2)^{N-1}$ for all values of $N$. The size of the set of edges $E$ (including the infinite cost edges between the full set of vertices) is $|E| = |V|^2 = (n + 2)^{2N-2}$. Note that in the graphs shown in Figures 3.4 and 3.5 we only show edges with non-infinite edge cost to improve readability.
Figure 3.4: A graph modelling the string regeneration problem for a general three word sentence using a trigram language model \((N = 3)\). The graph consists of \(s = 5\) districts, one for each word in the permutation. The vertices are assigned to a district based on the first word of the word sequence associated with the vertex. Each district contains \(m = 5\) vertices for a total of \(n = s \cdot m = 5 \cdot 5 = 25\) vertices. Only edges with non-infinite cost are shown in the graph. Two vertices together form a word sequence of three words. The conditional probability of the final word given the context of the first two words is provided by the trigram language model and equals the cost of the edge between the two vertices. Finding the shortest tour of length \(s\), visiting each district exactly once, equals the GATSP and finding the sentence permutation with the highest probability.
Figure 3.5: A graph modelling the solution to the string regeneration problem for a three word sentence *seize the day* using a trigram language model, \( N = 3 \). The shortest tour visiting each district exactly once is shown in bold and represents the word sequence \(<s>\) seize the day \(</s>\) with the log probability of \(-11.21\). The word sequence with the highest probability can easily be interpreted from the graph by reading the first word in every vertex in the solution tour.
Chapter 4

Implementation

In the previous chapter we described our graph-based approach to the string regeneration problem. We represent the words of the sentence permutation as vertices and use a language model to determine the cost of edges between them. In this formulation, the sentence permutation with the highest probability equals the solution to the Travelling Salesman Problem modelled by the constructed graph. In this chapter we describe the practical aspects of implementing the graph-based approach using a TSP solver and a language model. In the following section we give the overview of our approach and explain our choices of components. In Section 4.2 we focus on language models, while in Section 4.3 we discuss the use of TSP solvers.

4.1 Overview

The graph-based approach represents the problem of finding the sentence permutation with the highest probability as a variation of the TSP. Using a bigram language model, the problem equals solving the Asymmetric TSP. Using a higher order N-gram language model, the problem equals solving the Generalized Asymmetric TSP.

Both variations of the TSP are not as widely studied as the basic TSP and
fewer algorithms exist for solving them. In contrast, state-of-the-art algorithms exist for solving large problem instances of the TSP. One particular TSP solver, the Concorde TSP Solver (Applegate et al., 2006b), is prominent for continuously increasing the size of the largest optimally solved TSP instance over the last two decades. As the Concorde TSP Solver is freely available for academic research use, we have decided to use it with our approach.

The use of the TSP solver makes it necessary to transform the instances of ATSP and GATSP into regular TSP instances. Graph transformations exist that transform an s-partite directed weighted graph modelling the GATSP to a directed weighted graph modelling the ATSP; graph transformations that transform a directed weighted graph modelling the ATSP to an undirected weighted graph modelling the TSP exist as well. The downside of such graph transformations is that they significantly increase the number of vertices in the graph. We implemented two such graph transformations for transforming the GATSP to ATSP and ATSP to TSP instances. We use the latter to transform the graphs based on a bigram language model, and we use both in succession to transform graphs based on a higher order N-gram language model. The transformations are described in detail in Appendix A.

The downside of applying the two graph transformations is that each transformation doubles the number of vertices in the graph. For a higher order N-gram language model this means that the number of vertices is quadrupled after both transformations are applied. Given a sentence of length \( n \), the size of the set of vertices after the transformations equals \( |V| = 2n + 4 \) under a bigram language model and \( |V| = 4 \cdot (n + 2)^{N-1} \) under an N-gram language model for \( N > 2 \). Table 4.1 shows the total size of the vertex set after applying the transformations under several N-gram language models for increasing sentence size.

It is clearly desirable to compute the string regeneration of a sentence as fast as possible. The Concorde TSP Solver can be used to optimally solve very large instances of the TSP. The largest optimally solved instance was the 85,900 city problem described by Applegate et al. (2009). However, it took
Table 4.1: The vertex set size after applying the transformations for several N-gram language models at increasing sentence length.

<table>
<thead>
<tr>
<th>Language Model</th>
<th>5</th>
<th>10</th>
<th>15</th>
<th>20</th>
</tr>
</thead>
<tbody>
<tr>
<td>2-gram</td>
<td>14</td>
<td>24</td>
<td>34</td>
<td>44</td>
</tr>
<tr>
<td>3-gram</td>
<td>196</td>
<td>576</td>
<td>1,156</td>
<td>1,936</td>
</tr>
<tr>
<td>4-gram</td>
<td>1,372</td>
<td>6,912</td>
<td>19,652</td>
<td>42,592</td>
</tr>
<tr>
<td>5-gram</td>
<td>9,604</td>
<td>82,944</td>
<td>334,084</td>
<td>937,024</td>
</tr>
</tbody>
</table>

136 CPU years to compute the optimal solution for the problem instance.

Based on the example sizes of the vertex set shown in Table 4.1 and preliminary experiments we can estimate what can be achieved with the solver. For the purposes of this thesis, we consider a computation time of up to half an hour per sentence on a modern computer as reasonable.

Problem instances with less than 20 words using either a bigram or a trigram language model can be solved in reasonable time using the TSP solver. Using a 4-gram language model for instances of up to 10 words on average requires less than half an hour per sentence to compute. Sentences between 10 and 15 words take considerably more time to compute, up to a few hours per sentence. We were unable to obtain optimal solutions for most sentences longer than 18 words in 12 hours or less at which point we stopped the computation. Using a 5-gram language model even for short sentences the size of the vertex set was too big to solve optimally in 12 hours or less.

We can approach solving the instances that are too long to solve optimally in reasonable time in two ways. The first approach is to back-off to a lower N-gram language model and use it to compute the solution optimally under that language model. The second approach is to use a heuristic algorithm for solving the TSP. Heuristic algorithms do not guarantee finding the optimal solution, but attempt to find the best possible solution given a time constraint. We have experimented with both approaches. We decided to use LKH as the heuristic TSP solver, which is an effective implementation of the Lin-Kernighan heuristic. It currently holds the record for many large TSP instances with unknown optima. Similarly to the Concorde TSP Solver,
LKH is freely available for research use.

4.2 Language Models

A language model is the central piece of our approach to the string regeneration problem. As described in Chapter 3, the underlying assumption of our approach is that the solution to the string regeneration problem is the sentence permutation with the highest probability under a given language model. All the described approaches are methods of finding the sentence permutation with the highest probability as efficiently as possible.

For the experimental evaluation of our graph-based approach we used 2-gram, 3-gram, 4-gram and 5-gram language models. They were built using the SRI Language Modeling Toolkit\(^1\) (Stolcke, 2002) and KenLM Language Model Toolkit\(^2\) (Heafield, 2011). Both are popular open source toolkits for estimating and querying N-gram language models.

When estimating language models it is important to consider the domain of the test dataset. If a language model is estimated on a corpus based on a different domain than the test dataset, the performance of the language model will decrease (Jurafsky and Martin, 2009, Chap. 4). The test datasets used in the experimental evaluation are both in domain of news text. Consequently, the language models were estimated on corpora of news text. The following corpora were used:

- English Gigaword collection V2. Agence France-Presse (English Service, afp_eng) and Xinhua News Agency (English Service, xin_eng) parts of the collection were used.

- NIST Open Machine Translation 2012 Evaluation (OpenMT12) Dataset. Target side of parallel data for Arabic-English and Chinese-English tasks was used.

\(^1\)Available online at http://www.speech.sri.com/projects/srilm/
\(^2\)Available online at http://kheafield.com/code/kenlm/
The total size of the corpus for estimating the language models was 1.16 billion words.

We used the SRILM toolkit network server facility to query the language model for conditional probabilities. The network server facility loads a language model as a network server that listens for requests on a specific port and returns the conditional log probabilities.

This setup is well suited to our approach as it avoids loading the language models to memory for every problem instance. As the language model files tend to be very large for high order N-grams, they take a substantial amount of time to load. Querying the language model as a server means that the language model is loaded only once for all problem instances. Additionally, as described in Section 4.3, we compute the edge cost between pairs of vertices on the spot for which the language model as a server is well suited.

### 4.3 TSP Solvers

One of the main benefits of modelling the string regeneration problem as a Travelling Salesman Problem is that we are able to take advantage of the decades of research into the TSP and choose between hundreds of algorithms for solving it. We briefly discussed the most important approaches developed over the years in Section 2.2. We chose two approaches to solving the TSP. The Concorde TSP Solver\(^3\) is a state-of-the-art TSP solving software which implements a variety of algorithms for optimally solving the TSP and uses the best algorithm depending on the specific problem instance size. LKH\(^4\) is a modern implementation of the Lin-Kernighan heuristic. It currently holds the record for many large TSP instances with unknown optima. Both solvers are freely available for research use. We briefly describe the methods used by the Concorde TSP Solver and LKH in Appendix B. In the remainder of this section we discuss implementation issues related to both solvers.

\(^3\)Available online at http://www.tsp.gatech.edu/concorde/index.html
\(^4\)Available online at http://www.akira.ruc.dk/~keld/research/LKH/
By default, both solvers are given the TSP problem instance in a file defined according to the TSPLIB format (described by Reinelt (1995)). A problem instance can be specified by either listing the graph vertices with their coordinates or explicitly listing the edge costs in a cost matrix. The first approach is possible only with geometric TSP instances. Most of the standard TSP instances included in the TSPLIB are geometric (for example, finding the shortest tour of a large number of actual cities). The edge costs for geometric instances are computed using one of the distance functions provided by the TSP solver (for example, the Euclidean and Manhattan distance functions).

The graphs modelling our string regeneration approach are clearly not geometric. We therefore need to specify the edge cost matrix explicitly. This becomes a problem with long sentences and higher order N-gram language models. For example, a graph modelling an 18 word sentence using a 4-gram language model contains 32,000 vertices after applying both graph transformations. We estimated that the file size required to specify the edge cost matrix of a problem instance of such size would be around 3.8 GB. The same sentence using a 5-gram language model would be represented using 640,000 vertices and the size of the file specifying the edge cost matrix would be around 1.5 TB. This is clearly not a viable approach.

As most of the edge costs are infinite (represented by a very large integer) the cost matrix is sparse. A custom file format can be defined listing only edges with non-infinite edge cost. However, even that would result in a file of substantial size. We instead defined a file format listing only basic information such as the list of words in the sentence and the total number of nodes. We modified both TSP solvers to compute the edge cost on the spot. Given two vertices we query a language model server (described in Section 4.2) and use the negated conditional log probability as the cost of the edge between the vertices. The process of identifying words associated with the individual vertex, which is represented only by a number, is further complicated by the two graph transformations.

Although we have not performed benchmarking tests, we expect that querying a language model server using sockets introduces delays in the TSP solver
computation. However, given the alternative we believe this is a preferable solution.

Both TSP solvers represent the edge costs as integers and do not support floating point numbers. As the conditional log probabilities provided by the language model server are floating point numbers, we multiplied each log probability with a factor of $1000$ and rounded the result to the closest integer, effectively rounding the log probability to three decimal places.
Chapter 5

Evaluation

In this chapter we describe the procedures and the results of the experimental evaluation of the graph-based approach to the string regeneration problem.

We evaluated three different versions of the graph-based approach described in the previous chapter. This includes the systems based on bigram, trigram, and 4-gram language model. We evaluated each version of the system on three datasets of news sentences by computing the dataset-wide BLEU scores.

In the following section we discuss the suitability of the BLEU metric for measuring the fluency of the regenerated sentences. In Section 5.2 we describe the datasets used for the evaluation and the required preprocessing steps. In Section 5.3 we report and discuss the automatic evaluation results. Finally, in Section 5.4 we describe and report the results of manual evaluation of the output of the three systems.

5.1 Evaluation metric

The BLEU evaluation metric was developed as an inexpensive and fast method of measuring incremental progress of statistical machine translation systems. Espinosa et al. (2010) have investigated the use of various automatic evaluation metrics to measure the quality of Natural Language Generation
output. They found that several metrics, including BLEU, correlate moderately well with human judgements of fluency. They conclude that machine translation evaluation metrics are useful for evaluation of Natural Language Generation output, but should be used with caution, especially when comparing different systems.

Consequently, as the string regeneration problem is a basic form of Natural Language Generation (as discussed in the Introduction chapter), we believe BLEU is an appropriate measure of the system’s performance with regards to fluency of the output. To ensure that the BLEU scores of individual versions of the graph-based approach reflect actual changes in output quality, we provide examples of output and also conduct a manual evaluation.

As described in Section 2.4, the BLEU metric measures closeness of a candidate translation to a reference translation using N-gram precision. From a practical perspective with the string regeneration problem, the candidate translation equals the regenerated sentence and the reference translation equals the original sentence. We therefore measure closeness of the regenerated sentence to the original sentence using N-gram precision.

We used the case insensitive NIST BLEU script v13¹ against tokenized references to compute the BLEU scores.

5.2 Datasets and Preprocessing

We evaluated the graph-based approach to the string regeneration problem on three datasets, MT08, MT09, and SR11. In this section we describe the origin of the datasets, their properties, and the preprocessing steps taken to prepare them for evaluation.

MT08 The MT08 dataset is the target side (the 1st reference) of the Arabic-English newswire part of the 2008 NIST Open Machine Translation

¹Available online at: ftp://jaguar.ncsl.nist.gov/mt/resources/mteval-v13.pl
evaluation competition\textsuperscript{2}.

**MT09** The MT09 dataset is the target side (the 1st reference) of the Arabic-English newswire part of the 2009 NIST Open Machine Translation evaluation competition.

**SR11** The SR11 dataset is the plain text representation of the news text dataset used for the Surface Realisation Task at Generation Challenges 2011\textsuperscript{3}.

The MT08, MT09, and SR11 datasets contain 813, 586, and 2398 sentences respectively.

The first preprocessing step performed on all three datasets was de/tokenization. Tokenization is a heuristic procedure that separates the sentence into distinct tokens that are used in the string regeneration problem. Take the sentence *Dr. Webb’s cat died, she said*. The tokenization procedure needs to separate the sentence into *Dr. Webb’s cat died, she said*. The exact procedure and rules depend on the application. Detokenization is the reverse process, which is sometimes necessary to perform on a dataset in order to tokenize it differently or obtain the original sentences.

The distribution of sentence length for each dataset after de/tokenization is shown in Figure 5.1. Most sentences in the MT08 and MT09 datasets are between 10 and 50 tokens long. However, the distribution is spread out with a distinct tail containing long sentences and some outlier sentences with over 100 tokens. On the other hand, the SR11 sentence length distribution resembles normal distribution, with mean sentence length of around 20 tokens.

One of the challenges of the string regeneration problem, which we discussed in the Introduction chapter, is dealing with very short and very long sentences. We reasoned that it is difficult to regenerate very short and very long sentences, both for humans and machines, as a set words does not put any constraints on the solution. Regenerating such sentences is mainly guess-

\textsuperscript{2}More information can be found online: http://www.itl.nist.gov/iad/mig/tests/mt/

\textsuperscript{3}More information can be found online: http://www.nltg.brighton.ac.uk/research/sr-task/
Figure 5.1: The distribution of sentence length for MT08, MT09, and SR11 dataset after de/tokenization.

Figure 5.2: The distribution of sentence length for MT08, MT09, and SR11 dataset after chopping of sentences.
work. The purpose of the string regeneration problem is to demonstrate the capabilities of a system to improve fluency of a sentence by reordering its words. Evaluating the system’s performance on test sentences which require guesswork is undesirable and possibly harmful to the evaluation process.

As the datasets contain many long sentences we have taken preprocessing steps to chop each long sentence into manageable parts, which is a common practice in the field. Based on preliminary experiments we decided to limit the maximum length of the chopped sentence to 20 tokens. The chopping procedure was automated and used heuristic judgement to determine appropriate chopping points in the sentence, such as punctuation. Beside limiting the length of long sentences, we ignored short sentences containing 4 or fewer tokens. The chopped dataset length distribution is shown in Figure 5.2.

Each chopped sentence was regenerated separately and the regenerated chopped sentences were concatenated to form the original number of dataset sentences. We expect that the described preprocessing steps increased the reported BLEU scores to a certain degree. However, all systems compared in the experimental evaluation were subject to the same conditions and their scores are therefore comparable.

5.3 Automatic Evaluation

In this section we report and discuss the results of automatic evaluation of three versions of the graph-based approach, based on bigram, trigram, and 4-gram language models, on the three datasets described in the previous section.

In Chapter 4 we discussed the practical issues regarding the computation time at increasing sentence length and increasing N-gram order. We consider a computation time of up to half an hour per sentence on a modern computer as reasonable. After performing preliminary experiments we concluded that all sentences of length up to 20 can be optimally solved using a bigram or a trigram language model.
Using a 4-gram language model we can optimally solve sentences of up to 10 tokens in reasonable time, and most sentences of up to 18 tokens in less than 12 hours. The 12 hour time frame is clearly unreasonable for any practical application of the approach. However, as shown in Figure 5.2, only around a third of the chopped dataset sentences are of length 10 or less. Because we would like to evaluate the improvements offered by higher N-gram order language models, we decided to compute the optimal solutions under a 4-gram language model for sentences with up to 18 tokens. This includes the majority of chopped sentences. We used two approaches to solve the remaining sentences and combine them with the optimally solved sentences. The first approach is using the LKH heuristic algorithm with a set time limit. The second approach is to back-off to the trigram graph-based approach.

The BLEU scores for the four systems are reported in Table 5.1. The trigram graph-based approach performed considerably better than the bigram approach, increasing the BLEU score for 10 BLEU points or more on all three datasets. The 4-gram approach augmented with a heuristic TSP solver performed significantly worse than the trigram approach on MT08 and MT09 datasets, while performing better than the trigram approach on SR11 dataset. The reason for this difference is the different distribution of chopped sentence lengths between the three datasets. We can observe in Figure 5.2 that around one fourth of all chopped sentences in MT08 and MT09 datasets are longer than 18 tokens. On the other hand, less than 1% of chopped sentences of the SR11 dataset are longer than 18 tokens. This means that a significant part of the MT08 and MT09 datasets was solved using the heuristic approach, compared to a small part of the SR11 dataset. Using a heuristic TSP solver therefore clearly negatively affects the performance of the system. The 4-gram approach backing-off to the trigram approach achieved a higher BLEU score than the trigram approach over all datasets. However, the increase in BLEU score is smaller than the increase from the bigram to the trigram approach.

In Figure 5.3 we show the BLEU scores of three versions of the system, the bigram, trigram, and the 4-gram system with trigram back-off, at increasing
Table 5.1: BLEU scores for four versions of the graph-based approach, based on bigram, trigram, and 4-gram language models. We used the 4-gram language model approach on sentences of up to length 18. The remaining sentences were computed using either a heuristic TSP solver (opt +heur) or by backing-off to a trigram language model approach (4g +3g).

<table>
<thead>
<tr>
<th>LM</th>
<th>Solver</th>
<th>MT08</th>
<th>MT09</th>
<th>SR11</th>
</tr>
</thead>
<tbody>
<tr>
<td>2g</td>
<td>opt</td>
<td>44.4</td>
<td>45.1</td>
<td>40.6</td>
</tr>
<tr>
<td>3g</td>
<td>opt</td>
<td>57.9</td>
<td>58.0</td>
<td>50.2</td>
</tr>
<tr>
<td>4g</td>
<td>opt +heur</td>
<td>44.8</td>
<td>42.6</td>
<td>51.7</td>
</tr>
<tr>
<td>4g +3g</td>
<td>opt</td>
<td>59.1</td>
<td>59.5</td>
<td>51.8</td>
</tr>
</tbody>
</table>

Figure 5.3: BLEU scores of three versions of the system, the bigram, trigram, and the 4-gram system with trigram back-off, at increasing sentence length over combined MT08, MT09, and SR11 datasets.
sentence length over all three datasets. As in the results reported in Table 5.1, the BLEU scores shown in Figure 5.3 were computed after the regenerated chopped sentences were concatenated to form original sentences. We can observe that with increasing length of sentences the BLEU scores decrease significantly. The differences in the performance of the three systems are bigger with shorter sentences and decrease slightly with longer sentences.

In Figure 5.4 we show examples of regenerated sentences for three versions of the system. In the first example, we can see the improvements in the output fluency with better versions of the system. The improvements are reflected by the BLEU scores. The 4-gram output can be considered completely fluent. However, when compared to the original sentence, its BLEU score is not 100 because the numbers are switched. In this regard, BLEU score is harsh and not an ideal evaluation metric for the task. In the second example, the original sentence contains complicated wording which is reflected in poor performance of all three versions of the system, despite the high BLEU score of the trigram system. In the final example, we can observe the gradual improvement of fluency over the three versions of the system. This is reflected by the BLEU score, which reaches 100.0 for the 4-gram system, which produced an identical sentence to the original.

5.4 Manual Evaluation

In Section 5.1 we discussed the use of the BLEU metric for evaluation of Natural Language Generation output. We concluded that BLEU can be used to measure fluency of the NLG output, but needs to be used with caution. In order to ensure that the BLEU scores reported in the previous section reflect actual changes in the output quality, we conducted a manual evaluation. In this section, we describe the manual evaluation experiment design and results.

We manually evaluated three versions of the graph-based approach: the bigram, trigram, and 4-gram using trigram as back-off. We used a pairwise
<table>
<thead>
<tr>
<th>REF</th>
<th>Hypothesis</th>
<th>BLEU</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>meanwhile, Azim stated that 10 people were killed and 94 injured in yesterday’s clashes.</td>
<td></td>
</tr>
<tr>
<td>(a)</td>
<td>meanwhile, Azim and 10 people were injured in clashes yesterday’s stated that killed 94.</td>
<td>21.4</td>
</tr>
<tr>
<td>(b)</td>
<td>Azim, meanwhile stated that 94 people were killed and 10 injured in yesterday’s clashes.</td>
<td>50.4</td>
</tr>
<tr>
<td>(c)</td>
<td>meanwhile, Azim stated that 94 people were killed and 10 injured in yesterday’s clashes.</td>
<td>66.3</td>
</tr>
<tr>
<td></td>
<td>Zinni indicated in this regard that President Mubarak wants Egypt to work with the West.</td>
<td></td>
</tr>
<tr>
<td>(a)</td>
<td>Egypt wants Zinni in this regard to work with President Mubarak indicated that the West.</td>
<td>24.9</td>
</tr>
<tr>
<td>(b)</td>
<td>Zinni wants Egypt to work with the West that President Mubarak indicated in this regard.</td>
<td>63.4</td>
</tr>
<tr>
<td>(c)</td>
<td>work with Zinni indicated that President Mubarak wants the West to Egypt in this regard.</td>
<td>30.6</td>
</tr>
<tr>
<td></td>
<td>He stressed that this direction is taking place in all major cities of the world.</td>
<td></td>
</tr>
<tr>
<td>(a)</td>
<td>He stressed that the world is taking place in this direction of all major cities.</td>
<td>33.9</td>
</tr>
<tr>
<td>(b)</td>
<td>He stressed that all major cities of the world is taking place in this direction.</td>
<td>58.0</td>
</tr>
<tr>
<td>(c)</td>
<td>He stressed that this direction is taking place in all major cities of the world.</td>
<td>100.0</td>
</tr>
</tbody>
</table>

Figure 5.4: Output examples of three versions of the graph-based approach: (a) bigram, (b) trigram, and (c) 4-gram with trigram back-off. The original sentence is given for each of the three examples. Sentence BLEU scores are shown for each regenerated sentence.
evaluation of the three systems: for each evaluation sentence, we compared the output of a pair of systems and asked which output is more fluent.

We used the crowdsourcing website CrowdFlower\textsuperscript{4} to gather fluency judgments. Judges were asked 'Please read both sentences and compare the fluency of sentence 1 and sentence 2.' They were given three options: 'Sentence 1 is more fluent', 'Sentence 2 is more fluent', 'Sentence 1 and Sentence 2 are indistinguishable in fluency'. The order of presentation of the two systems was randomized for each sentence.

100 sentences of length between 5 and 18 tokens were chosen randomly from the combined MT08 and MT09 dataset. We gathered 5 judgements for each sentence of a single pairwise comparison of two systems. Each pairwise comparison of two systems is therefore based on 500 human judgements.

The CrowdFlower platform additionally offers a method of ensuring the quality of judgements using gold standard questions. A gold standard question is a single question for which the answer is known. We formulated the gold standard questions by randomly choosing additional 10 sentences from the joint MT08 and MT09 dataset with identical length restrictions. For each of the 10 sentences we chose one of the three graph-based systems and paired it with the original reference sentence. The original sentence is therefore the correct answer to a gold standard question.

The platform measures the reliability of judges by randomly posing gold standard questions in between regular questions. If a judge incorrectly answers a large percentage of gold standard questions, his judgements are deemed unreliable and are not used in the final result set. The judge is still paid the full amount for his work.

Using the gold standard questions we try to improve the quality of human judgements. A thorough discussion of suitability and reliability of crowdsourcing for NLP and SMT tasks and related ethical concerns can be found in: Snow et al. (2008); Zaidan and Callison-Burch (2011); Fort et al. (2011).

\textsuperscript{4}Website: http://crowdflower.com/
The pairwise comparison results are shown in Table 5.2. Each number represents the proportion of the human judgements that rated the output of the row system as better than the column system. The raw numbers of pairwise comparison judgements in favor of each system are shown in Table 5.3. A one-sided sign test indicated that we can reject the null hypothesis of the two systems being equal in favor of the alternative hypothesis of the first system being better than the second for all three system pairings: 3g and 2g, 4g and 2g, and 4g and 3g, \( p < 0.001 \) for all three comparisons. The results therefore confirm the BLEU score differences between the three graph-based systems.

<table>
<thead>
<tr>
<th>LM</th>
<th>2g</th>
<th>3g</th>
<th>4g</th>
</tr>
</thead>
<tbody>
<tr>
<td>2g</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>3g</td>
<td>65.4</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>4g</td>
<td>72.9</td>
<td>69.2</td>
<td>-</td>
</tr>
</tbody>
</table>

Table 5.2: Manual evaluation results of pairwise comparison between three versions of the system, the bigram, trigram, and the 4-gram system with trigram back-off. The numbers represent the percentage of judgements in favor of the row system when paired with the column system. The judgements marking the output of the systems as indistinguishable were not counted towards any of the systems.

<table>
<thead>
<tr>
<th>sys1</th>
<th>sys2</th>
<th>sys1 equal</th>
<th>sys2</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>2g</td>
<td>3g</td>
<td>124</td>
<td>142</td>
<td>234</td>
</tr>
<tr>
<td>2g</td>
<td>4g</td>
<td>102</td>
<td>124</td>
<td>274</td>
</tr>
<tr>
<td>3g</td>
<td>4g</td>
<td>92</td>
<td>201</td>
<td>207</td>
</tr>
</tbody>
</table>

Table 5.3: The raw numbers of pairwise comparison judgements between the three systems. The columns give the number of judgements in favor of the first or the second system and the number of judgements that deemed the fluency of the outputs as indistinguishable.
Chapter 6

Graph-Based Approach with Phrase Constraints

In the previous chapter we conducted an experimental evaluation of the graph-based approach to the string regeneration problem. We showed that the approach performs well with higher order N-gram language models. Although the approach using a 4-gram language model with back-off to trigrams performed best, the considerable computation time for sentences of even moderate length makes it impractical. The trigram graph-based approach performed slightly worse compared to the 4-gram approach, but the required computation time is significantly shorter for all sentences. In this chapter we describe and evaluate an improved version of the graph-based approach using phrases instead of words.

The string regeneration problem is concerned with constructing a fluent sentence from an unordered set of words. Consequently, the input of the string regeneration problem is relatively unconstrained. Alternatively, additional information about the sentence being generated can be provided in the form of constraints between the words. The constraints can be stated in the form of shallow syntactic dependency trees, deep semantic graphs, or other meaning representations. A simpler form of constraints are phrases. A phrase constraint states that a number of words should occur together in the generated
sentence. Under phrase constraints, the string regeneration problem changes from finding the best permutation of words to finding the best permutation of phrases, which can include 1-word phrases. It is expected that specifying additional constraints on the input increases the quality of the output, as the constraints provide additional information that is not available from the basic unordered set of words.

The remainder of the chapter is structured as follows. In the following section we describe the adaptation of the graph-based approach to enable specification of phrase constraints. In Section 6.2 we discuss how the phrase constraints can be obtained. Finally, in Section 6.3 we report and discuss the evaluation results of the phrase graph-based approach.

### 6.1 Graph-Based Approach with Phrase Constraints

We described the graph-based approach to solving the string regeneration problem in detail in Chapter 3. We model finding the sentence permutation with the highest probability under a language model as a (Generalized) Asymmetric Travelling Salesman Problem. The (G)ATSP is modelled by a directed graph consisting of vertices, which are associated with a word or a sequence of words, and edges, whose cost equals the language model probabilities.

Intuitively, we can adapt the bigram graph to include the phrase constraints by forcing certain edges to occur in the solution tour. This is achieved by changing their cost to a large negative number. An example of including a single phrase constraint in a graph is shown in Figure 6.1. The graph represents the same problem as the one shown in Figure 3.3. However, a constraint has been added to the graph, requiring the phrase *day seize* to occur in the final solution. The constraint can be seen as the modified edge cost between the *day* and *seize* vertices, which now equals a large negative
number. The highest probability sentence permutation under the constraint is found in the same manner as before by solving the ATSP modelled by the modified graph. The shortest tour in the graph is shown in bold and includes the required phrase.

Figure 6.1: A graph modelling the string regeneration problem for a three word sentence seize the day using a bigram language model with a constraint that the phrase day seize occurs in the final solution. The constraint is enforced by modifying the relevant edge cost to equal a large negative number. The shortest tour under the constraint is shown in bold and represents the word sequence <s> the day seize </s>.

Extending the described approach to include many long phrases in the same graph under a bigram language model is straightforward. Extending the phrase constraints to graph-based approaches using higher order N-gram language models requires some additional explanation. A phrase with \( N \) or more words is introduced to the graph-based approach under an N-gram language model by modifying several relevant edge costs. Consider a part of an example graph for a trigram language model shown in Figure 6.2. Adding a three word phrase constraint \( w_1w_2w_3 \) to the graph modifies not only the edge cost between vertices \( w_1w_2 \) and \( w_2w_3 \), but also the edge cost of all edges
that form a partial phrase, for example $w_0w_1w_2$ and $w_2w_3w_4$.

![Figure 6.2: A part of an example graph for a trigram language model demonstrating the edges modified in order to introduce the $w_1w_2w_3$ phrase constraint.](image)

A phrase with $N - 1$ or fewer words in a graph based on an N-gram language model is not associated with a specific edge, but with a vertex or a set of vertices. In this case, all incoming and outgoing edges of the vertex or the set of vertices are modified. Consider the example shown in Figure 6.3. To introduce the $w_1w_2$ phrase constraint into the trigram-based graph, all incoming and outgoing edges of the $w_1w_2$ vertex are modified. A two word phrase in a graph based on a 4-gram language model is associated with a set of vertices. Every incoming and outgoing edge of the set of vertices is then modified to introduce the phrase constraint.

![Figure 6.3: A part of an example graph for a trigram language model demonstrating the edges modified in order to introduce the $w_1w_2$ phrase constraint.](image)

Despite the considerably more complex formulation of phrase constraints in graphs based on higher order N-gram language models, the highest probabili-
ity sentence permutation under the constraints is found in the same manner as before by solving the GATSP modelled by the modified graph.

6.2 Phrase Selection

There are two distinct approaches to introducing phrase constraints into the string regeneration problem. The phrase constraints can be either specified with the unordered list of words as additional information given to the string regeneration system, or the string regeneration system can use additional methods and resources to try and infer the phrase constraints from the list of unordered words. In the former case, the phrase constraints are guaranteed to be correct as far as the string regeneration system is concerned. In the latter case, however, the system can introduce incorrect phrases which can potentially degrade the performance.

We investigated the latter approach of inferring the phrases based on the unordered list of words. We used the monolingual English data corpus for NIST Open Machine Translation 2012 Evaluation\(^1\) to provide information regarding occurrences of phrases. The phrases of length up to 5 words and containing only a subset of words occurring in the input set were extracted from the corpus along with their counts. The phrase selection problem is concerned with deciding which combination of phrases extracted from the corpus is used as the set of constraints on the string regeneration problem.

We can represent each phrase with a triple (coverage, words, count). A 3 word phrase for a 6 word sentence might look like (010011, \(w_1w_2w_3, n\)). The binary notation represents which words of the input set are covered by the phrase. The counts for longer phrases are usually much lower than for shorter phrases.

We developed the following scheme for scoring each individual phrase. Let \(p\) be a phrase of length \(|p|\) and \(c_p\) be the number of times it occurs in the

\(^1\)More information about the task can be found online: www.nist.gov/itl/iad/mig/openmt12.cfm
corpus. Let $p_i$ be the $i$-th word in the phrase $p$ and $c_p$ the number of times the $i$-th word occurs in the corpus. Because the phrases associated with the permutation include 1 word phrases, we have counts for all individual words that can make up the multi-word phrases.

Let $s_p$ be the score a multi-word phrase. It is computed in the following manner:

$$s_p = \max_{1 \leq i \leq |p|} (1 + \frac{c_p}{c_{p_i}})^{|p|}$$  \hspace{1cm} (6.1)$$

The score of a phrase is therefore the maximum ratio of the phrase count and the count of individual words of the phrase. This tells us how often the phrase occurred relative to the word in phrase that has occurred the least number of times. We take the maximum ratio as base with the length of the phrase as exponent to increase the otherwise low ratios of longer phrases. As some of the phrase counts can be very large we take the logarithm of the score to prevent computational overflow.

The score of a combination of phrases $x$ is computed in the following manner:

$$s_x = \min_{p \in x} (s_p)^{\sum_{p \in x} |p|^\alpha}$$  \hspace{1cm} (6.2)$$

Using the $\alpha$ parameter we are able to vary our preference towards longer phrase combinations.

The search procedure for the best combination of phrases starts with a single multi-word phrase in each combination. It groups all combinations of phrases with the same coverage (that is, an identical binary coverage vector) and keeps only the highest scoring combination. The combinations of phrases are then joined to form combinations with new coverage vectors through several iterations until no further changes can be made. The best combination is chosen as the one with the highest score irrespective of its coverage vector.
The combination is amended with the missing 1 word phrases to form a full coverage of the input sentence.

This method of selecting phrases was developed based on our intuitions of what type of phrases should be included as constraints to the string regeneration approach. A more thorough investigation of the phrase selection should be conducted to construct a method with sound theoretical backing. Potential areas of interest in this regard are information theory and information retrieval.

6.3 Evaluation

We evaluated the graph-based approach with phrase constraints under a bigram and trigram language models on the MT09 dataset described in Section 5.2. We varied the $\alpha$ parameter from $\alpha = 0.5$ to $\alpha = 8$ to change our preference for inclusion of longer phrase constraints.

The BLEU scores are reported in Table 6.1. The results are disappointing. The inclusion of selected phrase constraints resulted in a significant drop in BLEU scores for both systems at all values of parameter $\alpha$. The drop for the bigram based system is around 5 BLEU points or more, while for the trigram based system the drop is around 20 BLEU points. There is some variation in BLEU scores with the changing $\alpha$ values. For the bigram based system increasing our preference towards longer phrases results in a drop of around 1 BLEU point from $\alpha = 0.5$ to $\alpha = 8$. For the trigram based system increasing our preference towards longer phrases resulted in an improvement of around 1.5 BLEU point.

The most likely reason for the poor results is the phrase selection procedure. The phrase selection procedure attempts to determine which words from the input set should be used together as a phrase. The procedure described in Section 6.2 does not use any additional information provided alongside the input set of words, but instead attempts to infer phrases based on corpus counts. Statistics describing the selected phrases for the MT09 dataset are
Table 6.1: BLEU scores for the graph-based approach with phrase constraints under bigram and trigram language models. We varied the $\alpha$ parameter in Equation 6.2 to change our preference for inclusion of longer phrase constraints. The baseline performance without phrase constraints is reported for comparison.

<table>
<thead>
<tr>
<th>System</th>
<th>$\alpha$</th>
<th>MT09</th>
</tr>
</thead>
<tbody>
<tr>
<td>2g opt</td>
<td>0.5</td>
<td>39.5</td>
</tr>
<tr>
<td></td>
<td>1.0</td>
<td>39.0</td>
</tr>
<tr>
<td></td>
<td>2.0</td>
<td>38.9</td>
</tr>
<tr>
<td></td>
<td>4.0</td>
<td>38.8</td>
</tr>
<tr>
<td></td>
<td>8.0</td>
<td>38.4</td>
</tr>
<tr>
<td>3g opt</td>
<td>0.5</td>
<td>37.0</td>
</tr>
<tr>
<td></td>
<td>1.0</td>
<td>36.5</td>
</tr>
<tr>
<td></td>
<td>2.0</td>
<td>37.8</td>
</tr>
<tr>
<td></td>
<td>4.0</td>
<td>38.5</td>
</tr>
<tr>
<td></td>
<td>8.0</td>
<td>38.6</td>
</tr>
</tbody>
</table>

Table 6.2: The phrase selection statistics for each value of parameter $\alpha$. The statistics are computed only on phrases of length 2 or more. We report the average phrase length, the number of times a selected phrase occurred in the original sentence, the total number of phrases selected, and the accuracy of phrase selection ($\text{Accuracy} = \text{Matches} / \text{Total}$).

<table>
<thead>
<tr>
<th>$\alpha$</th>
<th>Avg Length</th>
<th>Matches</th>
<th>Total</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.5</td>
<td>3.8</td>
<td>16,300</td>
<td>18,339</td>
<td>88.9%</td>
</tr>
<tr>
<td>1.0</td>
<td>3.9</td>
<td>16,021</td>
<td>18,120</td>
<td>88.4%</td>
</tr>
<tr>
<td>2.0</td>
<td>4.0</td>
<td>15,593</td>
<td>17,750</td>
<td>87.8%</td>
</tr>
<tr>
<td>4.0</td>
<td>4.3</td>
<td>14,583</td>
<td>16,919</td>
<td>86.2%</td>
</tr>
<tr>
<td>8.0</td>
<td>4.7</td>
<td>13,097</td>
<td>15,673</td>
<td>83.6%</td>
</tr>
</tbody>
</table>
shown in Table 6.2. We measured the number of times a selected phrase has actually occurred in the original sentence. The accuracy of the selected phrases is the highest for $\alpha = 0.5$ and equals 88.9%. This means that more than 10% of selected phrases have not occurred in the solution sentence. Every such mistake has a significant influence on the search for the best permutation under the phrase constraints.

We show two example phrase selections in Figure 6.4. In the first example we can see that the phrase *suspicious of* was included in the selected phrases. The phrase is very common in English language. It is therefore not surprising that it occurred many times in the corpus and was consequently included in our selection. However, as the example shows, the phrase does not actually occur in the original sentence. This mistake highlights the problem with our phrase selection procedure. Relying only on phrase counts does not produce reliable phrases to be included as phrase constraints.

There are several approaches how this deficiency can be tackled. As mentioned in Section 6.2, a more thorough investigation of the phrase selection from a corpus should be conducted to construct a method with sound theoretical backing. Potential areas of interest in this regard are information theory and information retrieval. Additionally, other methods of constructing phrases from a set of unordered words should be investigated, such as building dependency trees over the words. Lastly, the graph-based approach using phrase constraints should yield far better results if the phrase constraints are provided alongside the unordered set of words. This approach could be considered as sentence generation from a shallow representation, instead of an unconstrained string regeneration problem.
bleu
refs
phrases
output
britain arrests in several cities and explosion of suspicious
car.
suspicious of explosion in cities.
br britain several arrests and suspicious of car explosion in
cities.

<table>
<thead>
<tr>
<th>REF</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Phrases</td>
<td>britain arrests in several cities and explosion of suspicious car.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>britain several arrests and suspicious of car explosion in cities.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>REF</td>
<td>Output</td>
</tr>
<tr>
<td>Phrases</td>
<td>links established between the failed london and glasgow attacks and</td>
</tr>
<tr>
<td></td>
<td>fears of new imminent attacks .</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>glasgow and established links between the attacks failed london</td>
</tr>
<tr>
<td></td>
<td>new attacks and fears of imminent .</td>
</tr>
</tbody>
</table>

| REF       | Output                                                                 |
| Phrases   |                                                                         |
|          |                                                                         |
|          |                                                                         |
|          |                                                                         |
|          |                                                                         |
|          |                                                                         |

<table>
<thead>
<tr>
<th>BLEU</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.8</td>
</tr>
<tr>
<td>24.6</td>
</tr>
</tbody>
</table>

Figure 6.4: Phrase selection and regenerated output for two example sentences. Sentences were regenerated under a bigram language model and $\alpha = 0.5$. Selected phrases are separated with vertical lines.
Chapter 7

Related Work

The basic task of all Natural Language Generation approaches is to take some sort of meaning representation as input and generate human-readable output. The approaches differ on how much information is required from the meaning representation. Deep representation can include dependency graphs annotated with semantic labels and other syntactic information (Belz et al., 2011). Shallow representations can include syntactic dependency trees annotated with POS tags and other syntactic information (Belz et al., 2011), IDL-expressions (Soricut and Marcu, 2005), and Abstract Meaning Representation (Langkilde and Knight, 1998). Taking an unordered set of words, which is the string regeneration problem we investigate in this thesis, can be considered as the most unconstrained and shallowest meaning representation provided to a natural language generation system.

Soricut and Marcu (2005) consider natural language generation in context of other popular natural language applications, such as Machine Translation, Summarization, and Question Answering. They view these as text-to-text applications that produce textual output from textual input. Because of this, many natural language applications need to include some form of natural language generation to produce the output text. However, the natural language generation in these applications is often handled in an application-specific way. They propose to use IDL-expressions as an application-independent
representation language for text-to-text natural language generation. They evaluate their approach on the string regeneration task and achieve moderate BLEU scores.

Wan et al. (2009) approach the string regeneration problem using dependency spanning trees. Their approach is to search for the most probable dependency tree containing each word in the input or, equally, finding the optimal spanning tree. Zhang and Clark (2011) propose a similar approach using Combinatory Categorial Grammar (CCG) which imposes stronger category constraints on the parse structure compared to dependency trees investigated by Wan et al. (2009). They primarily focus on the search problem of finding an optimal parse tree among all possible trees containing any choice and ordering of the input words. The CCG approach achieved higher BLEU scores compared to the approach proposed by Wan et al. (2009). Zhang et al. (2012) improve the CCG approach described in Zhang and Clark (2011) by incorporating an N-gram language model.

The purpose of studying and building approaches to solving the string regeneration problem is to improve grammaticality and fluency of machine generated text. As the input is an unordered set of words, any approach solving the string regeneration problem can be used to improve any machine generated text. In contrast to the general approach, application-specific approaches for improving machine generated text fluency exist. For example, Blackwood et al. (2010) propose to use Minimum Bayes Risk decoding to improve the output of Statistical Machine Translation systems. Fluency in generated output is a concern in speech synthesis applications as well (Sundaram and Narayanan, 2003).

Finally, approaches solving the string regeneration problem could be helpful in any NLP task which specifies a required vocabulary, e.g. words and phrases, but gives the system the freedom to produce alternative outputs.
Chapter 8

Conclusions and Future Work

8.1 Summary

In the thesis we investigated a graph-based approach to the string regeneration problem of recovering a fluent version of the original sentence given an unordered set of words. The approach works under the assumption that the best permutation of the input words is the one with the highest probability under a language model. We explored different approaches to finding the highest probability permutation which, beside the graph-based approach, also include a naive approach and an approach using constraint satisfaction.

The graph-based approach models the problem as a set of vertices containing words and a set of edges between the vertices, whose cost equals language model probabilities. Finding the permutation with the highest probability in the graph formulation is equal to finding the shortest tour in the graph or, equally, solving the Travelling Salesman Problem. We formulated the graph-based approach for a bigram language model and extended it to higher order N-gram language models.

We evaluated the graph-based approaches on three datasets. The BLUE scores and example output indicated that the graph-based approach is successful in constructing a fairly fluent version of the original sentence, but
some systematic problems, such as long-range agreement not captured by N-gram language models, still remain. The bigram based approach performed moderately well but was surpassed by the trigram based approach. The 4-gram based approach offered an improvement on the trigram but is not of much practical use due to its long computation times.

We confirmed the results of automatic evaluation by conducting a manual evaluation. The human judges were asked to compare the outputs of two systems and decide which is more fluent. The results are statistically significant and confirm the ranking of the systems obtained using the BLEU scores.

Finally, we extended the graph-based approach to enable the use of phrases instead of just words. We developed a phrase selection strategy using an additional corpus. The experimental evaluation showed a large decrease in BLEU scores. We discussed the possible reasons for this result and suggested future improvements.

8.2 Future Work

The graph-based approach to the string regeneration problem can be extended in a number of ways.

In Statistical Machine Translation and other Natural Language Processing applications a common method of improving the final output of the system is to use the basic system to generate an N-best list of outputs and use a different method to rerank the N-best list of outputs to choose the best one. The methods can range from rescoring the outputs with a higher-order language model or a dependency language model, to using discriminative machine learning.

An effective approach to finding N shortest tours through a graph could be useful for providing multiple reordered candidates. Here we describe the intuition behind a naive approach to finding N shortest tours in the graph. After finding the shortest tour in the graph containing \( n \) edges, we repeat
the search procedure \( n \) times, removing a single edge of the shortest tour from each repeated search. In this way we ensure that each of the \( n \) shortest tours found in the second step does not equal the original shortest tour. The second shortest tour in the original graph consequently equals the shortest of the \( n \) shortest tours. Finding the third shortest tour in the graph requires us to remove a pair of edges from the original graph, one from the shortest tour and one from the second shortest tour. This means that in order to find the third shortest tour, the search has to be repeated another \( n^2 \) times. Finding each subsequent shortest tour increases the total number of required TSP solver runs significantly.

This naive approach is obviously unsuitable for obtaining an N-best list of any significant size. The approaches of reranking the outputs mentioned above usually consider a few hundred or a thousand best outputs. For this to be possible with the graph-based approach, a better method of obtaining the N-best list needs to be discovered.

The extended graph-based approach with phrase constraints includes a phrase selection method for choosing which phrases occur in the final output. The phrase selection method is based on our intuitions of what type of phrases should be included as constraints. A more thorough investigation of the phrase selection should be conducted to construct a method with sound theoretical backing. Potential areas of interest in this regard are information theory and information retrieval. Additionally, it would be interesting to introduce other linguistically motivated constraints into the graph by modifying its vertices, edges, and edge costs.

Another interesting area of future research relating to the wider string regeneration problem is determining the human performance on the task. Based on a simple trial of trying to regenerate a long sentence by hand, it is clear that human performance on the task would not equal 100 BLEU points. It would therefore be interesting to determine the human performance on the string regeneration problem to provide a contrast and a point of comparison to the performance of machine systems.
Bibliography


Gerhard Reinelt. TSPLIB 95 documentation, 1995.


Yue Zhang, Graeme Blackwood, and Stephen Clark. Syntax-Based Word Ordering Incorporating a Large-Scale Language Model. In *Proceedings
Appendix A

Graph Transformations

In this chapter we describe two graph transformations for transforming the graphs modelling a variation of the TSP to a simpler variation. The first transformation transforms a problem instance of ATSP to a problem instance of TSP. The second transformation transforms a problem instance of GATSP to a problem instance of ATSP.

A.1 Asymmetric to Symmetric TSP Graph Transformation

The difference between ATSP and TSP is that the distance between two cities in the former is not necessarily the same in both directions. This extension enables representation of a wider set of problems. However, the ATSP is not as widely studied and fewer algorithms exist for solving it. Therefore, it is sometimes useful to transform an ATSP instance into a TSP instance and use the wide set of algorithms available to solve it. Any such transformation comes at a cost of increasing the number of vertices in the graph.

The ATSP is modelled as a directed weighted graph while the TSP is modelled as an undirected weighted graph. The goal of the ATSP to TSP trans-
formation is then to transform a directed weighted graph into an undirected weighted graph in such a way that the shortest tour and its cost in the resulting graph equal the shortest tour and its cost in the original graph (or the equivalence can be extracted in a systematic way).

Karp (1972) described a transformation that transforms an ATSP instance with \( n \) vertices into a TSP instance with \( 3n \) vertices. In this section we will describe and discuss the transformation introduced by Jonker and Volgenant (1983) that at most doubles the number of vertices in the graph.

Given a directed weighted graph \( G = (V, E) \) and its edge cost matrix \( C = [c_{ij}], i, j \in V \), we define the cost matrix \( \hat{C} = C \), except for \( \hat{c}_{ii} = -M, i \in V \), where \( M \) is a very large number. Let \( U = [u_{ij}], u_{ij} = \infty \) for \( i, j \in V \). The edge cost matrix of the transformed graph \( G' = (V', E') \) is then defined as:

\[
C' = \begin{bmatrix} U & \hat{C}^T \\ \hat{C} & U \end{bmatrix} \quad (A.1)
\]

The transformation doubles the number of vertices, \( |V'| = 2 \cdot |V| \). There is a one-to-one correspondence between the set of original ATSP and the transformed TSP solutions. We can obtain the original tour by deleting the vertices with indexes greater than \( |V| \) and its cost by adding \( n \cdot M \) to the cost of the transformed tour.

We demonstrate the ATSP to TSP transformation on an example graph shown in Figures A.1 and A.2. Figure A.1 shows the original directed weighted graph modelling an ATSP instance with 5 cities. The example graph is a complete digraph, meaning that every pair of vertices is connected by a pair of edges with different orientation. The solution tour is shown in bold along with the cost of edges in the tour.

Figure A.2 shows the transformed undirected weighted graph modelling a TSP instance with 10 cities. Every original vertex (marked with an upper case letter) has an additional auxiliary vertex associated with it (marked with
Figure A.1: An example directed weighted graph modelling the ATSP with 5 cities. The solution tour of the ATSP is shown in bold with associated edge costs.

Figure A.2: An example undirected weighted graph that was created as a result of the ATSP to TSP transformation of the graph shown in Figure A.1. Each original vertex has an auxiliary vertex associated with it. Together they model edge costs of two directed edges of the original graph using undirected edges. The solution tour that corresponds to the solution tour in the original graph is shown in bold.
a lower case letter). The cost of the undirected edge between them equals $-M$, where $M$ is a very large number. The graph is no longer complete as there are no non-infinite cost edges between any pair of vertices marked with the letters of the same case (for example, there is no edge between A and B).

The two original directed edges between two vertices are now represented as two undirected edges between two pairs of vertices. For example, the edge between vertex B and vertex c has the cost $c_{CB}$, which equals the cost of the edge with origin vertex C and destination vertex B in the original graph. The edge between vertex b and vertex C has the cost $c_{BC}$, which equals the cost of the edge with origin vertex B and destination vertex C in the original graph.

Due to the large negative cost of the edge between the original vertex and its auxiliary vertex the edges necessarily appear in the solution tour of the TSP. The solution tour that corresponds to the original solution tour is shown in bold. We can obtain the original tour by removing the auxiliary vertices from the tour, provided we choose the correct direction of the tour (auxiliary vertices model the outgoing edges of the original graph). The solution tour in Figure A.2 is A-a-E-e-D-d-B-b-C-c. By deleting the auxiliary vertices we obtain the tour A-E-D-B-C which is equal to the solution tour in the original graph. We can obtain the original tour cost by adding $-5 \cdot M$ to the cost of the transformed tour.

### A.2 Generalized ATSP to ATSP Graph Transformation

The Generalized Asymmetric Travelling Salesman Problem is represented by dividing the cities into districts and visiting one city in each district. With the GATSP we are able to model a wider set of problems than with ATSP, especially problems that are inherently hierarchical (Dimitrijević and Šarić, 1997). However, similarly to the ATSP, GATSP is not as widely studied and fewer algorithms exist for solving it. Because of this, we introduce a
transformation that transforms a GATSP instance into an ATSP instance. The ATSP instance can be further transformed into a TSP instance using the graph transformation described in the previous section. Any such transformation comes at a cost of increasing the number of vertices in the graph.

The GATSP is modelled as an $s$-partite directed weighted graph and its solution is the shortest $s$-cycle which includes exactly one vertex from each of the $s$ districts. The ATSP, on the other hand, is modelled as a directed weighted graph and its solution is the shortest tour of the graph. The goal of the GATSP to ATSP transformation is then to transform an $s$-partite directed weighted graph into a directed weighted graph in such a way that the shortest tour and its cost in the resulting graph equal the shortest $s$-cycle and its cost in the original graph or can be extracted in a systematic way.

In this section we will describe and discuss the transformation introduced by Dimitrijević and Šarić (1997) that doubles the number of vertices in the graph.

Given an $s$-partite directed weighted graph $G_{s,m} = (V, E)$ and its edge cost matrix $C = [c_{ij}], i, j \in V$, it is transformed to a directed weighted graph $G' = (V', E')$ in three steps:

1. For each original vertex $i \in V$ an auxiliary vertex $i'$ is introduced, $i, i' \in V'$. A directed edge connecting the original and the auxiliary vertex is added to the edge set, $(i, i') \in E'$, such that the cost of the edge is $c_{ii'} = M$, where $M$ is a large positive number, $M > \sum_{(i,j) \in E} c_{ij}$.

2. Inside each district, $m$ edges are introduced between original vertices so that they form a cycle, $(i, j) \in E', i, j \in V'_s$. A corresponding cycle with the opposite orientation is created between the auxiliary vertices, $(i', j') \in E', i', j' \in V'_s$. The cost of all $2m$ edges forming the two cycles equals $c_{i,j} = c_{i',j'} = 0$.

3. Every edge of the original edge set $E$, $(i, j) \in E$ is replaced by $(i', j) \in E'$. The auxiliary vertices therefore become the origin vertices and the original vertices become the destination vertices for all edges connecting vertices in different districts.
The transformation doubles the number of vertices, $|V'| = 2 \cdot |V|$. There is a one-to-one correspondence between the set of original GATSP and the transformed ATSP solutions. We can obtain the original tour by deleting all but the first vertex visited in each district. We can obtain the original solution cost by subtracting $s \cdot M$ from the cost of the transformed tour.

We demonstrate the GATSP to ATSP transformation on an example graph shown in Figures A.3, A.4, and A.5. Figure A.3 shows a graph modelling a GATSP instance with 3 districts each containing 2 cities ($s = 3$, $m = 2$, $n = 6$). The solution tour is shown in bold with associated edge costs.

The first step of the transformation adds an auxiliary vertex for each original vertex. The cost of the edge connecting the original and auxiliary vertex equals $M$. The first step is shown for one of the districts in the left part of Figure A.4. We mark the auxiliary vertices with lowercase letters.

The second step adds two cycles to each district, one connecting the original vertices and a corresponding cycle with opposite orientation connecting the
Figure A.4: The first and second steps of the GATSP to TSP transformation shown using a single district. The first step (right side) adds auxiliary vertices to every original vertex and connects them with an edge with cost $M$. The second step (left side) connects original vertices and auxiliary in two cycles with opposite orientation. All edges in the two cycles have cost 0.

auxiliary vertices. The cost of all edges in both cycles is zero. The second step is shown in the right part of Figure A.4. As there are only 2 original cities in the district, the difference in orientation between the cycles is not apparent.

The third step transforms the original edge set so that the auxiliary vertices become origin vertices and the original vertices become destination vertices. The final transformed ATSP graph is shown in Figure A.5. The solution the transformed ATSP instance is now a tour visiting all $2n$ vertices. We can obtain the original solution by deleting all but the first vertex visited in each district. By doing so, we obtain the original solution tour A1-B1-C1. We can obtain the original solution cost by subtracting $3 \cdot M$ from the transformed solution cost.
Figure A.5: An example directed weighted graph that was created as a result of the GATSP to ATSP transformation of the original graph shown in Figure A.3. The solution tour that corresponds to the solution tour in the original graph is shown in bold.
Appendix B

TSP Solvers

B.1 Concorde TSP Solver

In this section we briefly describe the Concorde TSP Solver. Our discussion draws from Applegate et al. (2006a).

The Concorde TSP Solver is a software package that implements a large number of algorithms developed over decades of research into the TSP with the goal of solving large and difficult TSP instances. This is in contrast to some other TSP solvers that might focus on computing smaller problem instances very fast.

The Concorde TSP Solver was introduced in early 1990s by Applegate, Bixby, Chvátal, and Cook. Since then it has become prominent for increasing the largest optimally solved TSP instance numerous times, the largest being the 85,900 city instance which was proved to be optimally solved in 2009 by Applegate et al. (2009).

The Concorde TSP Solver software is written in the C programming language and consists of roughly 130,000 lines of code. It builds on early approaches to solving the TSP, specifically the cutting plane method using linear programming first outlined by Dantzig et al. (1954). Due to the software’s complexity
and combination of various approaches we do not attempt to describe them in any detail. The authors offer a detailed description of the approaches used in the solver and a thorough computational study in Applegate et al. (2006a).

B.2 LKH

In this section we briefly describe the LKH heuristic and its origins. We start by describing the Lin-Kernighan heuristic to solving the TSP and continue by discussing its improvements, including the LKH algorithm. Our discussion draws from Applegate et al. (2006a).

The Lin-Kernighan heuristic, introduced by Lin and Kernighan (1973), is an algorithm for finding low cost solutions to the TSP. It represented a ground-breaking approach when it was first published and its improved implementations remain the most successful heuristic approaches to solving the TSP today.

The Lin-Kernighan heuristic is a tour improvement method. In its basic form, it takes a tour and attempts to change it in order to decrease the tour cost. The changes are performed in the following way. The algorithm takes a pair of edges and reconnects the four vertices so that the new pair of edges connect different pairs of vertices. This exchange is called a 2-opt move and was proposed by Flood (1956). However, instead of repeatedly performing 2-opt moves until no improving moves can be made as proposed by Flood (1956), Lin and Kernighan (1973) proposed to make several sequences of 2-opt moves that individually do not necessarily improve the tour but when applied in sequence result in an improved tour.

Over the years several improved methods of the Lin-Kernighan heuristic have been proposed. The Chained Lin-Kernighan algorithm, introduced by Martin et al. (1991), presents a substantial improvement over the original Lin-Kernighan algorithm. The original algorithm used the available computing time by repeating the computation from scratch as many times as allowed by the time constraint, starting from a different initial tour each time. The
Chained Lin-Kernighan algorithm instead takes the produced tour, perturbs it slightly, and continues the computation on the new tour. This approach resulted in significant improvements of the solutions.

LKH (Lin-Kernighan-Helsgaun), introduced by Helsgaun (2000), presents another milestone improvement on the original Lin-Kernighan heuristic after the Chained Lin-Kernighan algorithm. It differs from the original algorithm in a number of details. The most notable is the change of the search strategy for finding the best sequences of 5-opt moves. Compared to a 2-opt move, a 5-opt move considers reconnecting five edges at the same time instead of just two. The search strategy is efficient and considers only a small candidate set. Despite that, the algorithm finds very good solutions to the TSP, often finding the optimal solution even for large instances. The author reports that all instances up to a 13,509 city instance (the largest optimally solved TSP instance at the time) were solved optimally by the LKH heuristic algorithm.

More recently, LKH was able to produce an optimal solution to the 85,900 city instance, the largest optimally solved instance to date as mentioned in Section B.1. It currently holds the record for the best solution to many large TSP instances with unknown optima, including a 1,904,711 city World TSP instance, which includes all locations registered as populated cities or towns. Based on a lower bound computed by the Concorde TSP Solver, the LKH solution tour is at most 0.0474% greater than the length of the optimal tour (Cook, 2011).