OPTIMIZATION-DRIVEN EMERGENCE OF DEEP HIERARCHIES WITH APPLICATIONS IN DATA MINING AND EVOLUTION

A Dissertation
Presented to
The Academic Faculty

By

Payam Siyari

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in the
School of School of Computer Science

Georgia Institute of Technology

December 2018

OPTIMIZATION-DRIVEN EMERGENCE OF DEEP HIERARCHIES WITH APPLICATIONS IN DATA MINING AND EVOLUTION

Approved by:

Dr. Constantine Dovrolis, Advisor
School of Computer Science
Georgia Institute of Technology

Dr. Bistra Dilkina, Co-Advisor
School of Computational Science and Engineering
Georgia Institute of Technology
Department of Computer Science
University of Southern California

Dr. Thad Starner
School of Interactive Computing
Georgia Institute of Technology

Dr. Duen Horng (Polo) Chau
School of Computational Science and Engineering
Georgia Institute of Technology

Dr. Matthias Gallé
Senior Scientist
Naver Labs Europe

Date Approved: November 01, 2018

You are what you are in search of.

Rumi

To my family,

for their unconditional love and support.

ACKNOWLEDGEMENTS

I am indebted to many great people during the past 4 years at GeorgiaTech. I want to take the opportunity and acknowledge their valuable contributions here.

First and foremost, I want to express my gratitude to Professor Constantine Dovrolis as my advisor, and Professor Bistra Dilkina as my co-advisor for all their help, inspiration and encouragement during my doctoral studies journey. I was fortunate to enjoy their academic expertise as well as their warm-hearted support in my life.

I also want to deeply thank Dr. Matthias Gallé as a member of my PhD committee, and as my mentor during my internship at Xerox Research Center Europe. Matthias is a very smart and disciplined researcher and I am glad to know him as a colleague and as a friend.

Thanks to the other members of my committee, Dr. Thad Starner and Dr. Polo Chau, for their cooperation in reading the thesis and providing useful feedback.

During my graduate studies, I got to know many amazing people whose company made my PhD a much more enjoyable and exciting. Here, I try to remember as many of their names as possible:

At GeorgiaTech, I want to thank Faryad Darabi Sahneh, Ashkan Golgoon, Mostafa Faghih Shojaei, Amir Darabi, Amir Yazdanbakhsh, Amirhossein Afsharinejad, Amirhossein Salahshoor, Amirreza Shaban, Ardavan Afshar, Arezoo Shirazi, Hanif Hosseini, Kaeser M. Sabrin, Kamal Shadi, Mahdi Mahmoudzadeh, Maysam Nezafati, Mehrdad Farajtabar, Mojdeh Faraji, Mostafa Raissi, Neda Tavakoli, Omid Elliyoun, Pouya Asrar, Saman Yarmohammadi, Samaneh Ebrahimi, Shaghayegh Fathi, Shiva Bahrami and Yasaman Mohammad Shahi. In Atlanta, I want to thank Ala Fard, Ali Namayandeh, Navid Darvishzadeh, Neda Mohsenian and Shaghayegh Navabpour.

At Xerox Research Center Europe, I want to thank Vasu Sharma, Morteza Chehreghani, Miguel Collete, Saumya Jetley, Kunal Suri and Paco Nieto.

At Uber Advanced Technologies Group, I want to thank Andrew Duberstein, Collin

Otis, Skanda Shridhar, Steffon Davis, Alborz Alavian and Frits Bigham.

Finally, my deepest appreciation goes to my family, specially my beloved mother Parvin and my dearest father Davood. Despite the long distance, their love has always made me push forward in life. Thank you Peyman, my twin brother, for always being there. Negar, my dear sister, I can't say how much I'm thankful that I had you here in US during this journey. Masood, I owe you many things in my life, and I wish you the best and greatest in your life. Mohsen, I am lucky to have such an awesome brother like you and I always will be in debt for all you have done for me.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SUMMARY

The aim of this thesis is to develop methods for the analysis and modeling of hierarchical structures in real-world systems. A hierarchical architecture may be the result of the design or evolution of the system, and it may be chosen for meeting a criteria, such as better abstraction, easier maintenance, or cost/redundancy optimization. In this thesis, we aim to model hierarchical architectures that are shaped to optimize of the cost of the system. Examples of such systems exist both in nature and technology. For instance in software development, modular coding and reuse of previously built modules optimizes the redundancy and complexity of the system so that it is easier to debug and maintain it. Such cost optimization processes also take place in natural organisms, such as in energetic cost minimizations in the sensorimotor system.

We present an optimization framework over string data, called *Lexis*[1], that can be used to model hierarchical structures driven by optimization. We first use it to apply data mining tasks on a variety of string-represented datasets. We explore theoretical and algorithmic properties of this optimization problem using its close ties to the *Smallest Grammar Problem (SGP)*. Further, we build efficient heuristics, and show how we can leverage the Lexis framework to gain insights about the structure of the sequential data.

We also study a generalization of the SGP. The SGP – the problem of finding the smallest context-free grammar that generates exactly one given sequence – has never been successfully applied in grammatical inference. We provide efficient algorithms that approximate the SGP for the class of non-recursive grammars, instead of straight-line grammars (i.e., grammars with no recursion). Our empirical results show that we achieve smaller models than the current best approximations to the SGP on standard benchmarks, and that the inferred rules capture much better the syntactic structure of natural language. This research is a first and important step towards expanding smallest grammar applications in

[1] "Lexis" means "word" in Greek.

structure discovery in natural language.

Next, we propose a modeling framework, referred to as Evo-Lexis, that provides insight to some general and fundamental queries about evolving hierarchical systems. Evo-Lexis models the most elementary modules of the system as symbols ("sources") and the modules at the highest level of the hierarchy as sequences of those symbols ("targets"). Evo-Lexis computes the optimized adjustment of a given hierarchy when the set of targets changes over time by additions and removals (a process referred to as "incremental design"). In this research, we show that low-cost and deep hierarchies emerge when the population of target sequences evolves through tinkering and mutation. Strong selection on the cost of new candidate targets results in reuse of more complex (longer) nodes in an optimized hierarchy. The bias towards reuse of complex nodes results in an "hourglass architecture" (i.e., few intermediate nodes that cover almost all source-target paths). With such bias, the core nodes are conserved for relatively long time periods although still being vulnerable to major transitions and punctuated equilibria.

The predictions of the Evo-Lexis model should be tested using real data from evolving systems in which the outputs can be well represented by sequences. One such system is the iGEM synthetic DNA dataset [1]. Previous research has provided evidence on the hierarchical organization of this library of sequences. We investigate the time series of iGEM sequences, and whether the resulting iGEM hierarchies exhibit the qualitative properties predicted by the Evo-Lexis framework. Contrary to Evo-Lexis, in iGEM the amount of reuse decreases during the timeline of the dataset. Although this results in development of less cost-efficient and less deep Lexis-DAGs, the dataset exhibits a bias in reusing specific nodes more often than others. This results in the Lexis-DAGs to take the shape of an hourglass with relatively high H-score values and stable set of core nodes. Despite the reuse bias and stability of the core set, the dataset presents a high amount of diversity (as measured in this research) among the targets which is in line with modeling of Evo-Lexis.

CHAPTER 1
INTRODUCTION AND BACKGROUND

It is well known that many complex systems, both in technology and nature, exhibit modularity: independent modules, each of them providing a certain function, are combined together to perform more complex functions [2]. Additionally, modular systems are also organized in a hierarchical way: smaller modules are used within larger modules recursively [3]. Examples of such systems exist in a wide range of environments: in natural systems, it is believed that hierarchical modularity enhances evolvability (the ability of the system to adapt to new environments with minimal changes) and robustness (the ability to maintain the current status in the presence of internal or external variations) [4, 5]. In the technological world, hierarchically modular designs are preferred in terms of design and development cost, easier maintenance and agility (e.g., less effort in producing future versions of a software), and better abstraction of the system design [6].

The presented research in this thesis aims to position itself in developing methods for the analysis and modeling of hierarchical structures in real-world systems. Such hierarchical structure which is a result of the design or evolution of the system, might be chosen for meeting a variety of criteria, such as better abstraction, easier maintenance, or cost/redundancy optimization. In this thesis, we aim to model the hierarchical structures that are shaped for optimization of the cost of the system. Examples of such systems exist both in nature and technology: in designing a software, modular coding and reuse of previously built modules optimizes the redundancy and complexity of the system so that it is easier to debug and maintain it. Such cost optimization processes do take place in natural organisms too, such as in energetic cost minimizations in the sensorimotor system [7]. From a modeling and evolutionary perspective, it has been shown that hierarchy and modularity are two bi-products of the cost optimization in a design which allows reuse of

previously built modules [4, 8].

We present an optimization framework over string data, called *Lexis*[1], that is the foundation of our attempt for modeling hierarchical structures driven by optimization. The main data structure of this framework is a *Directed Acyclic Graph* (DAG) that shows how a set of top-level target strings can be constructed by reusing shorter and repeated substrings in lower levels. Such DAG can be used to model the network organization of a hierarchically modular system in which it is shown how higher level (and more complex) modules are reusing simpler and lower level modules. An example might be a function call-graph of an operating system. The following sections explain how we develop this framework and how we use it in various contexts ranging from mining hierarchical structures over sequential data to modeling the evolution of hierarchical systems.

1.1 Hierarchical structure discovery over sequential data

In both nature and technology, information is often represented in sequential form, as strings of characters from a given alphabet [9]. Such data often exhibit a hierarchical structure in which previously constructed strings are re-used in composing longer strings [10]. In some cases this hierarchy is formed "by design" in synthetic processes where there are some cost savings associated with the re-use of existing modules [11, 12]. In other cases, the hierarchy emerges naturally when there is an underlying evolutionary process that repeatedly creates more complex strings from simpler ones, conserving only those that are being re-used [10, 12]. For instance, language is hierarchically organized starting from phonemes to stems, words, compound words, phrases, and so on [13]. In the biological world, genetic information is also represented sequentially, and there is ample evidence that evolution has led to a hierarchical structure in which sequences of DNA bases are first translated into amino acids, then form motifs, regions, domains, and this process continues to create many thousands of distinct proteins [14].

[1] "Lexis" means "word" in Greek.

In the context of synthetic design, an important problem is to construct a minimum-cost Directed Acyclic Graph (DAG) that shows how to produce a given set of "target strings" from a given "alphabet" in a hierarchical manner, through the construction of intermediate substrings that are re-used in at least two higher-level strings. The cost of a DAG should be related somehow to the amount of "concatenation work" that the corresponding hierarchy would require. For instance, in *de novo* DNA synthesis [15, 16], biologists aim to construct target DNA sequences by concatenating previously synthesized DNAs in the most cost-efficient manner.

In other contexts, it may be that the target strings were previously constructed through an evolutionary process (not necessarily biological), or that the synthetic process that was followed to create the targets is unknown. Our main premise is that even in those cases it is still useful to construct a cost-minimizing DAG that composes the given set of targets hierarchically, through the creation of intermediate substrings. The resulting DAG shows the most parsimonious way to represent the given targets hierarchically, revealing substrings of different lengths that are highly re-used in the targets and identifying the dependencies between the re-used substrings. Even though it would not be possible to prove that the given targets were actually constructed through the inferred DAG, this optimized DAG can be thought of as a plausible hypothesis for the unknown process that created the given targets as long as we have reasons to believe that that process cares to minimize, even heuristically, the same cost function that the DAG optimization considers. Additionally, even if our goal is not to reverse engineer the process that generated the given targets, the derived DAG can have practical value in the applications such as compression or feature extraction.

Lexis produces an optimized hierarchical representation of a given set of *target* strings. The resulting hierarchy, *Lexis-DAG*, shows how to construct each target through the concatenation of intermediate substrings, minimizing the total number of such concatenations or DAG edges. Fig. 1.1 shows an example usage of Lexis framework over discretized time series data.

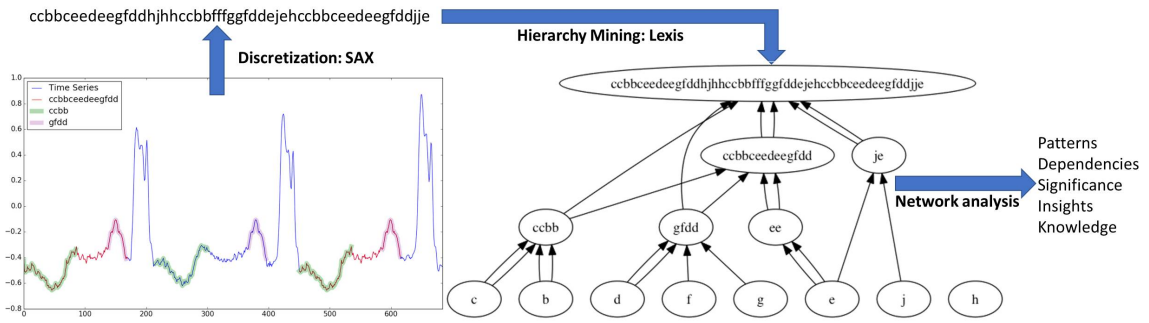


Figure 1.1: Illustration of the usage of Lexis framework on real data

SAX is a well-known discretization method for time series data. Nodes at the top and bottom of the directed-acyclic-graph represent targets and sources, respectively. Edges represent occurrence of the lower substrings in the upper ones. Note that direct edges from sources to target nodes are not drawn for clarity.

After we prove the NP-hardness of the Lexis optimization problem, we propose an efficient greedy algorithm for the construction of Lexis-DAGs. We also consider the problem of identifying the set of intermediate nodes (substrings) that collectively form the *core* of a Lexis-DAG, which is important in the analysis of Lexis-DAGs. We show that the Lexis framework can be applied in diverse applications such as optimized synthesis of DNA fragments in genomic libraries, hierarchical structure discovery in protein sequences, dictionary-based text compression, and feature extraction from a set of documents.

1.2 Extended search methods for inference of small hierarchical grammars

We explore the theoretical and algorithmic properties of the Lexis optimization problem using its close ties to the classic computer science problem, the *Smallest Grammar Problem* (SGP). SGP is the optimization problem of finding a smallest context-free grammar that generates exactly a given sequence. As such, it has some superficial resemblance to Grammatical Inference (i.e., finding the generative grammar over strings), both because of the choice of model to structure the data (a formal grammar) and because the goal of identifying structure is explicitly called-out as a potential application of SGP [17, 18, 19]. However, concrete applications so far of the SGP to grammatical inference are either re-

markably absent in the literature, or have been reported to utterly fail [20]. The main reason for this is that the definition of the SGP limits the inferred models to be straight-line programs (i.e., grammars with no recursion) which have no generalization capacity.

We investigate the reasons and propose an extended formulation that seeks to minimize non-recursive grammars, i.e., we remove the straight-line constraint over the inferred grammars and allow for search over branching grammars. The main idea accounting for the extension is the use of the repeated *contexts* as additional choices for the optimization algorithm. Given these choices, we provide very efficient algorithms that approximate the minimization problem of class of non-recursive grammars.

The proposed algorithm takes as input any straight-line grammar and infers additional generalization rules, optimizing a score function inspired both by the distributional hypothesis and regularizing through the Minimum Description Length (MDL) principle. Our empirical evaluation shows that we are able to find smaller models than the current best approximations to the Smallest Grammar Problem on standard benchmarks, and that the inferred rules capture much better the syntactic structure of natural language.

1.3 Modeling framework for evolution of optimized hierarchical systems

The emergence and/or design of systems that simultaneously are optimized according to a certain criteria, and evolve as their environment changes, has been a topic of significant interest across multiple research areas. Following the initial motivations in this chapter, we propose a modeling framework, referred to as Evo-Lexis, that provides insight to some general and fundamental queries about evolving hierarchical systems. Evo-Lexis is built over Lexis framework and models the system inputs as symbols ("sources") and the outputs as sequences of those symbols ("targets"). Evo-Lexis computes the optimized adjustment of a given hierarchy when the set of targets changes over time by additions and removals (a process referred to as "incremental design"). In general, a system interacts with its environment in a bidirectional manner: the environment imposes various constraints on the

system and the system also affects its environment. To capture this co-evolutionary setting in *Evo-Lexis*, we study how changes in the set of targets affect the resulting hierarchy but also how the current hierarchy affects the selection of new targets (i.e., whether a new candidate target is selected or not depends on its fitness or cost – and that depends on how easily that target can be supported by the given hierarchy). By incorporating well-known evolutionary mechanisms, such as tinkering (mutation), recombination, and selection, *Evo-Lexis* can capture such co-evolutionary dynamics between the generation of new targets and the hierarchy that supports them.

The main results of Evo-Lexis modeling can be summarized as follows:

- Low-cost and deep hierarchies emerge when the population of target sequences evolves through tinkering and mutation.
- Strong selection on the cost of new candidate targets results in reuse of more complex (longer) nodes in an optimized hierarchy.
- The bias towards reuse of complex nodes results in an "hourglass architecture" (i.e., few intermediate nodes that cover almost all source-target paths).
- With such bias, the core nodes are conserved for relatively long time periods although still being vulnerable to major transitions and punctuated equilibria.
- Finally, we analyze the differences in terms of cost and structure between incrementally designed hierarchies and the corresponding "clean-slate" hierarchies which result when the system is designed from scratch after a change.

1.4 Case-study of Evo-Lexis predictions on real-world data

The abstract modeling of Evo-Lexis is valuable because it can provide insights about the qualitative properties of the resulting hierarchies under different target generation models. Having said that however, we also believe that the predictions of the Evo-Lexis model

should be tested using real data from evolving systems in which the outputs can be well represented by sequences. One such system is the iGEM synthetic DNAs dataset [1]. The target DNA sequences in the iGEM dataset collectively form a library of synthetic DNA sequences and previous research has provided some evidence that these synthetic DNA sequences are designed by reusing existing components, and as such, this library has a hierarchical organization. We investigate the time series of iGEM sequences, and whether the resulting iGEM hierarchies exhibit the same qualitative properties we observed in this study through the abstract modeling of Evo-Lexis. Although the analysis shows that the iGEM Lexis-derived hierarchies are less cost-efficient and less deep than their Evo-Lexis counterparts (mostly due to an expanding set of source nodes), iGEM exhibits the Evo-Lexis bias to reuse certain nodes more often than others. This attribute results in hierarchies that exhibit the hourglass effect with stable core nodes, which is consistent with the predictions of the Evo-Lexis modeling framework.

CHAPTER 2

LEXIS: AN OPTIMIZATION FRAMEWORK FOR DISCOVERING THE HIERARCHICAL STRUCTURE OF SEQUENTIAL DATA

2.1 Introduction

Data represented as strings abound in biology, linguistics, document mining, web search, time series and many other fields. Such data often have a hierarchical structure, either because they were artificially designed and composed in a hierarchical manner or because there is an underlying evolutionary process that creates repeatedly more complex strings from simpler substrings.

In this chapter of the thesis, we propose an optimization framework, referred to as *Lexis*,[2] that designs a minimum-cost hierarchical representation of a given set of target strings. The resulting hierarchy, referred to as "Lexis-DAG", shows how to construct each target through the concatenation of intermediate substrings, which themselves might be the result of concatenation of other shorter substrings, all the way to a given alphabet of elementary symbols. We consider two cost functions: minimizing the total number of concatenations and minimizing the number of DAG edges. The choice of cost function is application-specific. The Lexis optimization problem is related to the smallest grammar problem [22, 23]. We show that Lexis is NP-hard for both cost functions, and propose an efficient greedy algorithm for the construction of Lexis-DAGs. Interestingly, the same algorithm can be used for both cost functions. We also consider the problem of identifying

The research in this chapter has resulted in the following publication:
[21] **P. Siyari**, B. Dilkina, C. Dovrolis, **"Lexis: An Optimization Framework for Discovering the Hierarchical Structure of Sequential Data"**, Proceedings of ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD'16), pp. 1185-1194, Aug. 2016.

[2] Lexis means "word" in Greek.

the set of intermediate nodes (substrings) that collectively form the "core" of a Lexis-DAG. This core is the minimal set of DAG nodes that can cover a given fraction of source-to-target paths, from alphabet symbols to target strings. The core of a Lexis-DAG represents the most central substrings in the corresponding hierarchy. We show that the Lexis framework can be applied in diverse applications such as optimized synthesis of DNA fragments in genomic libraries, hierarchical structure discovery in protein sequences, dictionary-based text compression, and feature extraction from a set of documents.

2.2 Problem Statement

2.2.1 Lexis-DAG

Given an alphabet S and a set of "target" strings T over the alphabet S, we need to construct a Lexis-DAG. A Lexis-DAG D is a directed acyclic graph $D(V, E)$, where V is the set of nodes and E the set of edges, that satisfies the following three constraints.[3]

First, each node $v \in V$ in a Lexis-DAG represents a string $\mathcal{S}(v)$ of characters from the alphabet S. The nodes V_S that represent characters of S are referred to as *sources*, and they have zero in-degree. The nodes V_T that represent target strings $T = \{t_1, t_2, \ldots, t_m\}$ are referred to as *targets*, and they have zero out-degree. V also includes a set of *intermediate nodes* V_M, which represent substrings that appear in the targets T. So, $V = V_S \cup V_M \cup V_T$.

Second, each node in $V_M \cup V_T$ of a Lexis-DAG represents a string that is the concatenation of two or more substrings, specified by the incoming edges from other nodes to that node. Specifically, an edge $e \in E$ from node u to node v is a triplet (u, v, i) such that the string $\mathcal{S}(u)$ appears as substring of $\mathcal{S}(v)$ at index i (the first character of a string has index 1). Note that there may be more than one edges from node u to node v. The number of incoming and outgoing edges for v is denoted by $d_{in}(v)$ and $d_{out}(v)$, respectively. $I(v)$ is the sequence of nodes u that appear in the incoming edges (u, v, i) of v, ordered by edge index i. We require that for each node v in $V_M \cup V_T$ replacing the sequence of nodes in

[3]To simplify the notation, even though D is a function of S and T, we do not denote it as such.

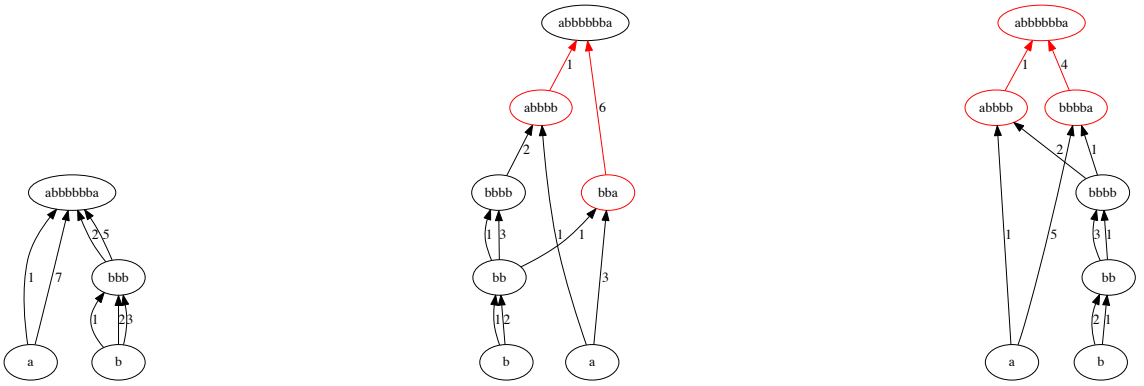


Figure 2.1: Example illustration of a Lexis-DAG
For targets $T = \{abbbbbba\}$ and sources $S = \{a, b\}$. Edge-labels indicate the occurrence indices: **(a)** A valid Lexis-DAG having both minimum number of concatenations and edges. **(b)** An invalid Lexis-DAG: two intermediate nodes are re-used only once. **(c)** An invalid Lexis-DAG: the top-layer string is not equal to the concatenation of its two in-neighbors (best viewed in color).

$I(v)$ with their corresponding strings results in exactly $\mathcal{S}(v)$.

Third, a Lexis-DAG should only include intermediate nodes that have an out-degree of at least two,

$$\forall v \in V_M, d_{out}(v) \geq 2. \tag{2.1}$$

In other words, every intermediate node $v \in V_M$ in a Lexis-DAG should be such that the string $\mathcal{S}(v)$ is re-used in at least two concatenation operations. Otherwise, $\mathcal{S}(v)$ is either not used in any concatenation operation, or it is used only once and so the outgoing edge from v can be replaced by re-wiring the incoming edges of v straight to the single occurrence of $\mathcal{S}(v)$. In both cases node v can be removed from the Lexis-DAG, resulting in a more parsimonious hierarchical representation of the targets. Fig. 2.1 illustrates the concepts introduced in this section.

2.2.2 The Lexis Optimization Problem

The *Lexis* optimization problem is to construct a minimum-cost Lexis-DAG for the given alphabet S and target strings T. In other words, the problem is to determine the set of intermediate nodes V_M and all required edges E so that the corresponding Lexis-DAG D

is optimal in terms of a given cost function $C(D)$.

$$\begin{aligned} &\min_{(E,V_M)} \ C(D) \\ &s.t.\ D = (V, E) \text{ is a Lexis-DAG for } S \text{ and } T \end{aligned} \tag{2.2}$$

The selection of an appropriate cost function is somewhat application-specific. A natural cost function to consider is the number of edges in the Lexis-DAG. In certain applications, such as DNA synthesis, the cost is usually measured in terms of the number of required concatenation operations. In the following, we consider both cost functions. Note that we choose to not explicitly minimize the number of intermediate nodes in V_M; minimizing the number of edges or concatenations, however, tends to also reduce the number of required intermediate nodes. Additionally, the constraint (2.1) means that the optimal Lexis-DAG will not have redundant intermediate nodes that can be easily removed without increasing the concatenation or edge cost. More general cost formulations, such as a variable edge cost or a weighted average of a node cost and an edge cost, are interesting but they are not pursued in this dissertation.

2.2.3 Edge cost

Suppose that the cost of each edge is one. The *edge cost* to construct a node $v \in V$ is defined as the number of incoming edges required to construct $\mathcal{S}(v)$ from its in-neighbors, which is equal to $d_{in}(v)$. The edge cost of source nodes is obviously zero. The edge cost $\mathcal{E}(D)$ of Lexis-DAG D is defined as the edge cost of all nodes, equal to the number of edges in D:

$$\mathcal{E}(D) = \sum_{v \in V} d_{in}(v) = |E| \tag{2.3}$$

2.2.4 Concatenation cost

Suppose that the cost of each concatenation operation is one. The *concatenation cost* to construct a node $v \in V_M \cup V_T$ is defined as the number of concatenations required to construct $\mathcal{S}(v)$ from its in-neighbors, which is equal to $d_{in}(v) - 1$. The concatenation cost $\mathcal{C}(D)$ of Lexis-DAG D is defined as the concatenation cost of all non-source nodes; it is easy to see that this is equal to the number of edges in D minus the number of non-source nodes,

$$\mathcal{C}(D) = \sum_{v \in V \setminus V_S} (d_{in}(v) - 1) = |E| - |V \setminus V_S| \tag{2.4}$$

Theorem 2.2.1 *The optimization problem in Eq.* (2.2) *is NP-hard for both the edge cost of Eq.* (2.3) *and the concatenation cost of Eq.* (2.4)*.*

We prove that the Lexis problem is NP-hard through a reduction from the *Smallest Grammar Problem* (SGP) [22].

Formally, *The Smallest Grammar Problem* for a string s is to identify a *Straight-Line Grammar* (SLG) G^* such that $L(G^*) = \{s\}$ and $|G^*| \leq |G|$ for any other G with $L(G) = \{s\}$, where $|G|$ denotes the size of grammar G. Charikar et al. [22] define the size of a grammar as the cumulative length of the right-hand side of all rules, i.e., $|G| = \sum_{T \to \alpha \in \Delta} |\alpha|$ where $|\alpha|$ is the number of symbols appearing in the term α of a grammar rule. Under this grammar size, Charikar et al. show that the Smallest Grammar Problem is NP-hard [22].

Proof of Theorem 1: Let us first start with edge cost. Consider an instance of SGP in which we are given string s and we are asked to compute an SLG G such that $L(G) = \{s\}$ and $|G| \leq m$, where $|G| = \sum_{T \to \alpha \in \Delta} |\alpha|$. We reduce it to an instance of the Lexis problem with a single target string $T = \{s\}$, in which we are asked to compute a Lexis-DAG D with $\mathcal{E}(D) \leq m$.

Given a grammar $G = (\Sigma, \Gamma, S, \Delta)$ as a solution to the SGP problem, we construct a solution D for the reduced Lexis problem. For each symbol in $\Sigma \cup \Gamma$, construct a node.

For a non-terminal $T \in \Gamma$, we refer to the corresponding node also as T, and associate that node with the string $\mathcal{S}(T)$ that is produced by expanding rule T according to grammar G. Also, for each rule $T \rightarrow \alpha$ in G, we scan α and add an edge in D from every node that corresponds to a terminal or nonterminal in α to the node that corresponds to T (along with the corresponding index). It is easy to see that D is acyclic since G is a straight-line grammar and that the number of edges in D is: $\mathcal{E}(D) = \sum_{T \rightarrow \alpha \in \Delta} |\alpha| \leq m$.

Conversely, consider a Lexis-DAG $D = (V_S \cup V_M \cup V_T, E)$ which is a solution to the Lexis problem from our reduction above , i.e., it has a single target string s, and $\mathcal{E}(D) \leq m$. We can construct a corresponding grammar G for the SGP as follows. For each source node in V_S, construct a terminal in Σ. For each node v in $V_T \cup V_M$, construct a nonterminal $NT(v)$ in Γ. For the single target node v in V_T, designate $NT(v)$ as the start symbol S. For each node v in $V_T \cup V_M$, add a rule in Δ with the right-hand side listing the corresponding $\Sigma \cup \Gamma$ symbols for all nodes in the sequence $I(v)$ (i.e., ordered as their respective strings appear concatenated in $\mathcal{S}(v)$). The constructed grammar is straight-line, since every nonterminal has one rule associated with it, and the grammar is also acyclic because the Lexis-DAG D is acyclic. It is easy to see that $|G| \leq m$.

The NP-hardness proof of the Smallest Grammar Problem with grammar size defined as $|G| = \sum_{T \rightarrow \alpha \in \Delta} |\alpha|$ [22] can be adapted for a modified grammar size definition, i.e., $|G| = \sum_{T \rightarrow \alpha \in \Delta} (|\alpha| - 1) = \left(\sum_{T \rightarrow \alpha \in \Delta} |\alpha|\right) - |\Delta|$. We can then use the same reduction from SGP to Lexis as in the case of edge cost to show that Lexis with concatenation cost is also NP-hard. ■

Note that the objective in Eq. (2.4) is an explicit function of the number of intermediate nodes in the Lexis-DAG. Hence the optimal solutions for the concatenation cost can be different than those for the edge cost. Fig. 2.2 illustrates an example that the Lexis-DAG, when optimizing for the edge cost function, may be different than the Lexis-DAG optimized for concatenation cost.

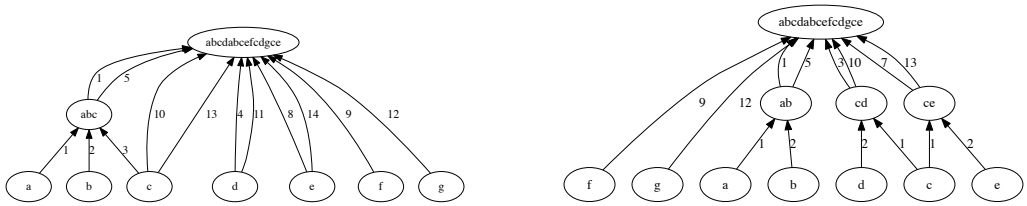


Figure 2.2: Difference in the inferred Lexis-DAGs with the edge-cost or concatenation-cost For target $T = \{abcdabcefcdgce\}$ and sources $S = \{a, b, c, d, e, f, g\}$ - **(a)** Lexis-DAG D_1 with $\mathcal{E}(D_1) = 13$ (optimal) and $\mathcal{C}(D_1) = 11$ (suboptimal). **(b)** Lexis-DAG D_2 with $\mathcal{E}(D_2) = 14$ (suboptimal) and $\mathcal{C}(D_2) = 10$ (optimal).

2.3 The Greedy Lexis algorithm

In this section, we describe a greedy algorithm, referred to as G-LEXIS, for both previous optimization problems. The basic idea in G-LEXIS is that it searches for the substring ξ that will lead, under certain assumptions, to the maximum cost reduction when added as a new intermediate node in the Lexis-DAG. The algorithm starts from the trivial Lexis-DAG with no intermediate nodes and edges from the source nodes representing alphabet symbols to each of their occurrences in the target strings. Similar ideas to this algorithm have been explored before in Nevill-Manning et al. [10] and Apostolico et al. [24].

Recall that for every node $v \in V_T \cup V_M$, $I(v)$ is the sequence of nodes appearing in the incoming edges of v, i.e., the sequence of nodes whose string concatenation results in the string $\mathcal{S}(v)$ represented by v. The sequences $I(v)$ can be interpreted as strings over the "alphabet" of Lexis-DAG nodes. Note that every symbol in a string $I(v)$ has a corresponding edge in the Lexis-DAG. We look for a repeated substring ξ in the strings $I_{T \cup M} = \{I(v) | v \in V_T \cup V_M\}$ that can be used to construct a new intermediate node. We can construct a new intermediate node for ξ, create incoming edges based on the symbols in ξ (remember ξ is a substring over the alphabet of nodes), and replace the incoming edges to each of the non-overlapping repeated occurrences of ξ with a single outgoing edge from the new node.

Consider the edge cost first. Suppose that ξ is repeated $R_{T \cup M, \xi}$ times in the strings

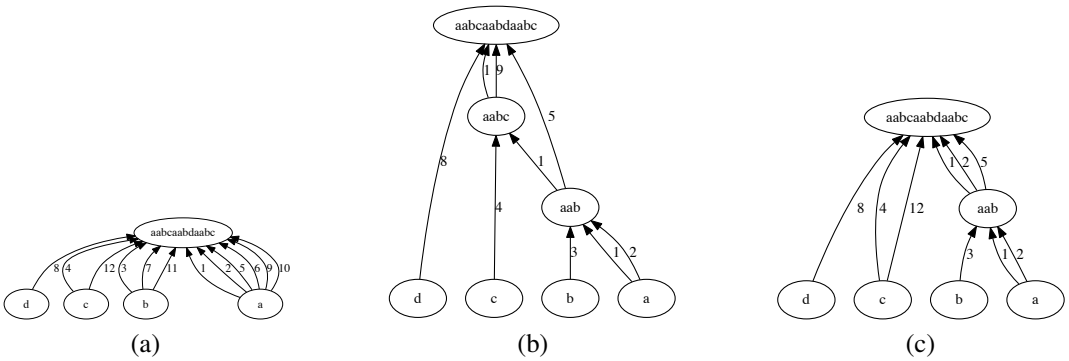


Figure 2.3: Illustration of G-LEXIS algorithm

Given target $T = \{aabcaabdaabc\}$ and sources $S = \{a, b, c, d\}$. - **(a)** Initial Lexis-DAG: The string passed to the suffix tree is $I(aabcaabdaabc) = aabcaabdaabc$. **(b)** Substring *aab* has maximum *SavedCost*. The target can be now written as $I(aabcaabdaabc) = \sigma_{aab}c\sigma_{aab}d\sigma_{aab}c$ where σ_{aab} is the substring *aab* of the new intermediate node. We also have that $I(aab) = aab$. The strings passed to the suffix tree are $\{I(aabcaabdaabc), I(aab)\}$. **(c)** Substring $\sigma_{aab}c$ has maximum *SavedCost* and is chosen for a new intermediate node. In this example, this iteration would be the last.

$I_{T\cup M}$. If these occurrences of ξ are non-overlapping, the number of required edges would be $|\xi|\, R_{T\cup M,\xi}$. After we construct a new intermediate node for ξ as outlined above, the edge cost will be $|\xi| + R_{T\cup M,\xi}$. So, the reduction in edge cost from re-using ξ would be $(R_{T\cup M,\xi} - 1)(|\xi| - 1) - 1$. Under the stated assumptions about ξ, this reduction is non-negative if ξ is repeated at least twice and its length is at least two.

Consider the concatenation cost now. If these occurrences of ξ are non-overlapping, the number of required concatenations for all the repeated occurrences would be $(|\xi| - 1)\, R_{T\cup M,\xi}$. After we construct a new intermediate node for ξ as outlined above, the concatenation cost will be $|\xi| - 1$. We expect a reduction in the number of required concatenations by $(R_{T\cup M,\xi} - 1)(|\xi| - 1)$.

So, the greedy choice for both cost functions is the same: select the substring ξ that maximizes the term $SavedCost = (R_{T\cup M,\xi} - 1)(|\xi| - 1)$. For this reason, our G-LEXIS algorithm can be used for both cost functions we consider. It starts with the trivial Lexis-DAG, and at each iteration it chooses a substring of $I_{T\cup M}$ in the Lexis-DAG that maximizes *SavedCost*, creates a new intermediate node for that substring and updates the edges of the

Lexis-DAG accordingly. The algorithm terminates when there are no more substrings of $I_{T\cup M}$ with length at least two and repeated at least twice. The pseudocode for G-LEXIS is shown in Algorithm 1. An example of application of the G-LEXIS algorithm is shown in Fig. 2.3.

ALGORITHM 1. G-LEXIS

Input: Alphabet S, Targets T **Output:** Lexis-DAG D

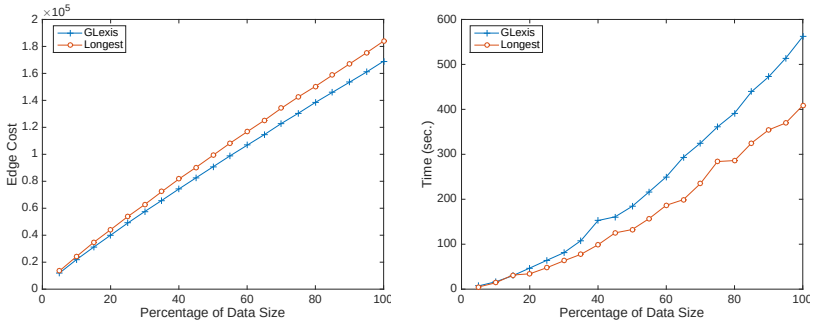
1. Initialize $V \leftarrow V_T \cup V_S$ and E, constructing each target in T from characters in S. $V_M \leftarrow \varnothing$.
2. Repeat:
   (a) $I_{T\cup M} \leftarrow \{I(v) | v \in (V_T \cup V_M)\}$;
   (b) Select ξ with maximum $(R_{T\cup M,\xi} - 1)(|\xi| - 1)$, where $R_{T\cup M,\xi}$ is the number of repeats of substring ξ in $I_{T\cup M}$; `#GreedyChoice`
   (c) If $(R_{T\cup M,\xi} - 1)(|\xi| - 1) = 0$, *break;*
   (d) $V \leftarrow V \cup \{\sigma_\xi\}$, where σ_ξ is the new intermediate node, and update E accordingly; `#UpdateLexis-DAG`

At each iteration of G-LEXIS, we need to find efficiently the substring of $I_{T\cup M}$ with maximum *SavedCost*. We observe that the substring that maximizes *SavedCost* is a "maximal repeat." Maximal repeats are substrings of length at least two, whose extension to the right or left would reduce its occurrences in the given set of strings. Suppose that it is not. Then, there is a substring $\hat{\sigma}$, which is not a maximal repeat, that maximizes *SavedCost*. If we can extend $\hat{\sigma}$ to the left or right we can increase its length without reducing its number of occurrences. By doing so, we construct a new substring with higher *SavedCost* than $\hat{\sigma}$, violating our initial assumption. So, the substring that maximizes *SavedCost* is a maximal repeat. A suffix tree over a set of input strings captures all right-maximal repeats, and right-maximal repeats are a superset of all maximal repeats [9]. To pick the one with maximum *SavedCost*, we need the count of non-overlapping occurrences of these substrings. A Minimal Augmented Suffix Tree [25] over $I_{T\cup M}$ can be constructed and used to count the number of non-overlapping occurrences of all right-maximal repeats in overall $O(L \log L)$ time, where L is the total length of target strings. Using a regular suffix tree

instead, this count can be achieved in only $O(L)$ time; but the suffix tree may count overlapping occurrences. In our implementation we prefer to use regular suffix tree, following related work [26] that has shown that this performance optimization has negligible impact on the solution's quality. So, the substring that is chosen for the new Lexis-DAG node is based on length and overlapping occurrence count. We then use the suffix tree to iterate over all occurrences of the selected substring, skipping overlapping occurrences. If a selected substring has less than two non-overlapping occurrences, we skip to the next best substring. Using the suffix tree, we can update the Lexis-DAG with the new intermediate node, and with the corresponding edges for all occurrences of that substring, in $O(L)$ time. The maximum number of iterations of G-LEXIS is $O(L)$ because each iteration reduces the number of edges (or concatenations), which at the start is $O(L)$. So, the overall run-time complexity using suffix tree is $O(L^2)$. Technically, if the alphabet size grows significantly, we should include the alphabet size in the complexity measure. However, Farach [27] proposes a linear algorithm for creating a suffix tree over integer alphabet (i.e., arbitrarily large alphabet) which makes the complexity claim for G-Lexis valid. Note that we do not use this algorithm in our implementations, hence we expect longer runtimes.

We have also experimented with other algorithms, such as a greedy heuristic that selects the longest repeat in each iteration of building the DAG, i.e., it chooses based on length among all substrings that appear at least twice in the targets or intermediate node strings. This heuristic can be efficiently implemented to run in only $O(L)$ time [28]. Our evaluation shows that G-LEXIS performs significantly better than the longest repeat heuristic in terms of solution quality, despite some running time overhead. Running both algorithms on a machine with an Intel Core-i7 2.9 GHz CPU and 16GB of RAM on the NSF abstracts dataset (introduced in Section 5) of $2,309$ target strings with total length $245,968$ symbols takes 562 sec for G-LEXIS and 408 sec for the longest repeat algorithm. The edge cost with G-LEXIS is 169,060 compared to 183,961 with the longest repeat algorithm.

We compare G-LEXIS with an algorithm that greedily replaces the longest repeated



(a) Cost Comparison (b) Runtime Comparison

Figure 2.4: Comparison of G-LEXIS with Longest Substring Replacement algorithm

substring, in terms of both runtime and cost. We implemented the latter, originally proposed in Inenaga et al. [28] using suffix trees, using our own efficient linked-suffix array. We used the NSF data described in the main text and ran the two algorithms on different fractions of the total dataset, repeating the experiments 10 times and recording the average runtime and edge cost. As seen in Fig. 2.4, the Longest Substring Replacement heuristic offers a better runtime, but its cost becomes increasingly worse than G-LEXIS as the size of the dataset grows. Also, G-LEXIS is still reasonably fast on all datasets we have analyzed so far.

2.4 Path-Centrality and the Core of a Lexis-DAG

After constructing a Lexis-DAG, an important question is to rank the constructed intermediate nodes in terms of significance or *centrality*. Even though there are many related metrics in the network analysis literature, such as closeness, betweenness or eigenvector centrality [29], none of them captures well the semantics of a Lexis-DAG. In a Lexis-DAG, a path that starts from a source and terminates at a target represents a dependency chain in which each node depends on all previous nodes in that path. So, the higher the number of such source-to-target paths traversing an intermediate node v is, the more important v is in terms of the number of dependency chains in which it participates. More formally, let $P_D(v)$ be the number of source-to-target paths that traverse node $v \in V_M$; we refer to

$P_D(v)$ as the *path centrality* of intermediate node v. The path centrality of sources and targets is zero by definition. First, note that:

$$P(v) = P_S(v)\, P_T(v) \tag{2.5}$$

where $P_S(v)$ is the number of paths from any source to v, and $P_T(v)$ is the number of paths from v to any target. This formulation suggests an efficient way to calculate the path centrality of all nodes in a Lexis-DAG in $O(|E|)$ time: perform two DFS traversals, one starting from sources and following the direction of edges, and another starting from targets and following the opposite direction. The first DFS traversal will recursively produce $P_S(v)$ while the second will produce $P_T(v)$, for all intermediate nodes.

Second, it is easy to see that $P_T(v)$ is equal to the number of times string $\mathcal{S}(v)$ is used for replacement in the target strings T. Similarly, $P_S(v)$ is equal to the number of times any source node is repeated in $\mathcal{S}(v)$, which is simply the length of $\mathcal{S}(v)$. Thus, the path centrality $P(v)$ of a node in a Lexis-DAG can be also interpreted as its "re-use count" (or number of replaced occurrences in the targets) times its length. Thus, an intermediate node will rank highly in terms of path centrality if it is both long and frequently re-used.

An important follow-up question is to identify the *core* of a Lexis-DAG, i.e., a set of intermediate nodes that represent, as a whole, the most important substrings in that Lexis-DAG. Intuitively, we expect that the core should include nodes of high path centrality, and that almost all source-to-target dependency chains of the Lexis-DAG should traverse at least one of these core nodes.

More formally, suppose K is a set of intermediate nodes and $\mathcal{P}^-(K)$ is the set of source-to-target paths after we remove the nodes in K from D. The core of D is defined as the minimum-cardinality set of intermediate nodes $\hat{K}$ such that the fraction of remaining

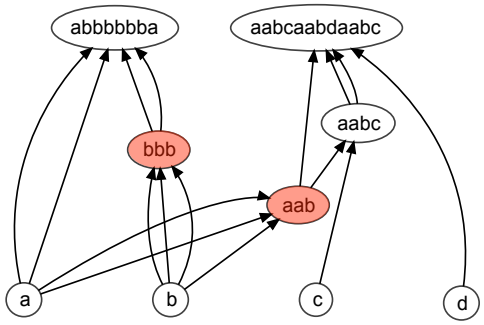


Figure 2.5: Illustration of the concepts regarding the core of a Lexis-DAG
For targets $T = \{abbbbbba, aabcaabdaabc\}$. Highlighted nodes are the core of the Lexis-DAG when $\tau = 0.75$. There are a total of 20 paths in the Lexis-DAG and by selecting the nodes aab and bbb, 15 paths (aab covers 9 paths and bbb covers 6 paths) will be covered.

source-to-target paths after the removal of $\hat{K}$ is at most τ:

$$\begin{aligned} \min_{K \subseteq V_M} \quad & |K| \\ s.t. \; & |\mathcal{P}^-(K)| \leq \tau \, |\mathcal{P}^-(\varnothing)| \end{aligned} \tag{2.6}$$

where $|\mathcal{P}^-(\varnothing)|$ is the number of source-to-target paths in the original Lexis-DAG, without removing any nodes.[4] Fig. 2.5 shows an example defining the concepts regarding the core of a Lexis-DAG.

Note that if $\tau = 0$ the core identification problem becomes equivalent to finding the min-vertex-cut of the given Lexis-DAG. In practice a Lexis-DAG often includes some *tendril-like* source-to-target paths traversing a small number of intermediate nodes that very few other paths traverse. These paths can cause a large increase in the size of the core. For this reason, we prefer to consider the case of a positive, but potentially small, value of the threshold τ.

The core identification problem is shown to be an NP-Hard problem in Sabrin [30] and we solve this problem with a greedy algorithm referred to as G-CORE. This algorithm adds in each iteration the node with the highest path centrality value to the core set, updates

[4]It is easy to see that $|\mathcal{P}^-(\varnothing)|$ is equal to the cumulative length of all target strings L.

the Lexis-DAG by removing that node and its edges, and recomputes the path centralities before the next iteration. The algorithm terminates when the desired fraction of source-to-target paths has been achieved.

G-CORE requires at most $O(|V|)$ iterations, and in each iteration we update the path centralities in $O(|E|)$ time. So the run-time complexity of G-CORE is $O(|V||E|)$.

One might wonder whether the order of removal of the nodes in G-CORE can be the same as the ones identified in G-LEXIS because of the similarities in the definition of path centrality and the gain measure utilized in G-LEXIS. The obvious difference is that we can consider removing sources in G-CORE while this selection is not possible in G-LEXIS. Our experiments also show that since path centrality of the nodes gets updated in G-CORE, the order of the nodes in these two algorithms are almost always different for the real datasets we have analyzed. An example of this difference is shown in the appendix.

2.5 Applications of Lexis

We now discuss a variety of applications of the proposed framework. Note that in all experiments, we use the library from [26] for extracting the maximal repeats and NetworkX [31] as the graph library in our implementation.

2.5.1 Optimized String Hierarchies

Lexis can be used as an optimization tool for the hierarchical synthesis of sequences. One such application comes from synthetic biology, where novel DNA sequences are created by concatenating existing DNA sequences in a hierarchical process [16]. The cost of DNA synthesis is considerable today due to the biochemical operations that are required to perform this "genetic merging" [16, 15]. Hence, it is desirable to re-use existing DNA sequences, and more generally, to perform this design process in an efficient hierarchical manner.

Biologists have created a library of synthetic DNA sequences, referred to as iGEM

[1]. Currently, there are 787 elementary "BioBrick parts" from which longer composite sequences can be created. Longer sequences are submitted to the Registry of Standard Biological Parts in the annual iGEM competition, then functionally evaluated and labeled. In the following, we analyze a subset of the iGEM dataset. In particular, this dataset contains $1,375$ composite DNA sequences that are labeled as iGEM *devices* because they have distinct biological functions; we treat these sequences as Lexis targets. The cumulative length of the target sequences is $6,957$ symbols. The 787 elementary BioBrick parts are treated as the Lexis sources. The iGEM dataset also includes other BioBrick parts that are neither devices nor elementary, and that have been used to construct more complex parts in iGEM; we ignore those because they do not have a distinct biological function (i.e., they should not be viewed as targets but as intermediate sequences that different teams of biologists have previously constructed).

We constructed an optimized Lexis-DAG for the given sets of iGEM sources and targets. To quantify the gain that results from using a hierarchical synthesis process, we compare the number of edges and concatenations in the Lexis-DAG versus a flat synthesis process in which each target is independently constructed from the required sources. The *Lexis* solution requires only 52% of the edges (or 56% of the concatenations) that the flat process would require. The sequence with the highest path centrality in the Lexis-DAG is *B0010-B0012*.[5] This part is registered as *B0015* in the iGEM library, and it is the most common "terminator" in iGEM devices. *Lexis* identified several more high centrality parts that are already in iGEM, such as *B0032-E0040-B0010-B0012*, registered as *E0240*. Interestingly, however, the Lexis-DAG also includes some high centrality parts that have not been registered in iGEM yet, such as *B0034-C0062-B0010-B0012-R0062*. A list of the top-15 nodes in terms of path centrality is given in Table 2.1.

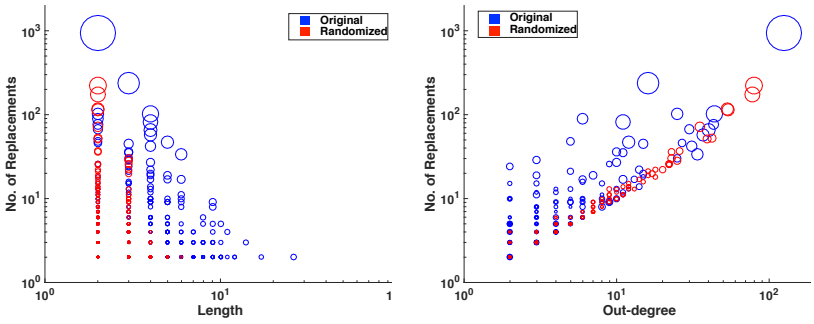
To explore the hierarchical nature of the iGEM sequences, we compared the *"Original"* Lexis-DAG, the one we constructed with the actual iGEM devices as targets, with *"Ran-*

[5] BioBricks start with *BBa_* prefix that are omitted here.

Table 2.1: Top-15 nodes with highest path-centrality in iGEM dataset's Lexis-DAG

Nodes shown in bold are already registered in the iGEM library.

B0010-B0012	**E0040-B0010-B0012**
B0034-E0040-B0010-B0012	**B0034-C0062-B0010-B0012**
B0032-E0040-B0010-B0012	B0034-C0062-B0010-B0012-R0062
B0034-E1010-B0010-B0012	**B0034-E1010**
B0034-I732006-B0034-E0040-B0010-B0012	**B0034-C0062**
B0030-E0040-B0010-B0012	**R0010-B0034**
K228000-B0010-B0012	**R0040-B0034**
C0051-B0010-B0012	



(a) (b)

Figure 2.6: Comparison of the Lexis-DAGs that result from the iGEM devices
The size of each point represents the path-centrality of that node.

domized" Lexis-DAGs. *"Randomized"* Lexis-DAG is the result of applying G-LEXIS to a target set where each iGEM device sequence is randomly reshuffled. We compare the Original Lexis-DAG characteristics to the average Lexis-DAG characteristics over ten randomized experiments. The Original Lexis-DAG has fewer intermediate nodes than the Randomized ones (169 in Original vs 359 in Randomized), and its depth is twice as large (8 vs 4.4). Importantly, the Randomized DAGs are significantly more costly: 44% higher cost in terms of edges and 52% in terms of concatenations.

To further understand these differences from the topological perspective, Fig. 2.6 shows scatter plots for the length, path centrality, and re-use (number of replacements) of each intermediate node in the Original Lexis-DAG vs one of the Randomized Lexis-DAGs. With randomized targets, the intermediate nodes are short (mostly 2-3 symbols), their re-use

is roughly equal to their out-degree, and their path centrality is determined by their out-degree; in other words, most intermediate nodes are directly connected to the targets that include them, and the most central nodes are those that have the highest number of such edges. On the contrary, with the original targets we find longer intermediate nodes (up to 11-12 symbols) and their number of replacements in the targets can be up to an order of magnitude higher than their out-degree. This situation happens when intermediate nodes with a large number of replacements are not only used directly to construct targets but they are repeatedly combined to construct longer intermediate nodes, creating a deeper hierarchy of re-use. In this case, the high path centrality nodes tend to be those that are both relatively long and common, achieving a good trade-off between specificity and generality.

2.5.2 Structure Discovery

As mentioned in the introduction, it is often the case that the hierarchical process that creates the observed sequences is unknown. *Lexis* can be used to discover underlying hierarchical structure as long as we have reasons to believe that that hierarchical process cares to minimize, even heuristically, the same cost function that Lexis considers (i.e., number of edges or concatenations). A related reason to apply Lexis in the analysis of sequential data is to identify the most parsimonious way, in terms of number of edges or concatenations, to represent the given sequences hierarchically. Even though this representation may not be related to the process that generated the given targets, it can expose if the given data have an inherent hierarchical structure.

As an illustration of this process, we apply Lexis on a set of protein sequences. Even though it is well-known that such sequences include conserved and repeated subsequences (such as motifs) of various lengths, it is not currently known whether these repeats form a hierarchical structure. That would be the case if one or more short conserved sequences are often combined to form longer conserved sequences, which can themselves be combined with others to form even longer sequences, etc. If we accept the premise that a conserved

sequence serves a distinct biological function, the discovery of hierarchical structure in protein sequences would suggest that elementary biological functions are combined in a Lego-like manner to construct the complexity and diversity of the proteome. In other words, the presence of hierarchical structure would suggest that proteins satisfy, at least to a certain extent, the *composability* principle, meaning that the function of each protein is composed of, and it can be understood through, the simpler functions of hierarchical components.

Our dataset is the proteome of baker's Yeast[6], which consists of 6,721 proteins. However, this dataset includes many protein homologues. It is important that we replace each cluster of homologues with a single protein; otherwise Lexis can detect repeated sequences within two or more homologues. To remedy this issue, we use the *UCLUST* sequence clustering tool [32], which is based on the *USEARCH* similarity measure (or identity search) [33]. The Percentage of Identity (PID) parameter controls how similar two sequences should be so that they are assigned to the same cluster. We set PID to 50%, which reduces the number of proteins to 6,033. Much higher PID values do not cluster together some obvious homologues, while lower PID values are too restrictive.[7] To reduce the running time associated with the randomization experiments described next, we randomly sample 1,500 proteins from the output of *UCLUST*.

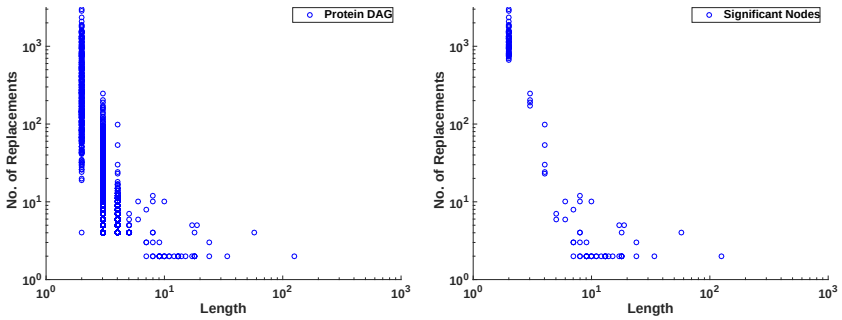
The total length of the protein targets is about 344K amino acids. The resulting Lexis-DAG has about 151K edges and 5,171 intermediate nodes, and its maximum depth is 7. Fig. 2.7(a) shows a scatter plot of the length and number of replacements of these intermediate Lexis nodes (repeated sequences discovered by Lexis).

Of course some of these sequences may not have any biological significance because their length and number of replacements may be so low that they are likely to occur just based on chance. For instance, a sequence of two amino acids that is repeated just twice in a sequence of thousands of amino acids is almost certain (the distribution of amino acids is not very skewed). To filter out the sequences that are *not* statistically significant, we rely on

[6] `http://www.uniprot.org/proteomes/UP000002311`
[7] `http://drive5.com/usearch/manual/uclust_algo.html`

the following hypothesis test. Consider a node that corresponds to a sequence with length l and number of replacements r in the given targets. The null-hypothesis is that sequences with these values of l and r will occur in a Lexis-DAG that is constructed for a random permutation of the given targets. To evaluate this hypothesis, we randomize the given target sequences multiple times, and construct a Lexis-DAG for each randomized sequence. We then estimate the probability that sequences of length l and number of replacements r occur in the randomized target Lexis-DAG, as the fraction of 500 experiments in which this condition is true. For a given significance level $\alpha = 0.1$, we can then identify the pairs (l, r) for which we can reject the null-hypothesis; these pairs correspond to the nodes that we view as statistically significant.[8]

On average, the randomized target Lexis-DAGs have a smaller depth (5.0) and more edges $(155K)$ than the original Lexis-DAG. Fig. 2.7(b) shows the intermediate nodes of the original Lexis-DAG that pass the previous hypothesis test.



(a) (b)

Figure 2.7: Length and number of replacements for the intermediate nodes in the Lexis-DAG of Yeast protein sequence data
The plot shows (a) before and (b) after we filter out the nodes that are not statistically significant.

Fig. 2.8 shows a small subgraph of the Lexis-DAG, showing only about 30 intermediate nodes; all these nodes have passed the previous significance test. The grey lines represent

[8]Another way to conduct this hypothesis test would be to estimate the probability that a specific sequence of length l will be repeated r times in a permutation of the targets. The number of randomization experiments would need to be much higher in that case, however, to cover all sequences that we see in the actual Lexis-DAG, each with a given value of r. Regarding testing and keeping the statistics of recurrent sequence motifs there has been research on efficient storage and retrieval of them [34, 35], however, we do not pursue them as we are interested in analysis of appearance of substrings in Lexis-DAGs.

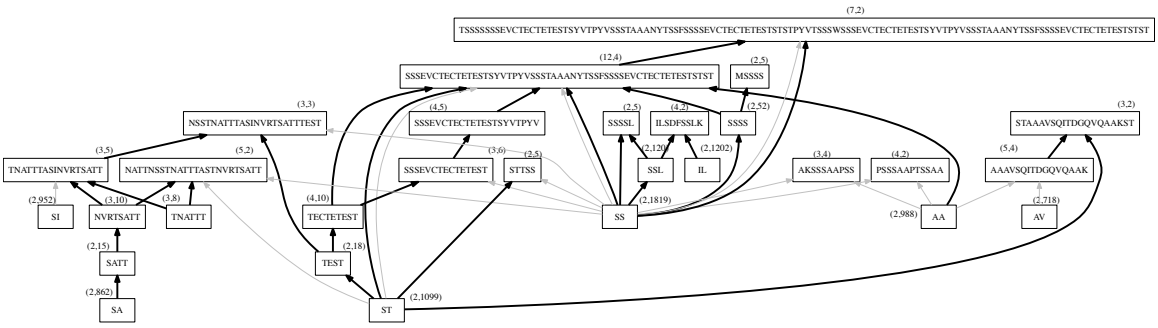


Figure 2.8: A small subgraph of the Lexis-DAG for 1,500 Yeast proteins
This plot only shows statistically significant nodes. Bold edges represent a direct connection between two nodes, while grey edges represent indirect connections (through nodes that are not shown in this plot). The label on top of each node shows the tuple (in-degree, number of replacements).

indirect connections, going through nodes that have not passed the significance test (not shown), while the bold lines represent direct connections. Interestingly, there seems to be a non-trivial hierarchical structure with several long paths, and with sequences of several amino acids that repeat several times even in this relatively small sample of proteins. Despite these preliminary results, it is clearly still early to draw any hard conclusions about the presence of hierarchical structure in protein sequences. We are planning to further pursue this question using Lexis in collaboration with domain experts.

2.5.3 Compression

Recent work has highlighted the connection between pattern mining and sequence compressibility [36]. Data compression looks for regularities that can be used to compress the data, while patterns are often useful as such regularities. In dictionary-based lossless compression, the original sequence is encoded with the help of a dictionary, which is a subset of the sequence's substrings as well as a series of pointers indicating the location(s) at which each dictionary element should be placed to fully reconstruct the original sequence. Following the Minimum Description Length Principle, one strives for a compression scheme that results in the smallest size for the joint representation of both the dictionary and the encoding of the data using that dictionary. The size of this joint representation is the total

space needed to store the dictionary entries plus the total space needed for the required pointers. We assume for simplicity that the space to store an individual character and a pointer are the same.

We now evaluate the use of a Lexis-DAG for compression (or compact representation) of strings. To do so, we need to decide 1) how to choose the patterns that will be used for compression, and 2) if a pattern appears more than once, which occurrences of that pattern to replace. A naive approach is to simply use the set of substrings that appear in the Lexis-DAG as dictionary entries, and compare them to sets of patterns found by other substring mining algorithms. Given a set of patterns as potential dictionary entries, selecting the best dictionary and pointer placement is NP-hard. A simple greedy compression scheme, that we refer to as *CompressLR*, is to iteratively add to the dictionary the substring that gives the highest compression gain when replacing all Left-to-Right non-overlapping occurrences of that substring with pointers. We re-evaluate the compression gain of candidate dictionary entries in each iteration. For a substring v with length $|v|$ and number of left-to-right non-overlapping occurrences R_v, the compression gain is:

$$R_v \times |v| - R_v - |v| = (R_v - 1) \times (|v| - 1) - 1 \tag{2.7}$$

We compare the substrings identified by Lexis with the substrings generated by a recent contiguous pattern mining algorithm called *ConSgen* [37] (we could only run it on the smallest available dataset). Additionally, we compare the *Lexis* substrings with the set of patterns containing all 2- and 3-grams of the data. The comparisons are performed on six sequence datasets: the "Yeast" and iGEM datasets of the previous sections, as well as four "NSF CS awards" datasets that will be described in more detail in the next section.

Table 2.2 shows the comparison results under the headings: Lexis-CompressLR, 2+3grams-CompressLR and ConSgen-CompressLR. These naive approaches are all on par with each other. This comparison, however, treats G-LEXIS as a mere pattern mining algorithm. In-

stead, the G-LEXIS algorithm constructs a Lexis-DAG that puts the generated patterns in a hierarchical context. One can think of the Lexis-DAG as the instructions in constructing a hierarchical "Lego-like" sequence. The edges into the targets tell us how to place the final pointers, i.e., which occurrences of a dictionary entry to replace in the targets. Further, the rest of the DAG shows how to compress the patterns that appear in the targets using smaller patterns. It is easy to see that using this strategy the compressed size becomes equal to the number of edges in the DAG. Using this strategy that is encoded in the Lexis-DAG results in an additional 2%-20% reduction in the compressed size over the CompressLR approaches.

Table 2.2: Comparison of compression ratio over NSF abstracts dataset

(i.e., percentage of compressed data size over original data size)

Dataset	Lexis DAG	Lexis CompressLR	2+3-gram CompressLR	ConSgen CompressLR
Know&Cog	**68.69**	76.58	77.13	—
Networks	**78.48**	86.47	86.43	—
Robotics	**73.19**	80.62	79.69	—
Theory	**79.41**	81.89	82.63	—
Yeast	**44.28**	51.08	50.71	—
iGEM	**47.86**	67.47	67.75	67.47

2.5.4 Feature Extraction

The Lexis-DAG can also be used to extract machine learning features for sequential data. The intermediate nodes that form the core of a Lexis-DAG, in particular, correspond to sequences that are both relatively long and frequently re-used in the targets. We hypothesize that such sequences will be good features for machine learning tasks such as classification or clustering because they can discriminate different classes of objects (due to their longer length) and at the same time they are general within the same class of objects (due to their frequent occurrence in the targets of that class).

To test this hypothesis, we used Lexis to extract text features for four classes of NSF re-

search award abstracts during the 1990–2003 time period.[9] We pre-processed each award's abstract through Stopword removal and Porter stemming so that the noise regarding occurrences of the long but noncontagious repeats, or the different variations of a single root, is reduced and they can be detected by G-LEXIS. The alphabet S is the set of word stems that appear at least once in any of these abstracts. Table 2.3 describes this dataset in terms of number of abstracts, cumulative abstract length, and average length per abstract for each class.

Table 2.3: Description of 4 classes of NSF award abstracts

Class	$\|T\|$	L	$L/\|T\|$	$\|V_M\|$
Knowledge &Cog Sci	411	47,858	116	2,902
Networks	836	74,738	89	3,730
Robotics	496	56,481	113	4,560
Theory	566	66,891	118	4,247

We constructed the Lexis-DAG for each class of abstracts, and then used the G-CORE algorithm to identify the core for each DAG. We stopped G-CORE at the point where 95% of indirect paths in the Lexis-DAG are covered. The strings in each core are the extracted features for the corresponding class of abstracts. Table 2.4 shows the 5 strings extracted by G-CORE for each class. We create a common set of G-CORE features by taking the union of the sets of core substrings derived for each class. The next step is to construct the feature vector for each abstract. We do so by representing each abstract as a vector of counts, with a count for each substring feature.

To assess how good these features are, we compare the classification accuracy obtained using the Lexis features with more mainstream representations in text mining on NSF data: "bag-of-words", 2-gram, 3-gram, and two combinations of these representations. We use a basic SVM classifier with an RBF kernel. We used the SVM implementation in MATLAB.

[9] `archive.ics.uci.edu/ml/machine-learning-databases/nsfabs-mld/nsfawards.data.html`

Table 2.4: Word stems in the Lexis-DAG core of each class of NSF abstracts dataset.

Knowledge & Cog	**Networks**
machine learn	request support nsf connect
knowledg base	bit per second
natur languag	two year
artifici intellig	provide partial support
neural network	high perform
Robotics	**Theory**
comput vision	comput scienc
first year	real world
robot system	complex class
object recognit	complex theori
real time	approxim algorithm

Table 2.5: SVM classification accuracy over NSF abstracts dataset

nonzeros is the number of nonzero elements in the term-document matrix with each feature set. The accuracies and the parameters are fit based on 10-fold cross-validation for each feature set.

Method (# Features)	**# Nonzeros**	(γ, c)	**Accuracy**
Bag-of-words (9,7k)	190,2k	*(0.0015,3)*	$88.3\% \pm 2.7\%$
G-CORE (14,4k)	**55,5k**	*(0.02,1)*	$90.0\% \pm 2.3\%$
2-Gram (124,9k)	228,0k	*(0.0015,3)*	$91.3\% \pm 2.0\%$
3-Gram (186,3k)	234,5k	*(0.001,1)*	$75.8\% \pm 5.6\%$
1+2-Gram (134,6k)	418,2k	*(0.001,1)*	$90.9\% \pm 2.1\%$
1+2+3-Gram (321,0k)	652,8k	*(0.001,1)*	$89.2\% \pm 2.5\%$

The accuracy results are similar to those with a KNN classifier that we tried with a Cosine distance, and the accuracy is evaluated with 10-fold cross-validation.

Table 2.5 shows that the Lexis features result in a much sparser term-document matrix, and so in smaller data overhead in learning tasks, without sacrificing classification accuracy. Lexis also results in a lower feature dimensionality (with the exception of the 1-gram method but the accuracy of that method is much lower). Note that most *Lexis* features are 2-grams but the *Lexis* core (for 95% of path coverage) may also include longer n-grams. Lexis becomes better, relative to the other feature sets, as we decrease the number of considered features. For instance, with $3,000$ features the accuracy with Lexis is 74%, while the accuracy with the 1-gram and 2-gram features is 69% and 64%, respectively.

2.6 Related Work

Lexis is closely related to the *Smallest Grammar Problem* (SGP), which focuses on the following question: *What is the smallest context-free grammar that only generates a given string?* The constraint that the grammar should generate only one string is important because otherwise we could simply consider a Σ^* as the generator of any string over Σ. The SGP is NP-hard, and inapproximable beyond a ratio of $\frac{8569}{8568}$ [22]. Algorithms for SGP have been used for string compression [38] and structure discovery [10] (for a survey see [26]). There are major differences between *Lexis* and SGP. First, in *Lexis* we are given a set of several target strings, not only one string. Second, *Lexis* infers a network representation (a DAG) instead of a grammar, and so it is a natural tool for questions relating to the hierarchical structure of the targets. For instance, the centrality analysis of intermediate nodes or the core identification question are well understood problems in network analysis, while it is not obvious how to approach them with a grammar-based approach.

One can also relate *Lexis* to the body of work on *sequence pattern mining*, where one is interested in discovering frequent or interesting patterns in a given sequence. Most work in this area has focused on mining subsequences, i.e., a set of ordered but not necessarily adjacent characters from the given sequence. In Lexis, we focus on identifying substrings, also known as *contiguous sequence patterns*. A couple of recent papers develop algorithms for mining substring patterns [37, 39], since sequence mining algorithms do not readily apply to the contiguous case. However, they rely on the methodology of candidate generation (commonly used in sequence pattern mining), where all patterns meeting a certain criterion are found, such as having frequency of at least two or being maximal. In the sequence mining literature, it has been recently observed that the size of the discovered set of patterns as well as their redundancy can be better controlled by mining for a set of patterns that meet a criterion collectively, as opposed to individually. This approach is useful when these patterns are used as features in other tasks such as summarization or classifi-

cation. Algorithms for such set-based pattern discovery have been recently developed for sequence pattern mining [36, 40]. Gallé [41] explores similar ideas of using key n-grams for classification of textual documents. In particular, it shows how equivalence class of largest-maximal repeats can obtain similar or better results than maximal repeats in choosing n-grams for text classification tasks. In the context of substring pattern mining, Paskov et al. [42] show how to identify a set of patterns with optimal lossless compression cost in an unsupervised setting, to be then used as features in supervised learning for classification. In a follow-up paper [43], DRACULA provides a "deep variant" of Paskov et al. [42] that is similar to Lexis, in terms of the general problem setup. DRACULA's focus is mostly on complexity and learning aspects of the problem, while Lexis focuses on network analysis of the resulting optimized hierarchy. For instance, DRACULA considers how to take into account how dictionary strings are constructed to regularize learning problems, and how the optimal Dracula solution behaves as the cost varies. We have shown that, although not specifically designed for feature extraction or compression, the *Lexis* framework also results in a small and non-redundant set of substring patterns that can be used in classification and compression tasks.

Optimal DNA synthesis is a new application domain, and we are only aware of the work by Blakes et. al. [15]; they describe DNALD, an algorithm that greedily attempts to maximize DNA re-use for multistage assembly of DNA libraries with shared intermediates. Even though the *Lexis* framework was not specifically designed for DNA synthesis, the Lexis-DAGs can be seamlessly used as solutions for this task. In our illustrative example with the iGEM dataset, G-Lexis returns solutions with 11% lower synthesis cost (equivalent to concatenation cost) than DNALD.

Structure discovery in strings has been explored from several different perspectives. For example, the grammar-based algorithm SEQUITIR [10] presents interesting possible applications in natural language and musical structure identification. In an information-theoretic context, Lancot et al. [44] shows how to distinguish between coding and non-

coding regions by analyzing the hierarchical structure of genomic sequences.

2.7 Conclusion

Lexis is a novel optimization-based framework for exploring the hierarchical nature of sequence data. In this chapter of the thesis, we stated the corresponding optimization problems in the admittedly limited context of two simple cost functions (number of edges and concatenations), proved their NP-hardness, and proposed a greedy algorithm for the construction of Lexis-DAGs. This research can be extended in the direction of more general cost formulations and more efficient algorithms. Additionally, we have worked on an incremental version of Lexis in which new targets are added to an existing Lexis-DAG, without re-designing the hierarchy which will be covered in detail later in this thesis.

We also applied network analysis in Lexis-DAGs, proposing a new centrality metric that can be used to identify the most important intermediate nodes, corresponding to substrings that are both relatively long and frequently occurring in the target sequences. This network analysis connection raises an interesting question: are there certain topological properties that are common across Lexis-DAGs that have resulted from long-running evolutionary processes? We have some evidence that one such topological property is that these DAGs exhibit the *hourglass effect* [45].

Finally, we gave four examples of how Lexis can be used in practice, applied in optimized hierarchical synthesis, structure discovery, compression and feature extraction. In future work, we plan to apply Lexis in various domain-specific problems. In biology, in particular, we can use Lexis in comparing the hierarchical structure of protein sequences between healthy and cancerous tissues. Another related application is *generalized phylogeny inference*, considering that horizontal gene transfers (which are common in viruses and bacteria) result in DAG-like phylogenies (as opposed to trees).

CHAPTER 3

THE GENERALIZED SMALLEST GRAMMAR PROBLEM

3.1 Introduction

Chapter 2 pointed to close connections between the Lexis optimization problem and the Smallest Grammar Problem. This problem was formally introduced and analyzed in Charikar et al. [18], and provides a theoretical framework for the popular Sequitur algorithm [17], as well as the work around grammar-based compression [47]. The Smallest Grammar Problem – the problem of finding the smallest context-free grammar that generates exactly one given sequence – has never been successfully applied to grammatical inference. The main reason for this is that the definition of the SGP constrains the inferred models to be straight-line grammars which have no generalization capacity.

We present here an alternative definition which removes this constraint and design an efficient algorithm that achieves at the same time:

- smaller grammars than the state-of-the-art, measured on standard benchmarks.
- generalized rules which correspond to the true underlying syntactic structure, measured on the Penntree bank.

We achieve this by extending the formalism from simple straight-line grammars to non-recursive grammars. Our algorithm takes as input any straight-line grammar and infers additional generalization rules, optimizing a score function inspired both by the distributional

The research in this chapter has resulted in the following publication:
[46] **P. Siyari**, M. Gallé, **"The Generalized Smallest Grammar Problem"**, Proceedings of International Conference on Grammatical Inference (ICGI'16), PMLR 57, pp. 79-92, 2017.

hypothesis [48] and regularizing through the MDL principle. It is very efficient in practice and can easily be run on sequences of millions of symbols.

3.2 Related Work

We take inspiration from three related areas: the work around the Smallest Grammar Problem, the use of Minimum Description Length principles in grammatical inference and concrete implementations of Harris substitutability theory [48].

The Smallest Grammar Problem is defined as the combinatorial problem of finding a smallest context-free grammar that generates exactly the given input sequence. By reducing the expression power from Turing machines to context-free grammars it becomes a computable (although intractable) version of Kolmogorov Complexity. This relationship is also reflected in the interest of studying that problem: by finding smaller grammars, the hope is not only to find better compression techniques but also to model better the redundancies and therefore the structure of the sequence. The current state-of-the-art algorithms that obtain the smallest grammars on standard benchmarks are based on a search space introduced in Carrascosa et al. [49] which decouples the search of the optimal set of substrings to be used, and the choice of how to combine these in an optimal parsing of the input sequence. That parsing can be solved optimally in an efficient way, and the algorithms diverge in how they navigate the space of possible substrings to be used, trading off efficiency with a broader search. These algorithms include IRRMGP [50] – a greedy algorithm –, ZZ [51] – a hill-climbing approach –, and MMAS-GA [52] – a genetic algorithm –. Applications to grammar learning has been studied in Chapter 3 of Eyraud [20], which concludes that the failure to retrieve meaningful structure is due to the fact that `Sequitur` "est un algorithme dont le but est de compresser un texte à' l'aide d'une grammaire [. . .] Or la fréquence d'apparition d'un motif n'est pas une mesure permettant de savoir si ce motif est un constituant"[2].

[2] "is an algorithm whose goal is to compress a text through a grammar [. . .] but the frequency of occurrences of a motif is not a signal to decide whether a motif is a constituent"

Algorithms to find smaller grammars focus on *intrinsic* properties of substrings (e.g., occurrences, length). However, in grammatical inference a primary focus is the *context* in which a substring occurs in, a principle that traces back to Harris **substitutability theory** [48]. Different class of substituable languages have been defined, which all start from the intuition that if two words occur in the same context they should belong to the same semantic class (be generated by the same non-terminal). Occurrences of strings uwv and $uw'v$ are therefore a signal that the words w and w' are substituable one by the other (as in "The car is fast" and "The bike is fast", with $w =$ car and $w' =$ bike). In recent years, very good learnability results have been obtained with different variations of substituable languages [53, 54, 55, 56], and those insights have long been the basis of unsupervised learning algorithms applied to natural language text [57, 58, 59].

The **Minimum Description Length (MDL)** principle is a popular approach for model selection, and states that the best model to describe some data is the one that minimizes jointly the size of the model and the cost of describing the data given the model. This principle has been applied often in tools targeted to discover meaningful substructure, including grammatical inference [60, 61]. In this context, the grammars obtained through the SGP are learning the data by heart as they do not perform any generalization. Instead of this, we propose to extend this model and to control its generalization capacity through MDL.

3.3 Model

By focusing on grammars that generate a single sequence only, the SGP limits itself to straight-line grammars, which are context-free grammars who do not branch nor loop. We propose here to relax the first of these constraints, allowing branching non-terminals. Such "non-recursive grammars" have found use in natural-language processing, where several applications have this characteristic [62], despite the fact that they are only able to generate finite languages.

Definition 3.3.1 *A non-recursive context free grammar is a tuple $G = \langle \Sigma, \mathcal{N}, S, \mathcal{P} \rangle$, with terminals Σ, non-terminals $\mathcal{N}$ starting symbol S and context-free production rules $\mathcal{P}$ such that for any $A, B \in \mathcal{N}$ not necessarily distinct, if B occurs in a derivation of A, then A does not occur in a derivation of B.*

Different from straight-line grammars, the language generated by non-recursive CFG can be larger than a single string, although it is always finite. In the spirit of the smallest grammar problem, we are still interested in encoding exactly one given string and have therefore to specify which of all the strings in the language is the encoded one.

The size of such a grammar with respect to a specific sequence s – which we will then try to minimize – will therefore be the sum of two factors: the size of the general grammar, and the cost of specifying s:

Definition 3.3.2 *Given a non-recursive context-free grammar $G = \langle \Sigma, \mathcal{N}, S, \mathcal{P} \rangle$, the size of G wrt s is defined as:*

$$|G|_s = \sum_{N \to \alpha \in \mathcal{P}} (|\alpha| + 1) + \textit{cost}(s|G)$$

where *cost(s—G)* is the cost of expressing s given G, and should be expressed in the same unit that the grammar itself, namely symbols.

We are now ready to define our generalized version of the smallest grammar problem:

Definition 3.3.3 *Given a sequence s, a smallest non-recursive grammar is a non-recursive context-free grammar G such that $s \in L(G)$ and $|G|_s$ is minimal.*

Note that we do not put any restrictions on the language that the grammar could generate. A more general grammar (one that generates a larger language) may have less or shorter production rules, but the decoding may be more expensive. This trade-off is a standard in any MDL formulation. The extreme case, where $|L(G)| = 1$ and $\text{cost}(s|G) = 0$,

reduces to the traditional smallest grammar problem. The goal of generalizing the definition is that by adding ambiguities, the grammar would end up having smaller size even with some amount of cost being added for resolving the ambiguities.

Before we define $cost(s|G)$, we need to introduce the mentioned type of ambiguity that when introduced, is potential to lead us to smaller grammars. From a compression perspective, we are interested in capturing more flexible patterns than just exact repeats. In that way, we are looking for non-terminals that generate words v and v', where both words are different although *similar*, so that disambiguating between them is cheap. Several such similarities have been defined in the domain of inexact pattern matching [63], and a common practice is to start with *seeds*, which are exact repeats and then try to extend them to enlarge their support (number of occurrences) while minimizing the added differences. Such an idea has been used for DNA compression [19, 64]. The particular kind of inexact motif we focus on is based on insights from theoretical and experimental results in grammatical inference. We assume that v and v' share a common prefix and suffix, and that all the changes are contained in the middle. That is, $v = pws$ and $v' = pw's$, with $w \neq w'$. This corresponds to typical distributional approaches which look for words v, v' that occur in the same context, and has been applied as such similarly in [57].

In Section 2.2 of Van Zaanen [65] argues that replacing unequal parts leads to smaller grammars than replacing equal parts. However, the analysis there does not take into account that replacing unequal parts adds ambiguity, which – from a lossless compression perspective – has to be disambiguated in order to retrieve the correct sequence. It is surprisingly hard to define an encoding in such a way that replacing such motifs results in grammars that are smaller than those that can be obtained by replacing exact repeats only, as has been reported previously [66].

3.4 Algorithm

We propose to extend the greedy algorithm for inferring small straight-line grammars [24, 67] to take into account such inexact motifs. That algorithm (`Greedy`) chooses in each iteration the repeat that reduces the current grammar the most. For an exact repeat u, the gain $f(u, G)$ is the the reduction in size of replacing all occurrences[3] of u in G by a new non-terminal N and adding a rule $N \rightarrow u$. By encoding the grammar in a single string, separating rules by special symbols[4], it can easily be shown that $f(u, G) = (|u| - 1)\ (occ_G(u) - 1) - 2$, where $occ_G(u)$ is the number of non-overlapping occurrences of u in G. Deducing such a formula for branching non-terminals is a bit more complicated, and depend strongly on the specific encoding used.

3.4.1 Encoding the Grammars

In order to provide a fair comparison with the straight-line grammars, we will model carefully the way the grammar is encoded. It should be done in such a way that the target sequence can be retrieved unambiguously from that encoding alone. This choice will then guide the optimization procedure to minimize it, and as we will see, it influences heavily the resulting grammar.

We first chose to encode all the possible branchings sequentially, separated by a special separator symbol (|, where $| \notin (\Sigma \cup \mathcal{N})$). Before encoding the final grammar, the non-terminal are sorted by their depth in the parsing trees. For this, we define $depth_G(N)$ for $N \in \mathcal{N}$ as the maximal depth over all parse trees of all sequences $s \in L(G)$, and have the following:

Proposition 3.4.1 *If G is a non-recursive grammar, then $depth_G(N)$ is well-defined.*

which comes directly from the absence of recursion in these grammars.

[3] Actually, a maximal set of non-overlapping occurrences.

[4] So that the size of a grammar is just $\sum_{N \rightarrow \alpha \in \mathcal{P}} |\alpha| + 1$

In the most general setting, we are interested in motifs of the form $u.^*v$, which matches any substring of the form uwv with $w \in (\Sigma + \mathcal{N})^*$. Searching for such general motifs is computationally expensive but feasible, by considering all pairs of repeats.

As an example, consider the following sequence:

s =

Alice was beginning to get very tired .

Alice was getting very tired .

Alice is very tired .

Alice will be very tired .

Alice was getting very tired .

where we assume the alphabet to be the set of English words. Consider now the following grammar G_1 generating s:

$$\begin{aligned} S &\rightarrow N.N.N.N.N. \\ N &\rightarrow \text{Alice } V \text{ very tired} \\ V &\rightarrow \text{was beginning to get} \mid \text{was getting} \mid \text{is} \mid \text{will be} \mid \text{was getting} \end{aligned} \tag{3.1}$$

which would be encoded as "$N_1.N_1.N_1.N_1.N_1$ # Alice N_2 is very tired # was beginning to get | was getting | is | will be | was getting #". Note that the expansion "was getting" is repeated twice. There is unnecessary redundancy in this case; an alternative solution would be to provide a list of occurrences. In addition, if this repeat represents a significant loss (because it occurs many times, or it is very long), it should have been captured by a non-branching rule. For such an encoding, *cost(s—G)* is the same for all s and is included in the encoding of the grammar. For this example, $|G_1| = 32$.

Unfortunately, this choice of general motifs and encoding proves to be unfit to compete against single repeats. It can be shown that the reduction in the grammar size achieved by replacing one such motif is always bounded by the gain obtained of replacing both exact repeats u and v.

The reason is the additional overhead from the separator symbols (|). A standard strategy in data compression to get rid of separators is to focus on fixed-length words. We adapted this strategy by restricting the inside part of the motif to be of fixed size. That is, we search for motifs of the form $u.^k v$, which matches any substring of the form uwv with $w \in (\Sigma \cup \mathcal{N})^k$. While more restrictive in what they can capture, those motifs allow a more efficient encoding. An example grammar G_2 that represents language L and uses the knowledge of fixed-length motifs is:

$$\begin{aligned} S \rightarrow\ & O\ . \\ & \text{Alice was beginning to get very tired .} \\ & \text{Alice is very tired .} \\ O \rightarrow\ & \text{Alice } I \text{ very tired} \\ I \rightarrow\ & \text{was getting} \mid \text{will be} \mid \text{was getting} \end{aligned} \tag{3.2}$$

which would be encoded as "Alice N_1 very tired . Alice was beginning to get very tired . Alice is very tired # was getting | will be was getting". Only one separator symbol is now needed for the non-terminal N_1, as it indicates the length of the inside rule (which may vary across different non-terminals). The length of the expansion until that point, together with the number of occurrences of N_1 indicates the end of that rule without need of providing an additional end-of-rule symbol. The total size of G_2 is then 25.

We can now deduce the gain introduced by replacing a motif $u.^k v$ with non-terminals O and I:

$$\begin{aligned} f(u.^k v, G) = & \big((|u| + |v|)\ occ_G(u.^k v)\big) \\ & - occ_G(u.^k v) - (|u| + |v| + 1 + 1) - (1 + 1) = \\ & \big((|u| + |v| - 1)\ (occ_G(u.^k v) - 1)\big) - 5 \end{aligned} \tag{3.3}$$

This procedure results in the algorithm $\texttt{NRGreedy}_{\texttt{fix}}$given in Alg. 2. $G_{w \mapsto N}$ refers to the grammar where all occurrences of the string w are replaced by the new non-terminal N and $N \rightarrow w$ is added to the productions. Similarly, $G_{u.^k v \mapsto O,I}$ is the grammar where all realizations of $u.^k v$ are replaced by the new non-terminal O, and the rules are extended with $O \rightarrow uIv$ and $I \rightarrow w$ for all w such that uwv occurs in G.

While a smallest possible fixed-length non-recursive grammar is obviously smaller than a smallest straight-line grammar (because it is more general), our experiments (see Sect. 3.5) show that the final grammars obtained with $\texttt{NRGreedy}_{\texttt{fix}}$ are actually larger than those obtained by minimizing the size of straight-line grammars only.

3.4.2 Post-processing Algorithm

We finally report the results of a simple but effective method that starts from any straight-line grammar, and infers branching rules from it (Alg. 3). This work is reminiscent of efforts done to generalize the output of the `Sequitur` algorithm [68, Chapter 5].

The algorithm starts from any proposal for the smallest grammar problem. We then search for fixed-lengths motifs $u.^k v$, replace them greedily one by one until no further compression can be achieved starting with the one that achieves the highest compression. Note that, because the algorithm starts from a straight-line grammar with no positive-gain repeats left, all repeated left and right contexts are of length one[5], therefore reducing greatly the execution time.

3.5 Experimental Results

We compared the effectiveness of our proposed algorithm with algorithms that approximate the smallest grammar problem in two areas. The first is the direct goal of SGP, namely, to find small grammars that encode the data. We show how the more expressive grammar can lead to consistently smaller grammars, and considerably so in sequences with a large

[5]Which could of course be a non-terminal.

ALGORITHM 2. GREEDY ALGORITHM $\texttt{NRGreedy}_{\texttt{FIX}}$ TO COMPUTE SMALL NON-RECURSIVE GRAMMAR GENERATING s

Input: String s **Output:** non-recursive grammar G such that $s \in L(G)$

1. $G \leftarrow \langle \Sigma(s), \{S\}, S, \{S \to s\} \rangle$;
2. **while true do**:
   (a) $w \leftarrow \max_{w \in (\Sigma \cup \mathcal{N})^*} f(w, G)$;
   (b) $u.^k v \leftarrow \max_{u,v \in (\Sigma \cup \mathcal{N})^*, k \in \mathbb{N}} f(u.^k v, G)$;
   (c) **if** $f(w, G) \leq 0 \wedge f(u.^k v, G) \leq 0$:
   (d) **then return** G;
   (e) **else if** $f(w, G) > f(u.^k v, G)$ **then**:
      i. N is a fresh non-terminal;
      ii. $G \leftarrow G_{w \mapsto N}$;
   (f) **else**:
      i. O, I are fresh non-terminals;
      ii. $G \leftarrow G_{u.^k v \mapsto O,I}$;

ALGORITHM 3. POST-PROCESSING ALGORITHM TO COMPUTE SMALL NON-RECURSIVE GRAMMAR GENERATING s

Input: String s, SGP algorithm $\texttt{sgp}$ **Output:** non-recursive grammar G such that $s \in L(G)$

1. $G \leftarrow \texttt{sgp}(s)$
2. **while true do**:
   (a) $u.^k v \leftarrow \max_{u,v \in (\Sigma \cup \mathcal{N})^*, k \in \mathbb{N}} f(u.^k v, G)$;
   (b) **if** $f(u.^k v, G) \leq 0$ **then**:
      i. **return** G;
   (c) **else**:
      i. O, I are fresh non-terminals;
      ii. $G \leftarrow G_{u.^k v \mapsto O,I}$;

number of fixed-size motifs. We also report results on qualitative measures of the obtained structure, using a linguistic corpus annotated with its syntactic tree structure.

3.5.1 Smaller Grammars

In this section, we compare the compression performance of our algorithm with four SGP solvers: `Greedy` [24, 67], IRRMGP, ZZ [50] and *MMAS-GA* [52]. We report the results on two datasets widely used in data compression: a DNA corpus[6] and the Canterbery corpus[7]. Details about these corpora are in Table 3.1.

Table 3.1: Statistics of the the DNA corpus (left) and Canterbury dataset (right)

We report length, number of different symbols and the number of repeats normalized by length.

sequence	$\|s\|$	$\|\Sigma(s)\|$	$\|\mathcal{R}(s)\|/\|s\|$	**sequence**	$\|s\|$	$\|\Sigma(s)\|$	$\|\mathcal{R}(s)\|/\|s\|$
chmpxx	121,024	4	0.82	alice29.txt	152,089	74	1.45
chntxx	155,844	4	0.77	asyoulik.txt	125,179	68	1.22
hehcmv	229,354	4	1.46	cp.html	24,603	86	4.32
humdyst	38,770	4	0.77	fields.c	11,150	90	5.03
humghcs	66,495	4	13.77	grammar.lsp	3,721	76	3.43
humhbb	73,308	4	9.01	kennedy.xls	1,029,744	256	0.08
humhdab	58,864	4	1.21	lcet10.txt	426,754	84	2.00
humprtb	56,737	4	1.07	plrabn12.txt	481,861	81	1.02
mpomtcg	186,609	4	1.36	ptt5	513,216	159	194.74
mtpacga	100,314	4	0.97	sum	38,240	255	17.44
vaccg	191,737	4	2.21	xargs.1	4,227	74	1.77

The results on the final sizes are reported in Table 3.2. As pointed out before, we did not achieve to obtain smaller grammars by incorporating branching-rules inference during the main process (algorithm $\texttt{NRGreedy}_{\texttt{fix}}$). The final grammars were consistently larger than the simplest baseline (`Greedy`), often considerably so (see for instance `humhdab`, `alice29.txt`). However, the same idea of inferring fixed-motifs proved to be successful when applied as a post-processing. Moreover, this strategy can be applied to any straight-

[6] `http://people.unipmn.it/~manzini/dnacorpus/historical/`
[7] `http://corpus.canterbury.ac.nz/`

line grammar and can therefore be used after any SGP algorithm. Under the columns #Ctx we give the number of branching rules that are inferred. The number of occurrences of these rules is of course much higher in general.

While the reduction in size is small, it applies consistently throughout all the SGP algorithms we tried[8]. The better the original algorithm, the smaller the gain. While this may point towards a convergence of the possible redundancy that can be extracted, it should be noted that our approach runs much faster than the more sophisticated algorithms (*ZZ*, *MMAS-GA*). Moreover, our best result in Table 3.2 become the new state-of-the-art in several cases, and we would expect an even better improvement if starting from the final grammars output by *MMAS-GA*.

We analyzed separately the huge difference in the gain obtained on the `kennedy.xlsx` file. This binary file encodes a large spreadsheet (347×228, in Excel format) containing numerical values, many of which are empty. Most of the gains over the straight-line baselines seem to come from the way these numbers are getting encoded, with a common prefix and suffix and a fixed-length field for the specific value. These fields are therefore ideal candidates for our branching-rule inference. We were able to recreate those results by generating random Excel tables, obtaining improvements of 6 to 33% (relative to the original size of IRRMGP) depending on the number of non-zero entries the table had.

3.5.2 Better Structure

Following the original motivation for closing the gap between the structures found in SGP and the structures that are sought in grammatical inference, we evaluated the obtained branching rules by their capacity for unsupervised parsing. We benchmarked our method in the task of *unsupervised parsing*, the problem of retrieving the correct tree structure of a natural language text. We took the standard approach in the field, starting from the Part-of-Speech (POS) tags of the Penn Treebank dataset [69]. Current *supervised* methods achieve

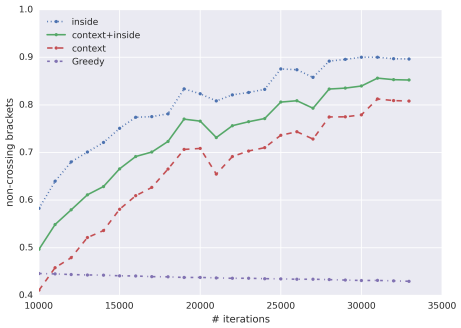
[8] We did not have access to the final grammars of *MMAS-GA*

a performance above 0.9 of F_1 measure [70]. As expected, *unsupervised* approaches report worse performance, around 0.8 [59]. These are in general very computationally intense methods and performance is only reported on top of WSJ10, sentences of size up to 10. We diverge from that, reporting results on all 49,208 sentences[9]. For evaluation we used precision over the set of brackets, together with the percentage of non-crossing brackets [71], a standard practice for which we relied on the EVALB tool[10] which removes singleton and sentence-wide brackets. While precision is the percentage of correctly retrieved brackets, non-crossing brackets is the percentage over the retrieved brackets that do not contradict a gold bracket and give an idea on how not-incorrect the results are (as opposed to correct).

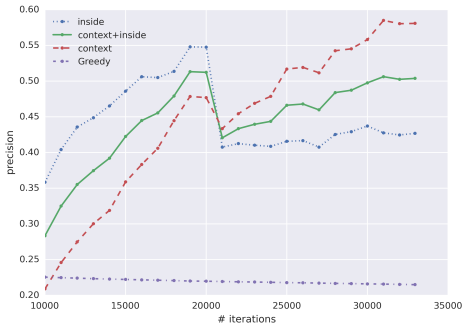
Our focus is on comparing the quality of the brackets of the branching rules with those of the non-branching rules. We furthermore distinguish the brackets covering a context, and the one covering an inside. For the rules $\{O \rightarrow \alpha I \beta, I \rightarrow \gamma_1 | \gamma_2\}$, the inside brackets cover γ_1 and γ_2, while the context rules cover $\alpha\gamma_1\beta$ and $\alpha\gamma_2\beta$. The number of context brackets is always the same as the number of inside brackets.

As before, we are mainly interested in comparing the additional rules added by non-recursive grammars. The `Greedy` algorithm creates around 950,000 brackets, of which only 21% are correct. The proposed post-processing adds another 792 brackets, but with a much higher precision (50.48%) and mostly consistent (85%). In order to evaluate the sensitivity of these results, and to see if they generalize if more brackets are retrieved, we stopped the `Greedy` algorithm earlier: this creates larger grammars, with more options for the creation of branching rules. The final results are summarized in Fig. 3.1. Because of the small number of brackets (Fig. 3.1c) we do not report recall. While the numbers of correct and consistent brackets decreases with increasing number of branching rules, they do so very gently and are much higher than the accuracy of non-branching rules. Furthermore, a stark difference appears between context and inside brackets: while context brackets are much more often correct (reaching almost 60%), they are less consistent than inside brack-

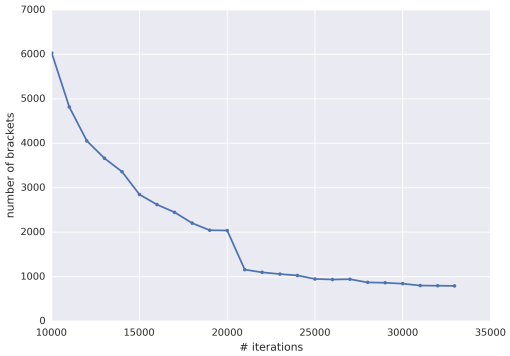
[9] Excluding sentence 1855, for which EVALB evaluation tool we used had trouble processing.
[10] `nlp.cs.nyu.edu/evalb/`



(a) Percentage of non-crossing brackets



(b) Bracketing precision



(c) Number of brackets of branching rules.

Figure 3.1: Structuring accuracy of the post-processing algorithm
x-axis is the number of iterations when we stopped the `Greedy` algorithm.

ets (which have a non-crossing percentage of 90%). These results get their whole meaning when compared to the brackets obtained by just considering the straight-line grammars. Their accuracy varies very little over the iterations, and are always extremely low (around 22%).

Finally, the drop around 20,000 iterations belongs to a point where highly frequent context patterns[11] stop being captured by branching rules and are modeled by repeats. This result also means that the good performance at the end is not due to these easy to model constituents.

As said, reported values on unsupervised parsing of this dataset focuses on sentences of length up to 10, which only represents less than 10% of the total corpus. On these

[11]Most notably opening/closing quotation and parentheses

sentences, the context brackets obtain a precision of 80%, considerably higher than other reported results, although this is not a fair comparison as the number of retrieved brackets is low. But it is worth to highlight that parsing the longer sentences did not pose any problems at all in our (not optimized) implementations: in fact the algorithm was run on the concatenation of the overall set (over $1.3M$ tokens).

3.6 Conclusion

In this chapter of the thesis, we provide a first step towards applying the results around the Smallest Grammar Problem for grammatical inference. We identify a probable reason for past failures, and show how to extend the work inferring small straight-line grammars towards non-recursive ones. In order to keep up comparison with standard benchmarks in the SGP, we design an encoding of this grammar that allows for efficient algorithms. Those algorithms consistently improve over the current state-of-the-art, substantially so in one case (`kennedy.xls` sequence). One direction of future work could focus on formalizing the phenomena exhibited by that sequence and where else it occurs.

With respect to the original motivation of structuring the sequences, the additional rules of our algorithm have a much higher precision than other methods for unsupervised parsing, without explicitly trying to optimize for it. Recall is much lower, due to the few rules that are actually inferred. However, we believe that our results show the potential of this generalized smallest grammar problem for this task and we are considering on how to build on the efficient algorithms developed in the field to capture more such rules.

Table 3.2: Size of the final grammars obtained with the different algorithms for producing smaller grammars

SGP algorithms (non-bold numbers) generate straight-line grammars, while $\texttt{NRGreedy}_{\texttt{fix}}$and the **+Post** columns infer non-recursive grammars. Green/Red down-/up-ward arrows show a reduction/increase in the grammar size with respect to the output of the reference algorithm in each section, and = shows no change. $\#Ctx$ is the number of branching rules detected by the post-processing algorithm. In the cases with -, either the final grammar or the output of ZZ algorithm was not available. The results for MMAS-GA are taken from Benz et al. [52].

Data	**Grammar Size**										
	Greedy	**$\texttt{NRGreedy}_{\texttt{fix}}$**	**+Post**	$\#Ctx$	**IRRMGP**	**+Post**	$\#Ctx$	**ZZ**	**+Post**	$\#Ctx$	**MMAS-GA**
chmpxx	28,704	**29,477** ↑	**28,534** ↓	64	27,683	**27,584** ↓	27	26,024	**26,024** =	0	25,882
chntxx	37,883	**38,212** ↑	**37,703** ↓	71	36,405	**36,285** ↓	25	33,942	**33,942** =	0	33,924
hehcmv	53,694	**54,451** ↑	**53,398** ↓	113	51,369	**51,242** ↓	30	-	-	-	48,443
humdyst	11,064	**11,166** ↑	**11,017** ↓	19	10,700	**10,680** ↓	6	10,037	**10,037** =	0	9,966
humghcs	12,937	**13,655** ↑	**12,908** ↓	14	12,708	**12,708** =	0	12,033	**12,023** ↓	5	12,013
humhbb	18,703	**18,893** ↑	**18,614** ↓	36	18,133	**18,060** ↓	20	17,026	**17,024** ↓	1	17,007
humhdab	15,311	**19,736** ↑	**15,242** ↓	27	14,906	**14,879** ↓	8	13,995	**13,995** =	0	13,864
humprtb	14,884	**17,122** ↑	**14,817** ↓	26	14,492	**14,451** ↓	11	13,661	**13,661** =	0	13,528
mpomtcg	44,175	**45,018** ↑	**43,930** ↓	89	42,825	**42,658** ↓	40	39,913	**39,911** ↓	1	39,988
mtpacga	24,556	**24,878** ↑	**24,408** ↓	58	23,682	**23,608** ↓	16	22,189	**22,189** =	0	22,072
vaccg	43,711	**44,261** ↑	**43,445** ↓	99	41,882	**41,778** ↓	29	-	-	-	39,369
alice29.txt	41,001	**50,777** ↑	**40,984** ↓	7	40,218	**40,218** =	0	37,702	**37,662** ↓	12	37,688
asyoulik.txt	37,475	**45,520** ↑	**37,464** ↓	4	36,910	**36,905** ↓	1	35,001	**34,953** ↓	16	34,967
cp.html	8,049	**8,310** ↑	**8,003** ↓	6	7,974	**7,971** ↓	1	7,768	**7,747** ↓	9	7,746
fields.c	3,417	**3,681** ↑	**3,380** ↓	7	3,385	**3,381** ↓	1	3,312	**3,285** ↓	13	3,301
grammar.lsp	1,474	**1,475** ↑	**1,458** ↓	2	1,472	**1,472** =	0	1,466	**1,462** ↓	1	1,452
kennedy.xls	166,925	-	**99,915** ↓	1,233	166,810	**98,479** ↓	1,174	166,705	**98,258** ↓	1,161	166,534
lcet10.txt	90,100	**115,625** ↑	**89,998** ↓	33	88,778	**88,750** ↓	9	-	-	-	87,086
plrabn12.txt	124,199	**165,122** ↑	**124,009** ↓	58	120,770	**120,760** ↓	2	-	-	-	114,960
ptt5	45,135	-	**45,118** ↓	7	44,129	**44,123** ↓	3	-	-	-	42,661
sum	12,207	**14,722** ↑	**11,761** ↓	52	12,127	**11,868** ↓	34	-	-	-	12,009
xargs.1	2,006	**2,092** ↑	**2,006** =	0	1,993	**1,990** ↓	1	1,973	**1,948** ↓	3	1,955

CHAPTER 4
EMERGENCE AND EVOLUTION OF HIERARCHICAL STRUCTURE IN COMPLEX SYSTEMS

4.1 Introduction

In this chapter, we present *Evo-Lexis*, a modeling framework for the emergence and evolution of hierarchical structure in complex modular systems. There are many hypotheses in the literature regarding the factors that contribute to either the hierarchy or modularity properties. Local resource constraints in social networks and ecosystems [72], modularly varying goals [8, 73, 74], selection for more robust phenotypes [75, 76], and selection for lower connection costs in a network [4] are some of the mechanisms that have been previously explored and shown to lead to hierarchically modular systems. The main hypothesis that we follow in this chapter is along the lines of Mengistu et al. [4], which assumes that systems in both nature and technology care to minimize the cost of their interconnections or dependencies between modules.

An additional focus of our work is the hourglass effect in hierarchical systems. Across many fields, such as in computer networking [77], deep neural networks [78], embryogenesis [79], metabolism [80], and many others [5], it has been observed that hierarchically modular systems often exhibit the architecture of an hourglass. Informally, an hourglass architecture means that the system of interest produces many outputs from many inputs through a relatively small number of highly central intermediate modules, referred to as the "waist" of the hourglass (Fig. 4.1). The waist of the hourglass (also referred to as "core"

The research in this chapter has resulted in the following publication:
P. Siyari, B. Dilkina, C. Dovrolis, **"Emergence and Evolution of Hierarchical Structure in Complex Systems"**, Accepted as a book chapter in *Dynamics On and Of Complex Networks vol. III*, Springer, 2018.

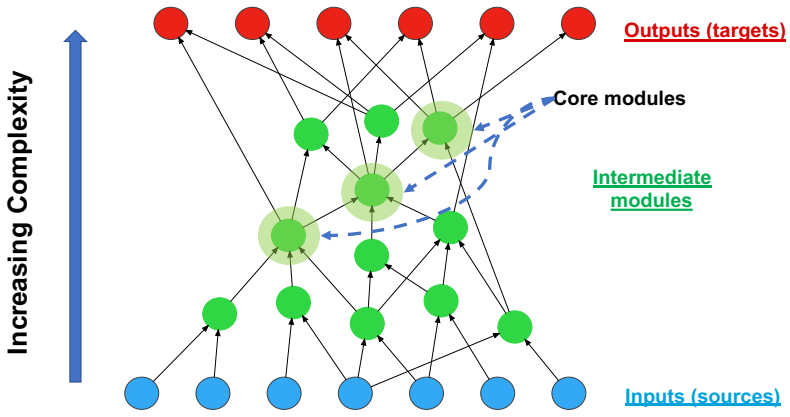


Figure 4.1: A hierarchical system is represented as a directed-acyclic graph. Each module is shown as a node, and the dependencies from more elementary modules to more complex modules are shown as upward edges. The hourglass effect occurs when the system of interest produces many outputs from many inputs through a relatively small number of intermediate core modules (here, highlighted nodes with transparent surroundings) [5].

in Sabrin et al. [5] as well as in this thesis) includes critical modules of the system that are also sometimes more conserved during the evolution of the system compared to other modules [77, 5]. Despite recent research on the hourglass effect in different types of hierarchical systems [5, 77, 81, 82], one of the questions that is still open is to identify the conditions under which the hourglass effect emerges in hierarchies that are produced when the objective is to minimize the cost of interconnections.

To develop *Evo-Lexis*, we extend the previously proposed optimization framework *Lexis* in Chapter 2. Lexis models the most elementary modules of the system as symbols ("sources") and the modules at the highest level of the hierarchy as sequences of those symbols ("targets"). *Evo-Lexis* is a dynamic or evolving version of Lexis, in the sense that the set of targets changes over time through additions (births) and removals (deaths) of targets. *Evo-Lexis* computes an (approximate) minimum-cost adjustment of a given hierarchy when the set of targets changes over time (a process we refer to as "incremental design"). For comparison purposes, *Evo-Lexis* also computes the (approximate) minimum-cost hierarchy that generates a given set of targets from a set of sources in a static (non-evolving)

setting (referred to as "clean-slate design"). The premise behind the incremental design approach is that in practice systems are rarely designed from scratch – instead, they are incrementally modified over time to accommodate the changes (e.g., provide new outputs and potentially to support new inputs every time there is a change).

The questions we focus on are:

1. How do key properties of the emergent hierarchies, e.g., depth of the network, reuse or centrality of each module, complexity (or sequence length) of intermediate modules, etc., depend on the evolutionary process that generates the new targets of the system?

2. Under what conditions do the emergent hierarchies exhibit the so called "hourglass effect?" Why are few intermediate modules reused much more than others?

3. Do intermediate modules persist during the evolution of hierarchies? Or are there "punctuated equilibria" where the highly reused modules change significantly?

4. Which are the differences in terms of cost and structure between the incrementally designed and the corresponding clean-slate designed hierarchies?

We develop the optimization framework, evolutionary target generation processes, and evaluation metrics needed to study these questions. Fig. 4.2 presents an overview of how well-known evolutionary mechanisms such as tinkering, mutation, selection and recombination play a role for the Lexis-DAGs to evolve certain properties such large depth, and the hourglass architecture.

The structure of this chapter is as follows: In Section 4.2, we present the components of the Evo-Lexis framework, along with the metrics that we use for the analysis of evolving hierarchies. In Section 4.3, we evaluate the evolution of hierarchies under different target generation models. Sections 4.4 and 4.5 present further analysis regarding the evolvability and major transitions in hierarchies produced using the most full-fledged (MRS) target

generation model. Finally, Section 4.6 focuses on the comparison between clean-slate and incremental design in terms of cost and structure. In Section 4.7, we review related work in the context of Evo-Lexis. Section 4.8 discusses the results and presents some future research possibilities.

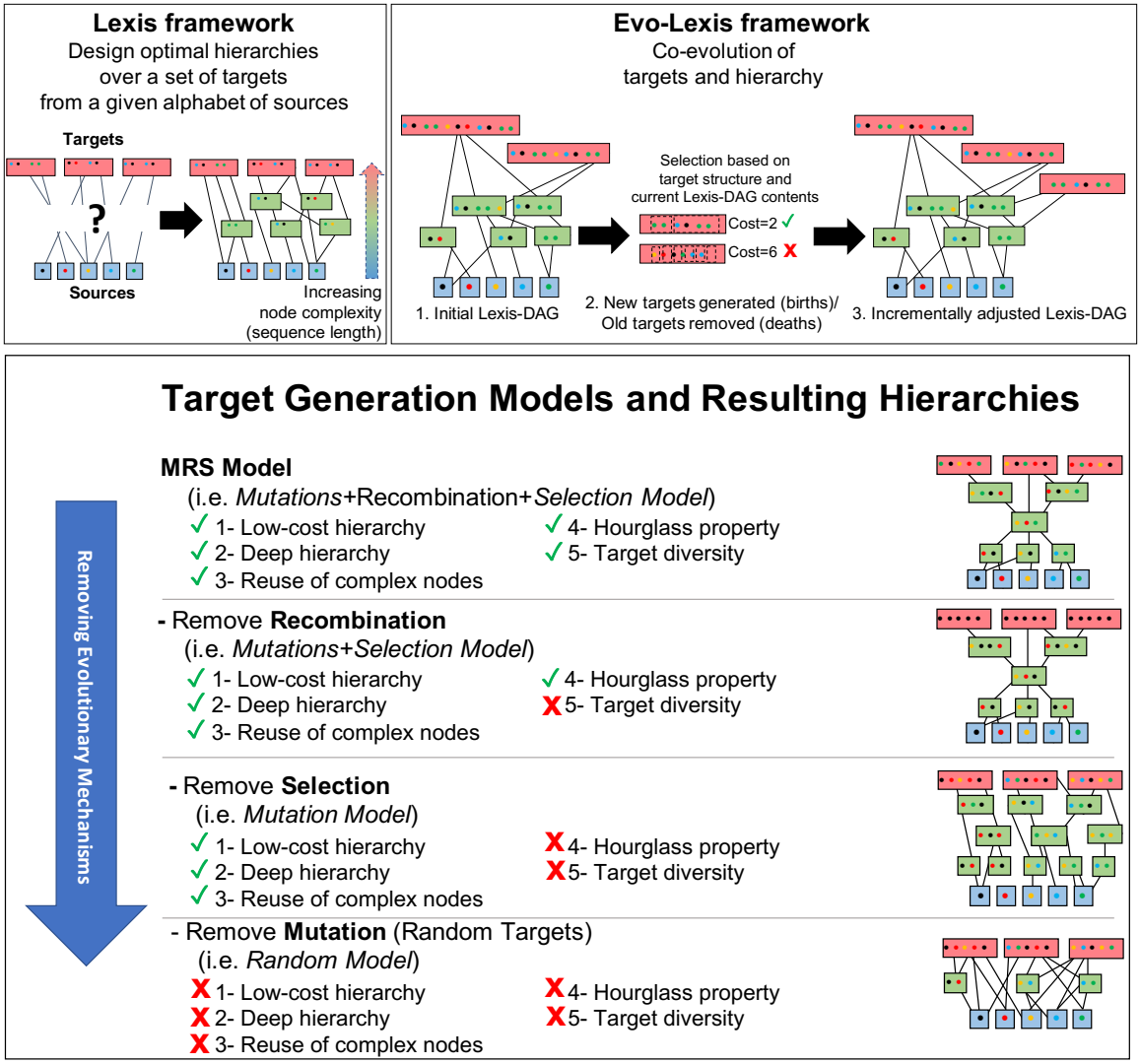


Figure 4.2: Overview of the study in Chapter 4.
The *Evo-Lexis* modeling framework captures the process of incrementally designing optimized hierarchies for a time-varying set of targets. Hierarchies are modeled as *Lexis-DAGs*. We focus on key properties of the resulting hierarchies (e.g., cost, depth, reuse of intermediate components) and on how these properties depend on the evolutionary mechanisms that generate new targets. By focusing on well-known evolutionary mechanisms such as mutations, recombination and selection, we analyze how each of them affects the structure and evolution of the resulting hierarchies. Blue, green and red nodes show source, intermediate and target nodes, respectively. Colored dots represent an instance of a source node and are used to show the extent of diversity among target nodes.

4.2 Evo-Lexis Framework and Metrics

The Evo-Lexis framework includes a number of components that are described below. A general illustration of the framework is shown in Fig. 4.3.

- **Lexis-DAG:** The network that encodes the system's architecture at a given point in time. The inputs of the system are the sources of the DAG and the outputs are the targets.

- **Target Generation Model:** This model specifies the evolutionary process that creates new targets. For simplicity, we consider the addition of only new targets, not new sources. The generation of new targets can be either independent of the current hierarchy (*exogenous target generation*) or it can depend on that hierarchy (*endogenous target generation*).

- **Target Removal Model:** Models the removal of older targets. The total number of targets remains constant during the evolution of the network.

- **Hierarchy Design Algorithm:** This is how the Lexis-DAG is adjusted whenever we introduce new targets. This procedure can be as simple as building a Lexis-DAG from scratch (by running the G-LEXIS algorithm) on the set of existing targets. We refer to this approach as *Clean-Slate design*. On the contrary, the algorithm can be incremental, starting with the previously constructed hierarchy and incorporating new targets in a way that minimizes the adjustment cost. We refer to this algorithm an *Incremental design*, and it is described next.

4.2.1 Incremental Design Algorithm

The Evo-Lexis algorithm generates an optimized hierarchy for the given set of targets in every evolutionary iteration. As mentioned previously, the Clean-Slate design approach is to discard the existing hierarchy and redesign from scratch a new Lexis-DAG for the given

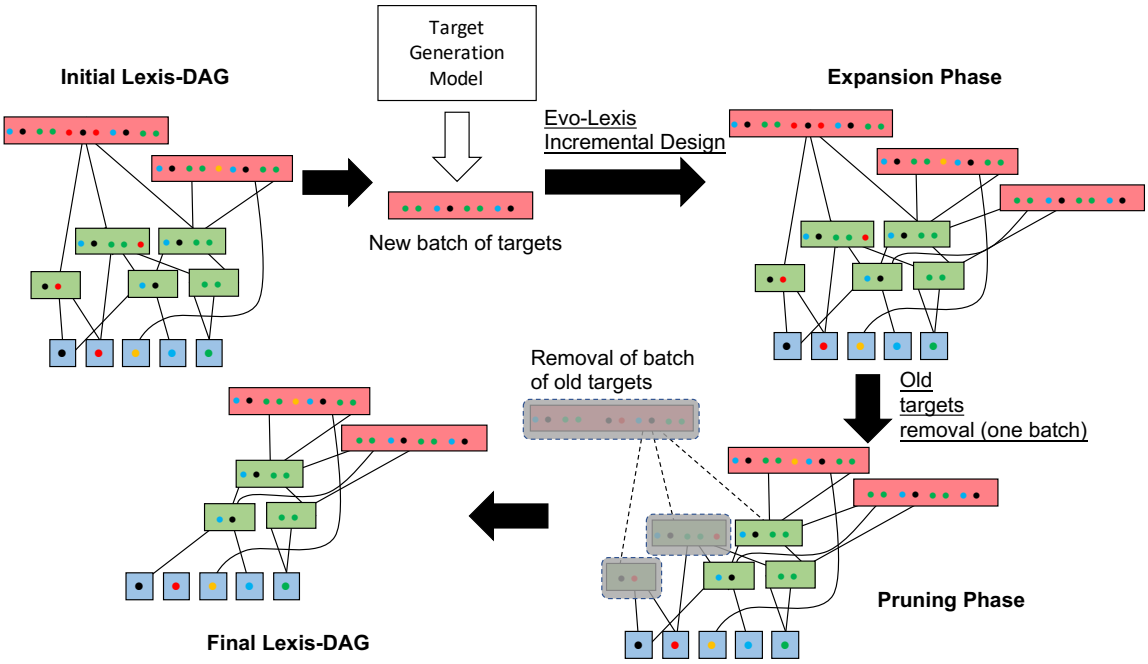


Figure 4.3: A diagram of the Evo-Lexis framework.
In every iteration, the following steps are performed: **(1)** A batch of new targets is generated via a target generation model. **(2)** In the "expansion phase", the new targets are added incrementally to the current Lexis-DAG by minimizing the marginal cost of adding every new target to the existing hierarchy. **(3)** If the number of targets that are present in the system has reached a steady-state threshold, we also remove the batch of oldest targets from the Lexis-DAG. During this "pruning phase," some intermediate nodes may also be removed because every intermediate node in a valid Lexis-DAG should have an out-degree of at least two.

set of targets using the G-LEXIS algorithm. Such a design methodology is not realistic however in either technological or natural evolution. A more realistic approach is to adjust the existing Lexis-DAG incrementally, as described below [2].

In incremental design, given a Lexis-DAG D_0 with a set of targets T_0, a set of new targets T_+ to be added, and a set of old targets T_- to be removed, the problem is to construct a Lexis-DAG D^{INC} that supports the set of targets $\{T_0 \cup T_+ - T_-\}$, and that minimizes the

[2]In Gallé et al. [83], the authors have looked into computing repeats incrementally, and in particular in measuring the gap with respect to the real repeats. The scenario that we are investigating is different in Evo-Lexis in the sense that we are looking for reuse of specific nodes in previous Lexis-DAGs in the current ones, and the calculation of repeats *per se* are not in focus.

cost difference with respect to D_0:

$$\begin{aligned} & min_{D^{\text{INC}}} \left\{ \mathcal{E}\left(D^{\text{INC}}\right) - \mathcal{E}\left(D_0\right) \right\} \\ & \text{s.t. } D^{\text{INC}} \text{ is a Lexis-DAG for } \{T_0 \cup T_+ - T_-\} \end{aligned} \tag{4.1}$$

If $D_0 = \phi$ (i.e., there is no initial Lexis-DAG), $T_- = \phi$, and T_+ is the entire target set, the incremental design problem becomes equivalent to the original Lexis Optimization Problem in Eq. (2.2).

The incremental design problem is NP-Hard (as the original Lexis design problem in which $D_0 = \phi$ and $T_- = \phi$), and so we rely on a heuristic that we refer to as INC-LEXIS. The algorithm proceeds in two phases: first, in the "expansion phase," it adds the set of new targets T_+ attempting to reuse as much as possible existing intermediate nodes. Second, in the "pruning phase", the algorithm removes the set of old targets T_-, and it also removes any intermediate nodes that are left with zero or one outgoing edges.

In more detail, the expansion phase of INC-LEXIS consists of two stages: in stage-1, we reuse intermediate nodes present in D_0 to cover T_+ with minimum cost. In stage-2 of the expansion phase, we further optimize the hierarchy that supports the targets in T_+ by building an optimized Lexis-DAG for them using G-LEXIS. The resulting new intermediate nodes and edges are added in the existing DAG.

Note that stage-1 relates to the well-known *Optimal Parsing* problem, which is: given a set of target strings T, a set of substrings M and the corresponding alphabet S, what is the minimum number of substrings and letters that can construct T from the elements of $M \cup S$? The optimal parsing problem can be formulated as a shortest-path problem in directed graphs [84]. If the length of the targets is N, it can be optimally solved in $O(N + |M \cup S|)$ as the corresponding directed acyclic graph has N nodes and $O(N + |M \cup S|)$ unweighted edges.

In the pruning phase, we remove the oldest batch of targets. We also ensure that there is no redundant node in the Lexis-DAG, as implied by the constraint: $\forall v \in V_M, d_{out}(v) \geq 2$.

This ensures that the Lexis-DAG does not include two types of redundancies: nodes with zero out-degree and nodes that are only reused once.

Figures 4.4 and 4.5 give an example of how INC-LEXIS adjusts a hierarchy, given a set of targets to be added and a set of targets to be removed.

4.2.2 Target Generation Models

The targets are generated through well-known evolutionary mechanisms, such as tinkering/mutation, recombination and selection:

- The generation of new targets from minor changes in earlier targets is similar to *Tinkering/Mutation*. Tinkering is common in technological evolution: small "upgrades" in a software or hardware artifacts are the most common example of this process. In biological systems, it is well-known that mutation is basically "the engine of evolution" [85]. In Evo-Lexis, tinkering/mutation is performed by replacing one character of a given target with a randomly chosen character.

- In the technological world, *Recombination* is known to be one of the central mechanisms for the creation of new technologies [86]. Technological design is often considered to be a search over a space of combinatorial possibilities [87]. In fact, many breakthroughs in the history of technology were in fact just a new combination of existing modules. A recent example is the first version of the iPhone in 2007, which was introduced to be "a phone, an internet communicator and an iPod." In biology, it is well known that recombination and crossover is essential as it produces highly diverse genotypes, compared to mutations.

- *Selection* is an essential mechanism in evolution. In natural systems, selection determines whether a new genotype can survive the competition with existing genotypes (i.e., the incumbents) by evaluating the phenotypic fitness of the former relative to the latter. In the technological world, selection is the process of evaluating the function-

ality and cost of a new product, perhaps during an R&D cycle [88]. In the Evo-Lexis framework, selection is performed to decide whether a candidate target can be accepted, by evaluating the cost of adding that target in the current hierarchy. In other words, selection creates an *endogenous* target generation process in which the existing hierarchy determines the cost of the potential new targets and thus, whether each new target is cost-competitive compared to the targets it evolved from.

MRS Model

The main target generation model we consider is based on **M**utation, **R**ecombination and **S**election, thus called *MRS model*. The mechanism for this model is illustrated in Fig. 4.6. In detail:

1. Two distinct targets t_{s_1} and t_{s_2} (referred to as "seeds") are chosen randomly from the existing set of targets. Their cost is denoted by $C(t_{s_1}) = d_{in}(t_{s_1})$ and $C(t_{s_2}) = d_{in}(t_{s_2})$, respectively, and it is equal to the number of incoming edges that form t_s from the intermediate nodes in the current Lexis-DAG.

2. A randomly chosen "crossover index" $1 \leq i \leq k-1$ is chosen (recall that k is the length of the targets) and the following recombinations are generated:

   - $t_1^* = t_{s_1}[1:i-1] + \widetilde{c} + t_{s_2}[i+1:k]$
   - $t_2^* = t_{s_2}[1:i-1] + \widetilde{c} + t_{s_1}[i+1:k]$
   - $t_3^* = t_{s_2}[i+1:k] + \widetilde{c} + t_{s_1}[1:i-1]$
   - $t_4^* = t_{s_1}[i+1:k] + \widetilde{c} + t_{s_2}[1:i-1]$

   where the numbers in braces show string indices, and $\widetilde{c}$ is a randomly chosen character that represents the mutated element. In other words, each recombination also includes a single-character mutation.

3. For each of the four recombinations, we calculate its cost when it is added as a new target to the current Lexis-DAG. This cost can be seen as the marginal overhead that t_x^* introduces when added to the current hierarchy D_0:

$$C(t_x^*) = \mathcal{E}\left(D^{\text{INC}}(D_0, \{t_x^*\})\right) - \mathcal{E}(D_0) \tag{4.2}$$

where $D^{\text{INC}}(D_0, \{t_x^*\})$ is the new hierarchy after adding t_x^* to D_0 using the INC-Lexis algorithm.

4. The model selects a newly generated recombination t_x^* if it satisfies the following selection constraint:

   - Suppose t_x^* is formed by recombining the fragments t_{x_1} (from t_{s_1}) and t_{x_2} (from t_{s_2}), where the length of these target fragments are $|t_{x_1}|$ and $|t_{x_2}|$.

     The *selection ratio* is defined as:

$$R = \frac{C(t_x^*)}{|t_{x_1}| \times C(t_{s_1}) + |t_{x_2}| \times C(t_{s_2})} \tag{4.3}$$

   - If $R \leq 1$, we definitely accept t_x^*.
   - If $R > 1$, we accept t_x^* probabilistically with *selection probability* $p = e^{-\beta(R-1)}$.

5. If none of the recombinations passes the previous selection constraint, the target generation process is repeated. However, if one or more recombinations pass the selection constraint, the model chooses one of them randomly and adds it as an accepted target in the batch of new targets.

β determines how strongly the current hierarchy influences the selection of new targets. The larger the parameter β is, the less likely it becomes that a new target that is more costly than its seeds (i.e., $R > 1$) will be selected. For large β, we get *Strong Selection* and refer to the model as *MRS-strong*. A small β implies *Weak Selection*, and the model is referred

to as *MRS-weak*. We use $\beta = 1$ and $\beta = 12$ for weak and strong selection, respectively. Fig. 4.7 shows the difference of the two β values for typical values of R (when $R > 1$).

To analyze the effect of each evolutionary mechanism, we also consider target generation models by removing certain elements from the MRS model – hence the name "ablation study."

MS Model

The MS model is derived from MRS by removing recombination (hence the name **M**utations+**S**election Model or MS Model). The model generates new targets as follows:

1. A target seed t_s is chosen from the existing set of targets. Suppose the cost of t_s is $C(t_s) = d_{in}(t_s)$ in the current Lexis-DAG D_0.

2. The seed is mutated (single character mutation), as in MRS model, to t_s^*.

3. We calculate the cost of adding t_s^* to the current Lexis-DAG. This cost can be seen as the marginal overhead that t_s^* introduces when it is added to the current Lexis-DAG:

$$C(t_s^*) = \mathcal{E}\left(D^{\text{INC}}(D_0, \{t_s^*\})\right) - \mathcal{E}(D_0) \tag{4.4}$$

4. The model will select the newly generated target t_s^* if it satisfies the following constraint:

   - $R = \frac{C(t_s^*)}{C(t_s)}$
   - If $R \leq 1$, accept t_s^*.
   - if $R > 1$, accept t_s^* probabilistically where selection probability $p = e^{-\beta(R-1)}$.

   Otherwise, the newly generated target is rejected and the target generation repeats.

M Model

This is derived from the MS model by removing the Selection constraint. Note that with this change the target generation process is not influenced by the current Lexis-DAG and it operates "exogenously" to the hierarchy. This model is referred to as **M**utation Model (or M Model) and it generates targets as follows:

1. Among the targets that exist in the current Lexis-DAG, a seed target t_s is chosen randomly.

2. The seed target t_s is mutated to t_s^* through a random single character mutation.

3. If the newly generated target t_s^* is a duplicate of one of the existing targets, the new target is rejected and the target generation repeats. If not, the generated target is added to the batch of new targets.

RND Model

We also consider a random target generation process, referred to as *RND*, where tinkering/mutation are removed from Mutation model. In this model, a new target is randomly generated using k random and independent choices among the sources.

4.2.3 Key Metrics

Cost Metrics

Normalized Cost: This is the cost of the Lexis-DAG D_T (the Lexis-DAG for the target set T) normalized by the total length of the targets, $\mathcal{L}_T$. We denote the normalized cost by $\mathcal{C}_{\mathcal{N}}(D_T)$:

$$0 \leq \mathcal{C}_{\mathcal{N}}(D_T) = \frac{\mathcal{E}(D_T)}{\mathcal{L}_T} \leq 1 \tag{4.5}$$

Penalty of Incremental Design (PID): This measure evaluates the cost overhead of incremental design relative to a clean-slate design:

$$PID_T = \frac{\mathcal{E}(D_T^{\text{INC}})}{\mathcal{E}(D_T^{\text{CS}})} \tag{4.6}$$

where D_T^{INC} is the incremental design for the target set T, and D_T^{CS} is the clean-slate design for the same set of targets. The value of PID is bounded as follows:

$$1 \leq PID_T \leq \frac{\mathcal{L}_T}{\mathcal{E}(D_T^{\text{CS}})} \tag{4.7}$$

because an incremental design cannot be more efficient than a clean-slate design (at least when the two design problems are optimally solved), and the maximum cost of incremental design is $\mathcal{L}_T$).

Topological Metrics

Average Depth: This metric is an indicator of how deep a Lexis-DAG hierarchy is. For each target t, we calculate the average length of all source-target paths ending on that target: $\overline{d}(t)$. The average across all t is defined as the average depth of the hierarchy:

$$\overline{\mathcal{D}}(D_T) = \frac{\sum_{t \in T} \overline{d}(t)}{|T|} \tag{4.8}$$

Core Stability: We have already defined the core size and the H-score. Here we define an additional metric, related to the stability of the core across time.

We track the stability of the core set by comparing two core sets at two different times. A direct comparison of the core sets via the Jaccard index leads to poor results. The reason is that often the strings of the two sets are similar to each other but not completely identical.

Thus, we define a generalized version of Jaccard similarity that we call *Levenshtein-Jaccard Similarity*:

- The Levenshtein distance $LD(s,t)$ between two strings s and t is the number of deletions, insertions, or substitutions required to transform one string to another. The higher the number of required operations, the more distant two strings are from each other [89].

- Suppose we aim to compute the similarity of two sets A and B of strings. We define the mapping $A \rightarrow B$ where every element $a \in A$ is mapped to the most similar element $b \in B$. We also define the mapping $B \rightarrow A$ from every element $b \in B$ to the most similar element $a \in A$:

$$\begin{cases} A \rightarrow B = \{(a,b)\ s.t.\ a \in A\ \&\ b \in B\ \&\ b = arg\ max_{x \in B} Sim(a,x)\} \\ B \rightarrow A = \{(b,a)\ s.t.\ a \in A\ \&\ b \in B\ \&\ a = arg\ max_{x \in A} Sim(b,x)\} \end{cases} \tag{4.9}$$

  where $Sim(a,b)$ is the similarity of a to b and is calculated as:

$$Sim(a,b) = 1 - \frac{LD(a,b)}{max(|a|,|b|)} \tag{4.10}$$

  Notice that $max(|a|,|b|)$ is the maximum value of Levenshtein distance between a and b. This consideration ensures that if $a = b$ then $Sim(a,b) = 1$, and if a and b have the maximum distance then $Sim(a,b) = 0$.

- Considering both $A \rightarrow B$ and $B \rightarrow A$, we get the union of the two mappings and define the Levenshtein-Jaccard similarity as follows:

$$LevJac(A,B) = \frac{\sum_{(a,b) \in A \rightarrow B} Sim(a,b) + \sum_{(b,a) \in B \rightarrow A} Sim(b,a)}{(|A| + |B|)} \tag{4.11}$$

  We can see that if $A = B$ (all weights are equal to one) then $LevJac(A,B) = 1$. Also if none of the elements in A are similar to B (all the element pairs take zero similarity value), then $LevJac(A,B) = 0$.

For example, suppose that $A = \{abc, cdef, fgh\}$ and $B = \{abcd, cgef, xyh\}$. The similarity of the most similar pairings is shown next:

$$\begin{cases} A \to B = \{(abc, abcd), (cdef, cgef), (fgh, xyh)\} \\ \quad \text{where: } Sim(abc, abcd) = \frac{3}{4}, Sim(cdef, cgef) = \frac{3}{4}, Sim(fgh, xyh) = \frac{1}{3} \\ \quad \Rightarrow \sum_{(a,b)\in A\to B} Sim(a, b) = 1.83 \\ B \to A = \{(abcd, abc), (cgef, cdef), (xyh, fgh)\} \\ \quad \text{where: } Sim(abcd, abc) = \frac{3}{4}, Sim(cgef, cdef) = \frac{3}{4}, Sim(xyh, fgh) = \frac{1}{3} \\ \quad \Rightarrow \sum_{(b,a)\in B\to A} Sim(b, a) = 1.83 \end{cases} \tag{4.12}$$

Hence, we have:

$$LevJac(A, B) = \frac{\sum(A \to B) + \sum(B \to A)}{|A| + |B|} = \frac{1.83 + 1.83}{3 + 3} = 0.61 \tag{4.13}$$

Target Diversity Metric

Suppose we have a set of strings $T = \{t_1, t_2, ..., t_n\}$. The goal is to provide a single number that quantifies how dissimilar these elements are to each other.

- We first identify the *medoid* $\mathcal{M}_T$ within the set T, i.e., the element that has the lowest average distance from all other elements. We use Levenshtein distance:

$$\mathcal{M}_T = arg\ min_{m\in T} \sum_{t\in T} LD(t, m) \tag{4.14}$$

- To compute how diverse the elements are with respect to each other, we average the distance of all elements from the medoid. We call this measure σ_T, the *Diversity* of set T. The bigger the diversity metric, the more diverse the set of strings is (because the distance of each target from the medoid is the number of single-character

operations needed to convert any element within the set to the medoid):

$$\sigma_T = \frac{\sum_{t \in T} LD\left[t, \mathcal{M}_T\right]}{|T|} \tag{4.15}$$

4.3 Computational Results

4.3.1 Parameter Values and Evolutionary Iteration

We can summarize an evolutionary iteration of the Evo-Lexis framework as follows:

1. Initially, we start with a small number s of randomly constructed targets. Each target has the same length k, and the number of possible sources is n. An initial Lexis-DAG is constructed using the G-LEXIS algorithm.

2. In every evolutionary iteration, the following steps are performed:

   (a) A new batch of b targets is generated via a target generation model.

   (b) In the Incremental Design approach, the Evo-Lexis algorithm adjusts the existing hierarchy minimizing the marginal cost of adding each new target in the existing hierarchy.

   (c) If the total number of targets that are present in the system have reached a steady-state (the number of targets is T_s), we also remove the oldest batch of b targets from the Lexis-DAG. This target removal process may also trigger the removal of intermediate nodes that are not reused by at least two other nodes in the hierarchy. The total number of targets remains constant (T_s) because the number of target additions is equal to the number of removals (b).

   (d) The evolutionary process is repeated for a user-specified number of iterations. The parameters n, k and b do not change during this process. We run each model ten times for a total of 5,000 iterations. We take the mean value of each metric.

The parameters used in the following experiments are presented in Table 4.1.

Table 4.1: Definition and parameter values of Evo-Lexis in following experiments

Parameter	Definition	Value
s	Number of initial targets	10
n	Number of sources	100
k	Target length (characters)	200
b	Batch size for new targets birth/old targets death	10
T_s	Steady-state number of targets present in Lexis-DAG	100

4.3.2 Results

Emergence of low-cost hierarchies due to tinkering/mutation and selection In Fig. 4.8a and 4.8b, we observe a significant reduction in the normalized cost between the RND model and all other models. The main reason for this reduction is that in all other models, we generate targets that are similar to earlier targets and not randomly constructed. Further, we observe that endogenous models (MS-strong and MRS-strong) further reduce the cost of the resulting hierarchies. The reason is the large bias for selecting targets that can be constructed with lower (or comparable) cost than the seed targets they evolved from. Thus, introducing tinkering/mutation and selection both contribute to the emergence of more efficient hierarchies in the Evo-Lexis framework.

Low-cost design resulting in deeper hierarchies and reuse of more complex modules Having a lower cost hierarchy also means that intermediate nodes are reused more frequently and/or that those intermediate nodes are more complex (i.e., longer strings). We observe this across models in Fig. 4.8c, 4.8e, 4.8d and 4.8f – models with lower normalized cost have deeper Lexis-DAGs and higher intermediate node length. These longer re-used nodes further decrease the cost of the hierarchy. Hence, tinkering/mutation and selection also develop deeper hierarchies with longer intermediate nodes. These two outcomes are ubiquitously observed in both natural and technological systems. Examples include call-graphs and metabolic networks. For instance, for the OpenSSH call-graph and the monkey

metabolic network, it has been reported that the underlying dependency networks have an average depth of 10.4 and 8.1, respectively [5].

The recombination mechanism creates target diversity Realistic hierarchies should support a diverse set of requirements or outputs. For example, in network protocol stacks, many different functionalities at the top level of the hierarchy (application layer) are supported by the same hierarchical infrastructure. In our framework, this translates to having a set of targets with high diversity. In Fig. 4.8g and 4.8h, we show the target diversity across different models. The RND model produces the highest target diversity as there are no correlations among the generated targets. In Fig. 4.8h, we observe that the tinkering/mutation in the M model results in 50% to 70% decrease in target diversity. Strong selection in the MS-strong model further decreases the diversity to the point that the targets are almost identical, with only minor variations of the same main string. Such low target diversity is not realistic in natural and technological systems. The reason that the MS-strong model behaves in this manner is that it generates new targets only through single-character mutations and only when the resulting mutants can be constructed using the existing intermediate nodes (otherwise they would have much higher cost and they would not be selected). Hence, the set of accepted new targets gets very narrow and quite similar to its seed targets.

In biological systems, the evolution of complex species required recombination and sexual reproduction (i.e., crossover). Similarly in the Evo-Lexis framework, the addition of recombination in the MRS model results in increased target diversity (Fig. 4.8g) while keeping the earlier properties of the Lexis-DAGs (i.e., low-cost, large depth, long intermediate nodes).

Reuse of complex modules in the core set by strong selection Looking at the contents of the core at the 5,000th iteration of all models in Fig. 4.10, shows that in models without selection, or with weak selection, the core includes only a small number of intermediate nodes. The reason is that random mutations make the reuse of longer intermediate nodes

unlikely. Note that this does not mean that long intermediate nodes do not exist in Lexis-DAGs under the M & MS-weak & MRS-weak models – such nodes are less likely, however, to be reused often. As a result, shorter nodes and mostly sources are more likely to appear in the new targets, and end up in the core set.

On the other hand, models with strong selection (MS and MRS) limit the locations where the seed(s) can be mutated when generating new targets. This constraint results in reusing longer intermediate nodes. Thus, selection creates a bias towards the reuse of longer intermediate nodes. In the long run, this results in some long nodes dominating the core set in the MS-strong and MRS-strong models (Fig. 4.10d & 4.10f).

Emergence of hourglass architecture due to the heavy reuse of complex intermediate modules in models with strong selection Appearance of longer and heavily reused intermediate nodes in the models with strong selection means that the architecture exhibits the hourglass effect. Indeed, we observe in Fig. 4.9a & 4.9b that the core size gets significantly smaller in the presence of strong selection (MS and MRS models). Additionally, Fig. 4.9c & 4.9d show that the MS-strong and MRS-strong models also result in higher H-score values (0.4 and 0.65 on average, respectively). Lexis-DAGs with high H-score values have a small core size with respect to the equivalent flat Lexis-DAG whose core is made up of sources and targets only.

Overall, the reuse of longer intermediate nodes caused by selection results in hierarchies with an hourglass architecture. This observation is consistent with a mechanism (known as *Reuse-Preference [5]*) that was proposed earlier for the emergence of the hourglass effect in general dependency networks.

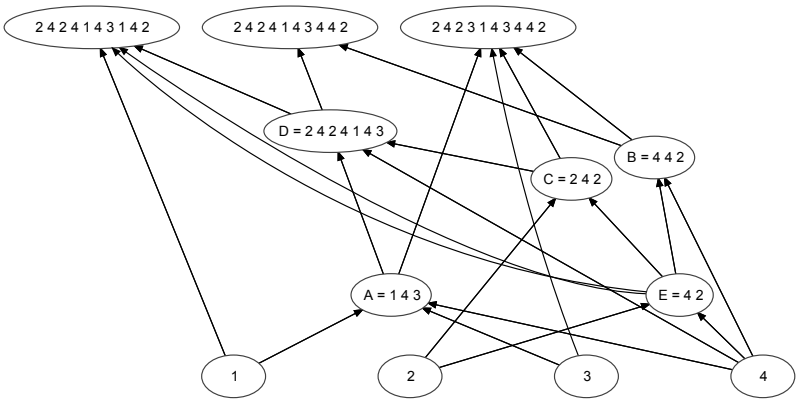
Stability of the core set due to selection Selection also promotes the stability of the core set, as shown in Fig. 4.9h for the MS-strong model. We see an increase in core stability (i.e., similarity of the core during evolution) compared to the MS-weak and M models whose cores mostly consist of sources. Similarly, a stable core is also observed in the MRS-

weak and MRS-strong models in Fig. 4.9g. We have already seen that long intermediate nodes appear more often in the core set of models with strong selection. Hence the core stability results show that selection not only contributes to the emergence of a small core, consisting of few highly reused intermediate nodes, but it also promotes the conservation of these core nodes during evolution. This observation is in agreement with the properties of several systems in which the waist of the hourglass architecture includes critical modules of the system that are highly conserved [77, 5]. We return to this point later, where we further show that this core stability is occasionally interrupted by major transitions and punctuated equilibria.

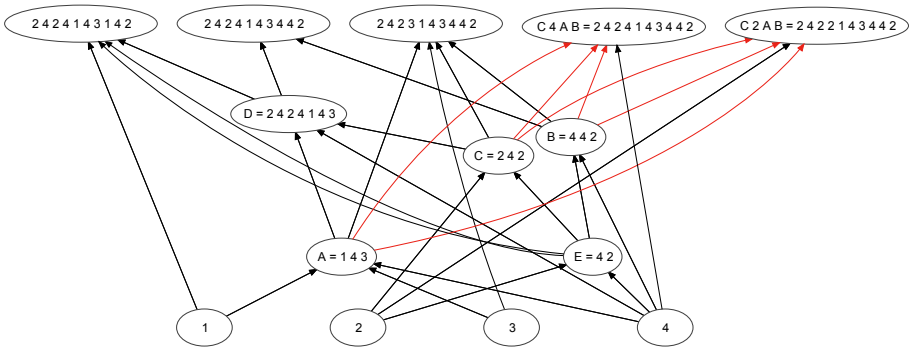
Fragility caused by stronger selection Fig. 4.9e and 4.9f show how the generated hierarchies perform in terms of *robustness*, when we remove the most central nodes in the system, i.e., the members of the core. Robustness generally relates to the ability to maintain a certain function even when there are internal or external perturbations [5]. Fig. 4.9f and 4.9e show how the removal of one or more core nodes, in order of importance, contributes to cutting source-target paths in each of the Lexis-DAGs produced (at the 5,000th iteration of each model).

In hourglass architectures (MS-strong and MRS-strong model), core nodes contribute much more significantly to the overall hierarchy by covering many more source-target paths. Hence, such architectures are fragile if the core nodes are perturbed. This observation is similar to the concept of removal of hub nodes in scale-free network [90]. Weakening selection, reduces the H-score (as in Fig. 4.9c) and, hence, reduces the contribution of core nodes in covering source-target paths.

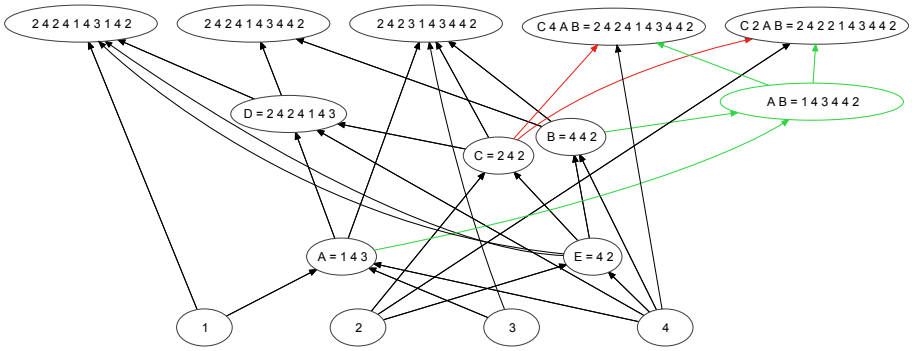
Fig. 4.11 summarizes the properties of the hierarchies that emerge in the models we described in this section.



(a)

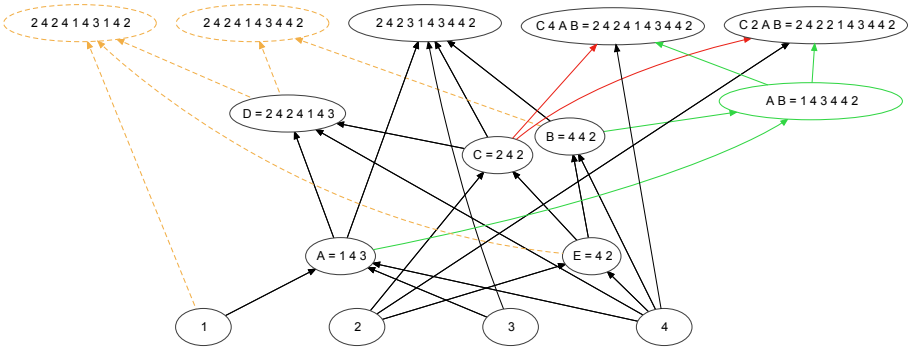


(b)

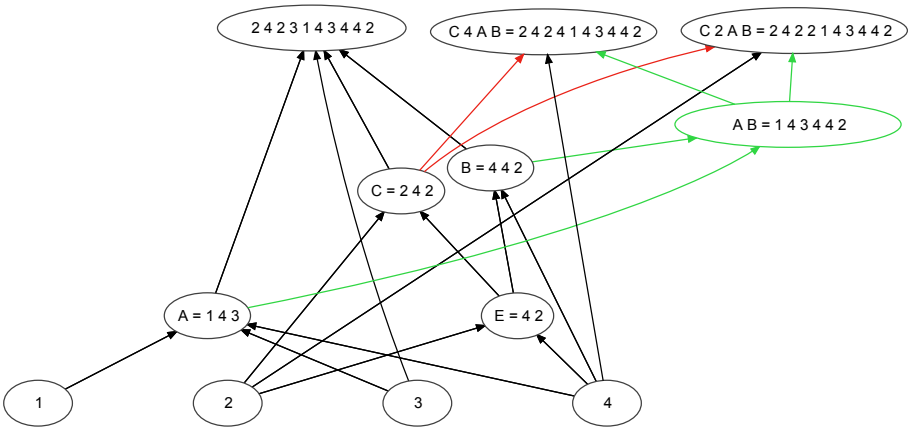


(c)

Figure 4.4: Illustration of INC-LEXIS.
(a) Initial Lexis-DAG D_0 with $T = \{2424143142, 2424143442, 2423143442\}$, $S = \{1, 2, 3, 4\}$ and $M = \{2424143, 442, 242, 143, 42\}$. **(b)** The new targets are $T_+ = \{0424143442, 2424143242, 2422143442\}$. In the first stage of INC-LEXIS, the substrings in $M \cup S$ are reused to construct T_+. Red edges show the reuse of substrings in T_+. Node labels show the representation of each node using the extended alphabet formed by intermediate nodes. This representation is used in the second stage of the expansion phase to run G-LEXIS on T_+. **(c)** The Lexis-DAG after running G-LEXIS on the set T_+ in its extended alphabet form. The green nodes and edges are the results of this stage.
(Continued in Fig. 4.5)



(a)



(b)

Figure 4.5: (Continued from Fig. 4.4) Illustration of INC-LEXIS.
(a) The target nodes 2424143142 and 2424143442 are removed during the pruning phase. All incoming edges (dashed and shown in yellow) will also be removed, which leaves the node $D = 2424143$ with zero out-degree. **(b)** The final Lexis-DAG after removal of targets and intermediate nodes with zero and one out-degree.

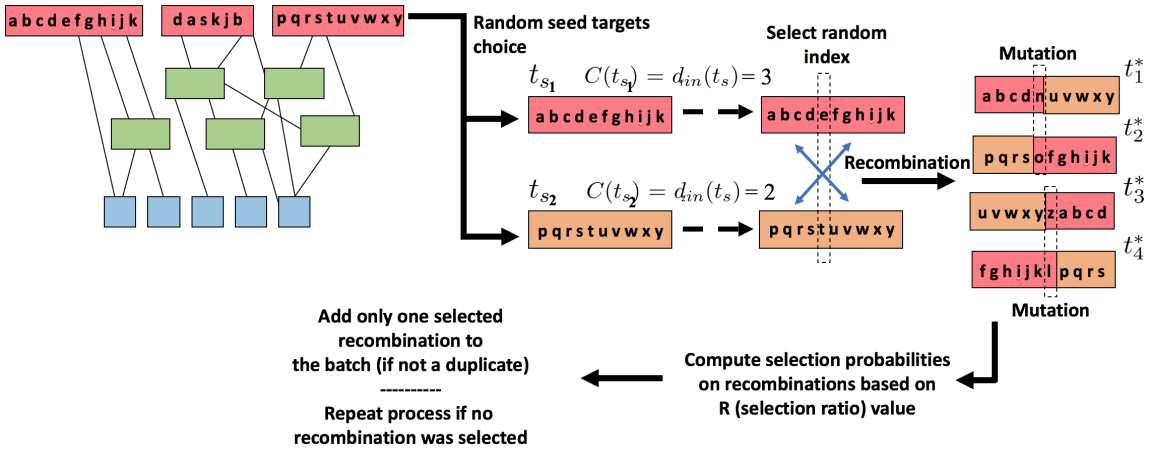


Figure 4.6: Illustration of MRS Model

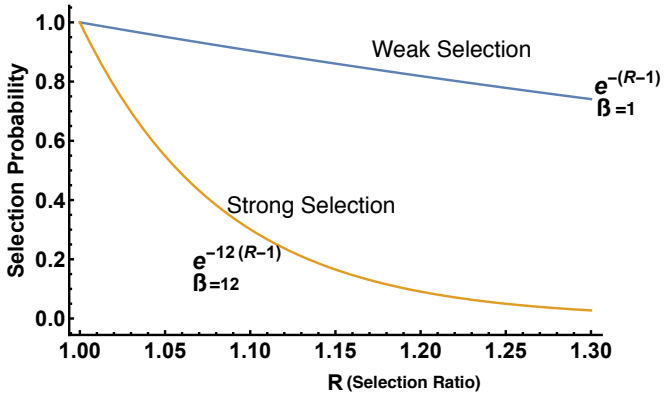
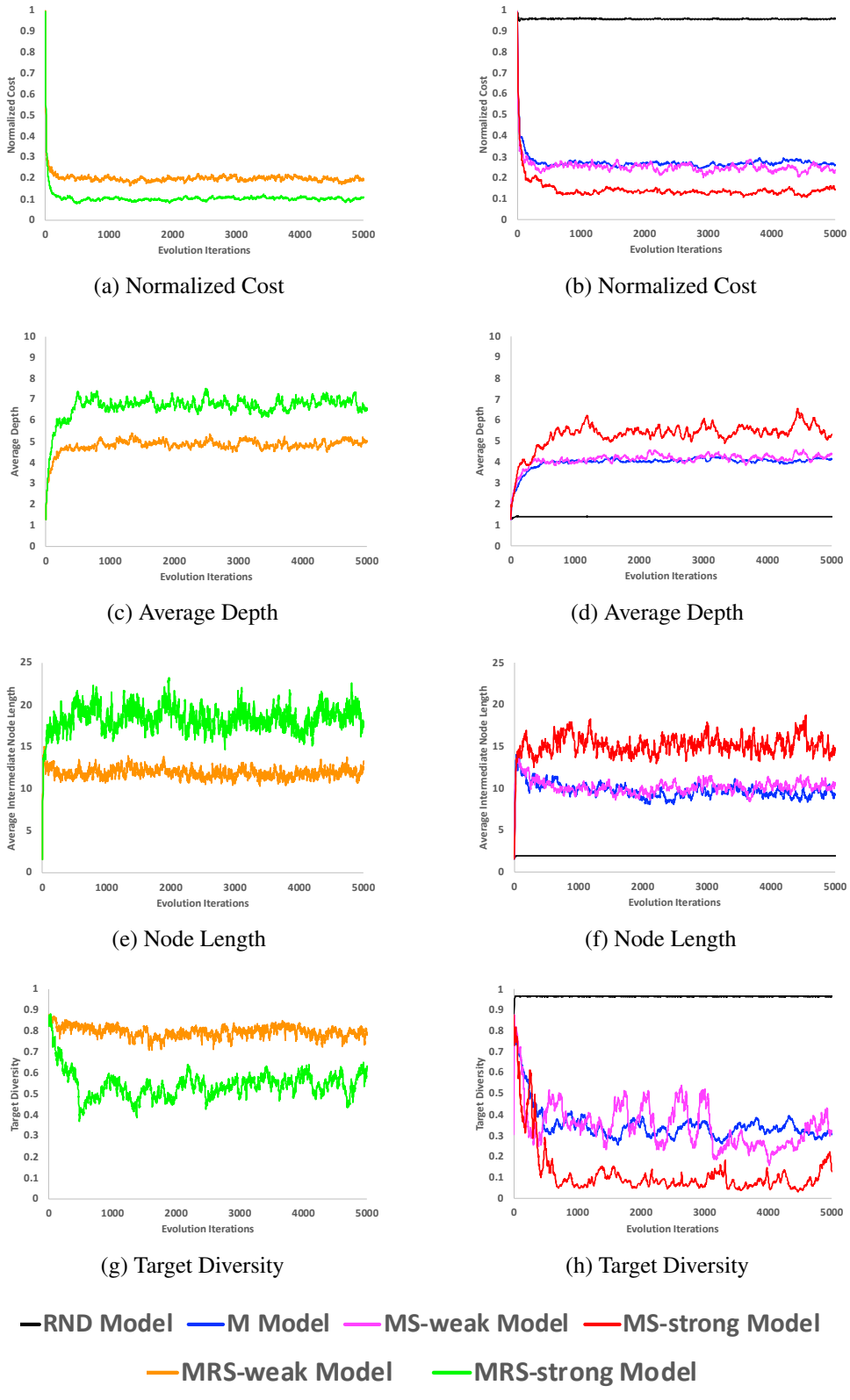


Figure 4.7: The difference of the new target acceptance probability for weak ($\beta = 1$) and strong ($\beta = 12$) selection.

R is the ratio between the cost of the new candidate target and the cost of the targets it evolved from. In MRS-weak, the probability of accepting the new target is high. However, this probability quickly drops in the MRS-strong model.



(a) Normalized Cost

(b) Normalized Cost

(c) Average Depth

(d) Average Depth

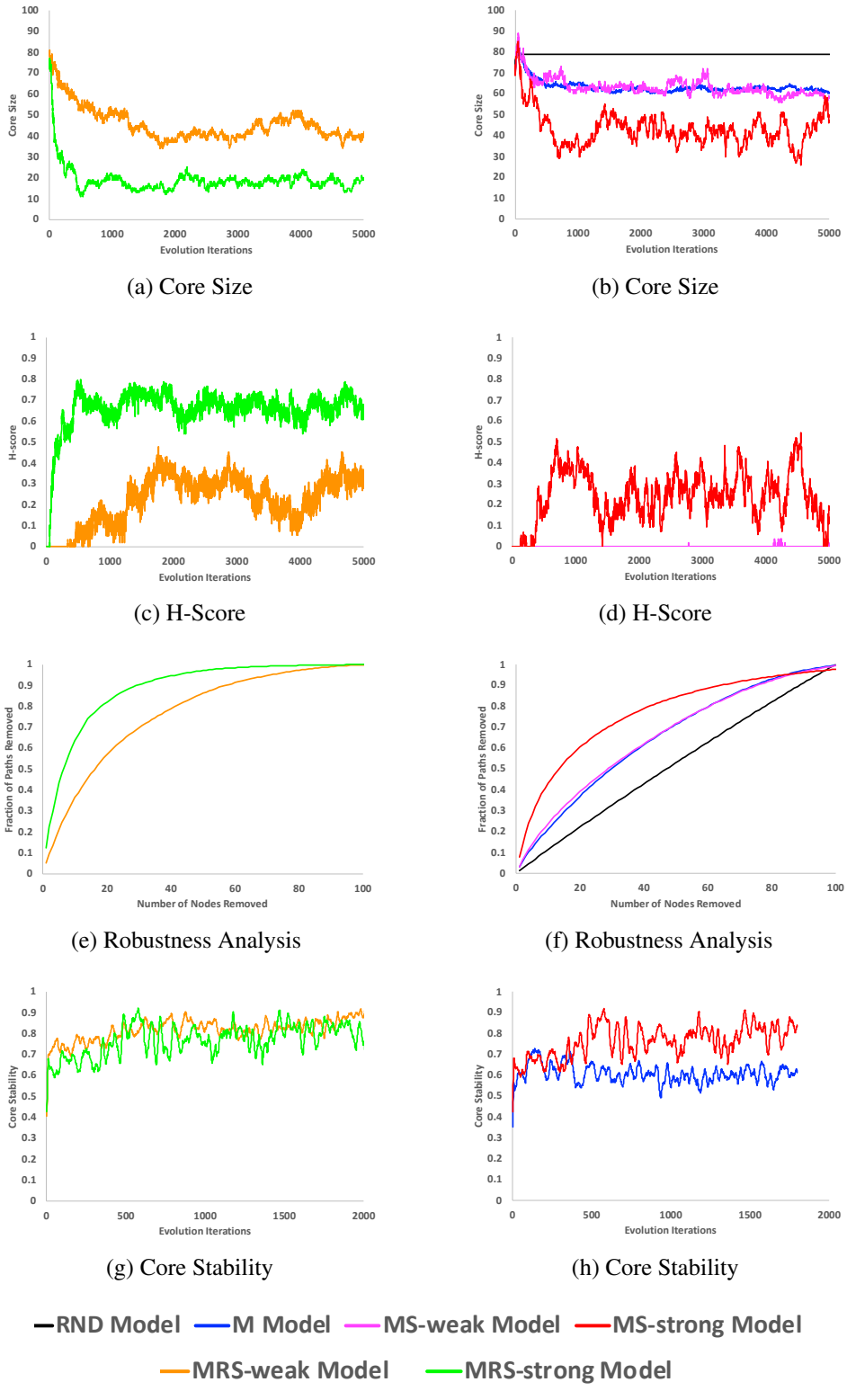
(e) Node Length

(f) Node Length

(g) Target Diversity

(h) Target Diversity

Figure 4.8: Normalized Cost, (average) Hierachical Depth, (average) Intermediate Node Length, and Target diversity of Lexis-DAGs produced by various target generation models (weak selection models: $\beta = 1$, strong selection models: $\beta = 12$). (Continued in Fig. 4.9)



(a) Core Size

(b) Core Size

(c) H-Score

(d) H-Score

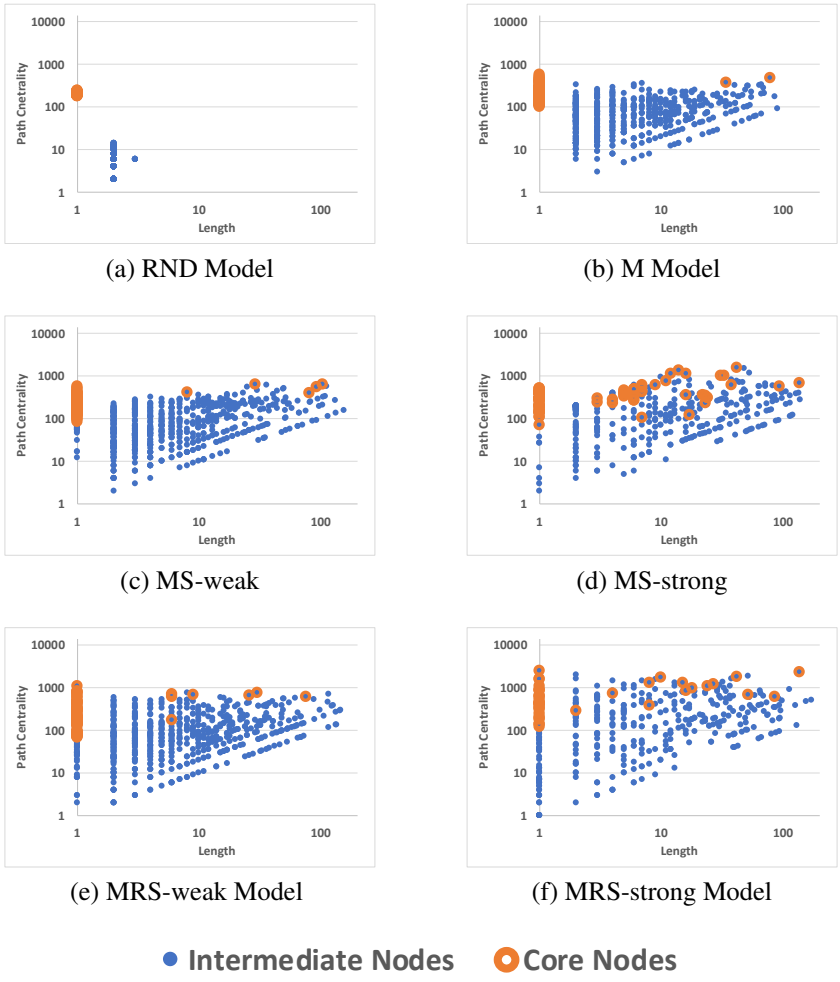
(e) Robustness Analysis

(f) Robustness Analysis

(g) Core Stability

(h) Core Stability

Figure 4.9: (Continued from Fig. 4.8) Core size, H-score, Robustness to core node removals, and Core stability of Lexis-DAGs produced by various target generation models (weak selection models: $\beta = 1$, strong selection models: $\beta = 12$). For core selection, we set $\tau = 0.85$. For core stability, a sliding window equal to the size of 10 batches is used to track changes in the core set.



(a) RND Model

(b) M Model

(c) MS-weak

(d) MS-strong

(e) MRS-weak Model

(f) MRS-strong Model

Figure 4.10: Comparison of node length and path-centrality in Lexis-DAGs at the 5,000th iteration
(for weak selection model $\beta = 1$ and for strong selection model $\beta = 12$). For core selection, we set $\tau = 0.85$.

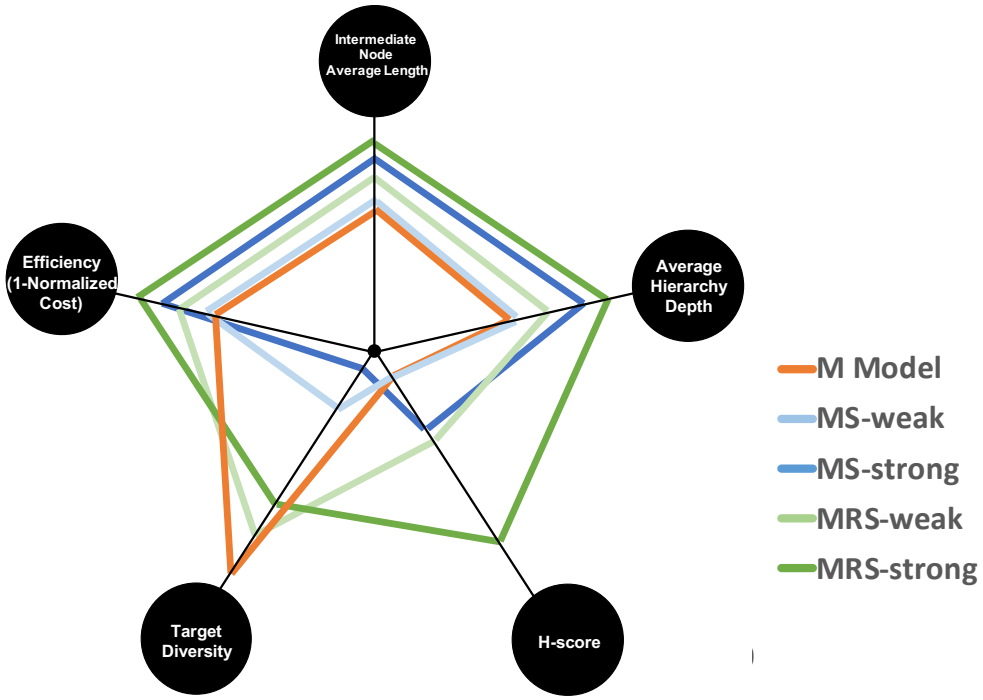


Figure 4.11: Visualizing the various properties of the generated hierarchies (excluding the RND model). The MRS model produces all properties. This figure shows an approximate value for each metric at the 5,000th iteration of evolution. We define Efficiency $= 1 -$ Normalized Cost.

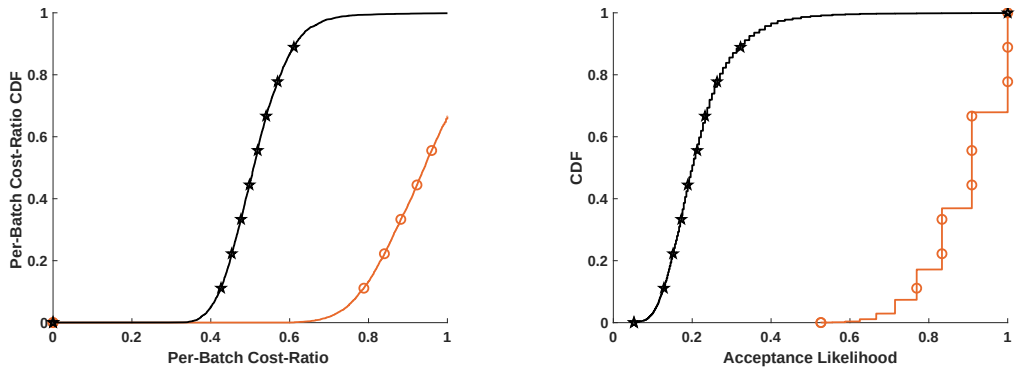
4.4 Evolvability and the Space of Possible Targets

As shown in the previous section, the MRS-strong model leads to hourglass hierarchies, maintaining at the same time significant target diversity. In this section, we further show that hourglass architectures have two important properties. On the positive side, they are more evolvable in the sense that new targets can be constructed at a low cost, mostly reusing the intermediate modules in the core of the hierarchy. On the negative side however, hourglass architectures only accept a small fraction of the candidate new targets, restricting what a biologist would refer to as the "phenotypic space" of the system. This interplay between evolvability and the space of feasible system phenotypes or functions is an important issue in both biological and technological systems (e.g., Internet architecture [91]).

We first look at the cost of targets produced with and without selection. For this pur-

pose, we compare two models: one is the MRS-strong model that acts as an "endogenous" target generation process. The other is a variation of MRS without selection that we call *MR model* (only mutations and recombination) – this model is an "exogenous" target generation process that does not depend on the current state of the hierarchy. The MR model allows us to examine how selection affects the cost and space of acceptable targets with and without the selection constraint.

In Fig. 4.12, we calculate the ratio between the average cost of accepted targets per batch in the MRS-strong model over the corresponding cost in the MR model – we refer to this as *MRS-over-MR per-batch cost-ratio*. The average and median values of this ratio are 0.53 and 0.52, respectively. This observation suggests that the targets generated under stronger selection are of much lower cost (around half) compared to the targets generated without selection. So, the presence of strong selection allows the system to construct new targets at a much lower cost because those selected targets can be constructed mostly reusing the intermediate nodes present in the hierarchy.



(a) CDF of Per-Batch Cost-Ratio Values of MRS Model

(b) CDF of Target Acceptance-Likelihoood Values in MRS Model

Figure 4.12: CDF of MRS-over-MR per-batch cost-ratio and CDF of the target acceptance-likelihood.

(a) CDF of MRS-over-MR per-batch cost-ratio, the ratio between the average cost of targets per batch in the MRS model (weak or strong selection) over the average cost of targets per batch in the MR model. **(b)** CDF of the target acceptance-likelihood, i.e., the number of accepted targets generated per-batch in the MRS model divided by the total number of generated targets per batch with the same model.

As a result of strong selection, the *acceptance-likelihood* of new targets generated by the MRS-strong model is much lower than that with the MR model. Specifically, the acceptance-likelihood in Fig. 4.12b is defined as the fraction of accepted targets generated per-batch. The mean and median of this likelihood in the MRS-strong model are equal to 0.2. In other words, about 80% of the new targets generated through mutations and recombination are not selected because their cost, given the existing architecture, would be prohibitively high.

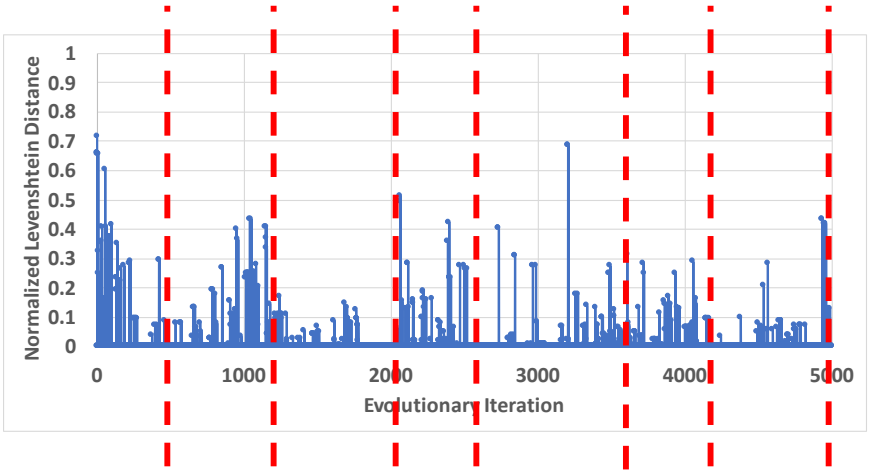
It should be also noted that the MRS-weak model behaves quite similar to the MR baseline in terms of both the MRS-over-MR cost ratio and the target acceptance likelihood.

Overall, the results in this section show that despite having the benefit of lower cost new targets, and thus higher evolvability, selection restricts significantly the phenotypic space of accepted new targets. Given that the MRS-strong model generates hourglass architectures, we can summarize as follows: hourglass-like hierarchies under the MRS-strong model allow the construction of new functions (accepted targets) at a low cost, by mostly reusing core modules, but at the same time such architectures significantly restrict which of these functions can be supported. Targets that are quite different than the intermediate modules of the existing hierarchy would most likely not be selected.

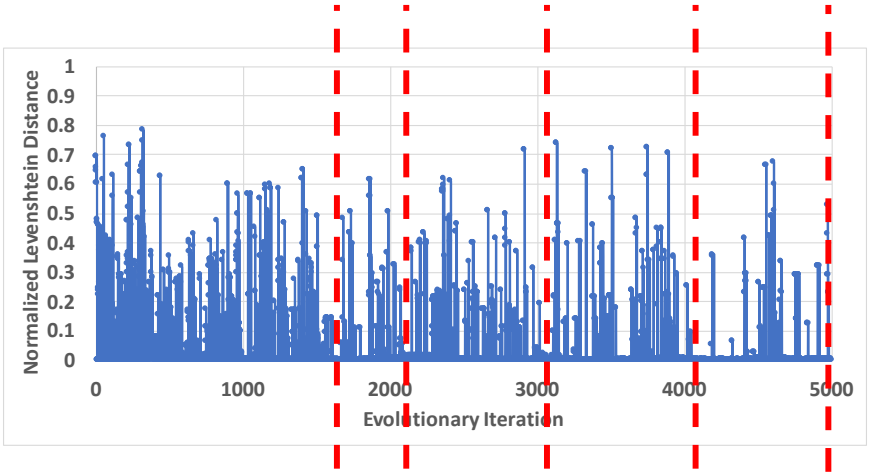
4.5 Major Transitions

Major transitions have been an important and interesting phenomenon in both natural and technological evolution. Such transitions create significant shifts in evolutionary trajectories, ecosystems and "keystone species" [92]. There are many examples of such events in natural systems, such as the "invention" of sexual reproduction and evolution of multicellularity [93]. In technological evolution, innovations occasionally lead to the emergence of disruptive new technologies, such as the steam engine in the 19th century or air transportation in the 20th century. In the context of computing, the evolution of programming languages has gone through punctuated equilibria, interrupted by new languages that were

developed by tinkering or combining different structural components of older languages [94].



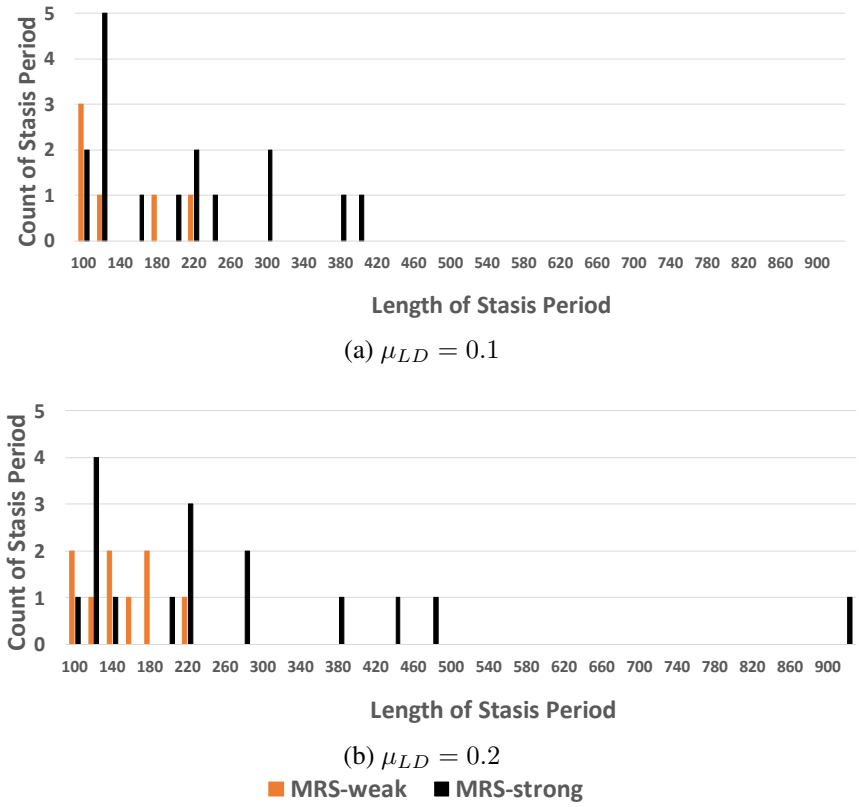
(a) MRS-strong Model



(b) MRS-weak Model

Figure 4.13: Variability across successive iterations of the top-1 core node (measured using the Levenstein distance) in the MRS model (both strong and weak selection).
The highlighted iterations illustrate some of the stasis periods, in which the top-1 core node remains identical for many iterations.

The results of Fig. 4.9g suggest that the structure of the core is locally stable, when comparing core nodes in adjacent iterations. To further investigate the stability of the core during evolution, we focus on the most central node in the core of the Lexis-DAGs, i.e., the core node that covers the largest fraction of source-target paths. We refer to this node of the Lexis-DAG as *top-1 core node*.

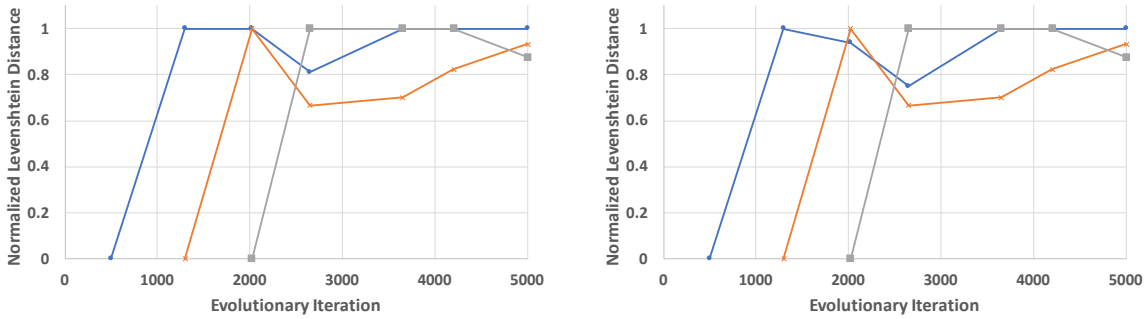


(a) $\mu_{LD} = 0.1$

(b) $\mu_{LD} = 0.2$

Figure 4.14: Count of stasis periods (lasting at least 100 iterations) for two values of the Levenshtein distance threshold, μ_{LD}, in Fig. 4.13.
Strong selection leads to longer and more frequent stasis periods.

First, we track the variability of this node locally, by comparing its normalized Levenshtein distance to the top-1 core node in the next iteration. Fig. 4.13 shows the results of this analysis for both MRS-strong and MRS-weak. In the MRS-strong model, we observe that in most iterations the top-1 core node does not change significantly. Even though there are some spikes in which the Levenshtein distance is larger than 0.2, in 82.6% of the evolutionary iterations the variability of the top-1 core node is less than that. Further, there are several *stasis periods* in which the top-1 core node is practically the same (Levenstein distance lower than 0.1 or even 0). In Fig. 4.13 we highlight with red vertical lines a small number of stasis periods in which the top-1 core node remains exactly the same for tens of hundreds of iterations. On the other hand, the MRS-weak model has significantly higher variability in the top-1 core node, and fewer/shorter stasis periods. This suggests



(a) Top-1 Core Node Changes in MRS-strong

(b) Top-2 Core Node Changes in MRS-strong

Figure 4.15: Starting from three different stasis periods (with $\mu_{LD} = 0.1$), the top-1 and top-2 core node does not stay the same in subsequent stasis periods.
The normalized Levenstein distance between the top-1 and top-2 node at the start of each curve and at successive stasis periods is close to 1, suggesting that these nodes have changed. We observed similar results for other core nodes.

that selection is the key factor in generating these long periods of stability in the core of the hourglass architecture.

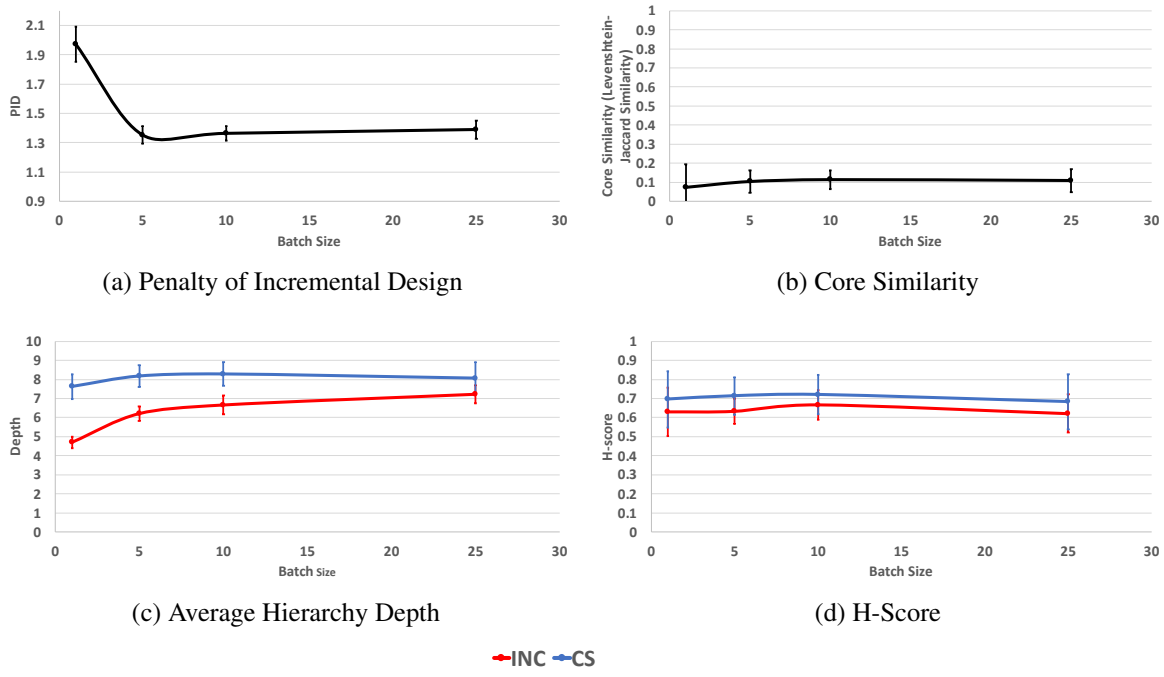
To further quantify this point, we focus on stasis periods that last at least 100 iterations (recall that the entire evolutionary paths in these results consist of 5000 iterations). Fig. 4.14 shows that there are fewer and shorter stasis periods in MRS-weak model than in MRS-strong. The fraction of iterations that account for stasis conditions is $\frac{478}{5000} \sim 0.095$ in MRS-weak, and $\frac{2928}{5000} \sim 0.585$ in MRS-strong, when the minimum Levenshtein distance is $\mu_{LD} = 0.1$ (also $\frac{1049}{5000} \sim 0.209$ in MRS-weak and $\frac{4133}{5000} \sim 0.826$ in MRS-strong when $\mu_{LD} = 0.2$).

The presence of stasis periods under strong selection suggests that the most central intermediate nodes at the waist (or core) of the hourglass architecture can be quite stable and time-invariant. What happens however across different stasis periods? Does that stability persist across different stasis periods, or does the architecture exhibit major transitions and punctuated equilibria?

To answer this question, we focus again on the top-1 core node and measure its variability across successive stasis periods. In Fig. 4.15, we consider three different stasis periods (one curve for each initial stasis period), and calculate the normalized Levenstein

distance between the top-1 core node in its initial stasis period and the top-1 core node in subsequent stasis periods. Note that the top-1 core node changes significantly across stasis periods. In fact, the Levenshtein distance is so high (often close to 1), suggesting that these are completely different core nodes. This observation gives more evidence that the top contributors to the core can lose their importance during evolutionary time scales, causing major transitions in both the core set and, consequently, in the overall hierarchy. We have confirmed that this is even more common for lower centrality core nodes too, and it is certainly even more true under weak selection.

4.6 Overhead of Incremental Design



(a) Penalty of Incremental Design

(b) Core Similarity

(c) Average Hierarchy Depth

(d) H-Score

Figure 4.16: Comparison between Incremental (INC) design and Clean-Slate (CS) design, in terms of four metrics and for different batch sizes.
For each batch size, the MRS-strong model is run for 5,000 iterations and an average of each metric is taken over 50 distinct iterations. The considered batch sizes are: 1, 5, 10, 25.

In this section, we compare the cost and structural characteristics of *Incremental design* (INC) relative to *Clean-Slate* (CS) design, i.e., the ideal case in which a new Lexis hierarchy

is designed from scratch every time the set of targets is changed. Of course such clean-slate designs are rare or infeasible in practice, especially in biological evolution. CS design is still valuable however as a baseline for evaluating the cost efficiency of INC, and the hierarchy that is produced by the latter.

In the Evo-Lexis framework, a key factor that quantifies the difference between INC and CS design is the *batch size*. If the batch size b is equal to the total number of targets in steady state T_s, INC and CS are equivalent because the set of targets completely changes in each iteration. At the other extreme, if the batch size is only one target and $T_s \gg 1$, INC performs a minimal adjustment of the hierarchy to support the new target while CS still redesigns the complete hierarchy. In other words, the fraction b/T_s controls the degree of change in each evolutionary iteration. Both in natural and technological systems, evolution proceeds rather slowly – for this reason we only consider the lower range of this ratio, between 1/100 and 25/100.

In the following we only consider the MRS-strong model (based on the results of the earlier sections). Fig. 4.16 compares INC and CS in terms of four key metrics. The first metric relates to cost: recall that the Penalty of Incremental Design (PID) is the ratio of the cost of an evolving INC hierarchy over the cost of the corresponding CS hierarchy for the same set of targets. With the exception of the minimum possible batch size (b=1), it is interesting that INC does *not* lead to much less efficient hierarchies than CS. The PID metric shows that INC is typically around 30% more costly than CS for a wide range of batch sizes, suggesting that INC is able to often reuse intermediate nodes in constructing the given targets, despite the fact that it cannot redesign the complete hierarchy. The PID is substantially higher when b=1 however. The reason is that when the INC-Lexis algorithm is given only one new target in every iteration, it is unlikely to identify segments of that single target that repeat more than once. Thus, when b=1, INC rarely adds new intermediate nodes in the hierarchy even though successive targets can be quite similar. CS, on the other hand, exploits the similarity of the set of targets in each iteration constructing more intermediate

nodes, and reducing cost through their reuse.

Interestingly, even when the INC and CS designs have similar costs, they are very different in terms of the nodes that form the core. This observation can be made in Fig. 4.16b: the similarity of the two cores according to the Levenshtein-Jaccard similarity is around 0.1. This result implies that the two design approaches lead to substantially different architectures in terms of the actual intermediate nodes they reuse.

Additionally, the average hierarchical depth of CS architectures is larger (see Fig. 4.16c) because this design approach is able to identify more and longer intermediate nodes that can be reused to construct the entire set of targets. INC, on the other hand, is constrained to not adjust the existing portion of the hierarchy, and it can only form new intermediate nodes when it detects fragments in the set of new targets that are repeated more than once. So, the INC hierarchies are typically not as deep as those in CS.

Despite their differences, both design approaches lead to hourglass architectures when the targets are created with the MRS-strong model. This observation can be made in Fig. 4.16d, and it suggests that even though INC is constrained, as described above, it is still able to identify few intermediate nodes that can be reused many times to construct the time-varying set of targets.

4.7 Discussion and Prior Work

The Evo-Lexis model is primarily related to three research themes: first, the emergence of modularity and hierarchy in complex systems; second, the hourglass architecture in hierarchical networks; and lastly, the comparison between offline (or "clean slate") design and online (or incremental) design.

4.7.1 Modularity and Hierarchy

The modeling framework of "Modularly Varying Goals", by Kashtan and Alon, is a plausible explanation for the emergence of modularity [73, 74]. By applying incremental changes

in logic circuits and evolving neural networks for pattern recognition tasks, they show that modularity in the goals (what we refer to as "targets") leads to the emergence of modularity in the organization of the system, whereas randomly varying goals do not lead to modular architectures. Similarly, Arthur et al. focus on the evolution of technology using a simple model of logic circuit gates [95]. Each designed element is a combination of simpler existing elements. Their simulation model results in a modularly organized system, in which complex functions are only possible by first creating simpler ones as building blocks. These models are similar to Evo-Lexis in the following way: when the system targets are not randomly constructed but they are generated through an evolutionary process that involves mutations, recombination and selection, the target functions are computed through deep hierarchies that reuse common intermediate components.

Clune et al. show that modularity is a key driver for the evolvability of complex systems [8]. The authors demonstrate that selection mechanisms that minimize the cost of connections between nodes in a networked system result in a modular architecture. This result is shown by evolving networks that solve pattern recognition tasks and Boolean logic tasks. The inputs sense the environment (e.g., pixels) and produce outputs in a feed-forward manner (e.g., the existence of patterns of interest). In other words, the networks that have evolved for optimizing both performance (accuracy in recognition) and cost (network connections) are more modular and evolvable (in the sense of being adaptable to new tasks) than those optimized for performance only. In a follow-up study by Mengistu et al. in [4], it is shown that the minimization of the cost of connections also promotes the evolution of hierarchy, the recursive composition of sub-modules. When not modeling the cost of connections, even for tasks with hierarchical structure (e.g., a nested boolean function), a hierarchical structure does not emerge. These modeling frameworks are similar to Evo-Lexis because the latter also aims to minimize the number of connections in the resulting hierarchical network, and it is this cost minimization that provides the incentive for reuse of intermediate components.

At the empirical side, prior work has established that technology evolves similarly to biological evolution, through tinkering, new combinations of existing components, and selection. For instance, a study of USPTO data gives evidence for the combinatorial evolution of technology [87]. The authors find that the rate of new technological capabilities is slowing down but a huge number of combinations allows for a "practically infinite space of technological configurations." By considering technology as a combinatorial process, [96] uses USPTO data to investigate the extent of novelty in patents. They propose a likelihood model for assessing the novelty of combinations of patent codes. Their results show that patents are becoming more conventional (rather than novel) with occasional novel combinations.

4.7.2 Hourglass Architecture

A property of many hierarchical networks is the *hourglass effect*, which means that the system receives many inputs and produces many outputs through a relatively small number of intermediate modules that are critical for the operation of the entire system [5]. This property is also one of the main themes investigated in our work.

Akhshabi et al. studied the *developmental hourglass* which is the pattern of increasing morphological divergence towards earlier and later embryonic development [81]. The authors conclude that the main factor that drives the emergence of the hourglass architecture in that context is that the developmental gene regulatory networks become increasingly more specific, and thus sparser, as developmemt progresses. Earlier, the same authors in [77] were inspired by the hourglass-resemblence of the Internet protocol stack in which the lower and higher layers tend to see frequent innovations, while the protocols at the waist of the hourglass appear to be "ossified." The authors present an abstract model, called *EvoArch*, to explain the survival of popular protocols at the waist of the protocol stack. The protocols which provide the same functionality in each layer compete with each other and, just as in Akhshabi et al. [81], the increasing specificity and sparsity is what causes the

network to have an hourglass architecture. The Evo-Lexis model is neither layered, nor probabilistic, and so it is fundamentally different than *EvoArch*, but it also generates hierarchies in which the nodes that represent shorter strings (equivalent to lower-layer nodes in EvoArch) are reused more frequently and so they have a higher out-degree.

Friedlander et al. focus on layered networks that perform a linear input-output transformation [82] and show that in such systems the hourglass architecture emerges when that transformation is compressible. In their model, this concept is interpreted as rank-deficiency of the input-output matrix that describes the function of the system. A further requirement is that there should be a goal to reduce the number of connections in the network, similar to Evo-Lexis. This rank-deficiency in the input-output matrix resembles the case in which Evo-Lexis targets are not constructed independently but through an evolutionary process that generates significant correlations between different targets.

The hourglass architecture has been also investigated in general (non-layered) hierarchical dependency networks, similar to Evo-Lexis, by Sabrin et al. [5]. That analysis is based on identifying the core of a dependency network, as the minimum set of nodes that cover at least a fraction τ of all source-to-target dependency paths. We have adopted that approach, as well as the hourglass metric proposed in Sabrin et al. [5]. Their study shows the presence of the hourglass property in various technological, natural and information systems. The authors also present a model called *Reuse-Preference*, capturing the bias of new modules to reuse intermediate modules of similar complexity instead of connecting directly to sources or low complexity modules.

Despite this prior work, the interplay between the emergence of hourglass architectures and cost optimization in hierarchical networks has not been explored in previous research. Evo-Lexis identifies the conditions under which the hourglass property emerges in optimized dependency networks.

4.7.3 Interplay of Design Adaptation and Evolution

A main theme in our study is the interplay between changes in the environment (the targets that the system has to support) and the internal architecture of the system.

Bakhshi et al. investigate a network topology design scenario in which the goal is to design a valid communication network between a set of nodes [97]. The authors formulate and compare the consequences of two different optimization scenarios for that goal: *incremental design* in which the modification cost between the two last snapshots of the design is minimized, and *optimized design* in which the total cost of the network is minimized in every increment. Focusing on the case of ring networks, even though the incremental designs are more costly, the relative cost overhead is shown to not increase as the network grows. In a follow-up study, focused on mesh networks, the same observation is made and further, the incremental design is shown to be producing larger density, lower average delay and more robust topologies [98].

Incremental design approaches are also considered in other contexts, such as in deep neural networks (DNNs). Specifically, an important problem in machine learning is how to transfer learned features of a deep network from one task to another [99]. Transfer learning can be considered analogous to the way in which new targets are added in an Evo-Lexis hierarchy: new targets (output functions) are incrementally included in the Lexis-DAG (incrementally learned), by re-using previously constructed intermediate nodes (features of intermediate complexity) and then optimizing the part of the DAG between those nodes and the new targets (learning the weights between the existing features and the new outputs).

The incremental design policies that we consider in this chapter are studied in computer science under the umbrella of *online algorithms* [100]: an online algorithm finds a sequence of solutions based on the inputs it has seen so far, without knowing the entire input sequence in advance. The main emphasis of research in online algorithms is to perform *competitive analysis*, i.e., to derive worst-case theoretical bounds between of the quality (or cost) of the solution of an online algorithm relative to its offline counterpart that knows the entire input

sequence [101]. The Incremental Design approach in Evo-Lexis is an online algorithm but our focus is quite different: we compare empirically the cost and topological structure of the hierarchies produced by incremental design relative to an optimized ("clean-slate") algorithm that designs a minimum-cost hierarchy for the input sequence that has been seen so far.

4.8 Conclusion

We presented Evo-Lexis, an evolutionary framework for modeling the interdependency between an incrementally designed hierarchy and a time-varying set of output functions, or targets, constructed by that hierarchy. We leveraged the Lexis optimization framework, proposed in earlier work [21], which allows the design of an optimized hierarchical network for a given set of sequences.

We developed the optimization framework, evolutionary target generation processes, and evaluation metrics needed to study the emergence and evolution of optimized hierarchies. We summarize the results of our study as follows:

1. Tinkering/mutation in the target generation process is found to be a strong initial force for the emergence of low-cost and deep hierarchies. The presence of selection, however, intensifies these properties of the emergent hierarchies.

2. Selection is also found to enhance the emergence of more complex intermediate modules in optimized hierarchies. The bias towards reuse of complex modules results in an hourglass architecture in which almost all source-to-target dependency paths traverse a small set of intermediate modules.

3. The addition of recombination in the target generation process is essential in providing target diversity in optimized hierarchies.

4. Hourglass-shaped optimized hierarchies are found to be fragile if the core nodes (i.e.,

nodes with highest centrality) are perturbed, similar to the concept of removal of hub nodes in scale-free networks.

5. We show that an hourglass architecture introduces a trade-off between the cost of introducing new targets and the diversity between selected targets: hourglass architectures are evolvable in the sense that they allow the introduction of new targets at a low cost but they only explore a small part of the "phenotypic space" of all possible targets. These are targets that can be constructed at a low cost reusing the larger intermediate modules in the hierarchy.

6. Our results suggest the existence of major transitions and punctuated equilibria in the evolutionary trajectory of hourglass-shaped hierarchies. The "extinction" of central modules is found to be the main factor behind this effect.

7. The comparison between incremental design and clean-slate shows that although the former is much more constrained, it has similar cost and it also exhibits the hourglass effect under the proposed evolutionary scenarios. Despite these similarities, each of these design policies results in a very different set of core modules.

CHAPTER 5

A CASE STUDY: ANALYSIS OF IGEM SYNTHETIC BIOLOGY SEQUENCES

5.1 Introduction

The Evo-Lexis model is quite general and abstract, and it does not attempt to capture any domain-specific aspects of biological or technological evolution. As such, it makes several assumptions that can be criticized for being unrealistic, such as the fact that all targets have the same length, or their length stays constant, or the fitness of a sequence is strictly based on its hierarchical cost. We believe that such abstract modeling is still valuable because it can provide insights into the qualitative properties of the resulting hierarchies under different target generation models. However, we also believe that the predictions of the Evo-Lexis model should be tested using real data from evolving systems in which the outputs can be well represented by sequences.

One such system is the iGEM synthetic DNA dataset [1]. The target DNA sequences in the iGEM dataset are built from standard "BioBrick parts" (more elementary DNA sequences) that collectively form a library of synthetic DNA sequences. These sequences are submitted to the registry of standard biological parts in the annual iGEM competition. Previous research in [15, 21] has provided some evidence that these synthetic DNA sequences are designed by reusing existing components, and as such, it has a hierarchical organization. In this chapter, we investigate how to apply the Evo-Lexis framework in the time series of iGEM sequences, and whether the resulting iGEM hierarchies exhibit the same qualitative properties we observed in Chapter 4 which was solely based on abstract target generation models.

The questions we try to answer in this chapter are:

1. How can we analyze the iGEM dataset using the evolutionary framework of Evo-

Lexis? How are the batches of targets formed? What properties of the iGEM batches are different than Evo-Lexis's setting?

2. When formed incrementally over the iGEM dataset, which are the architectural properties of Lexis-DAGs, and why?

    - Are they cost-efficient? Are they deep hierarchies?
    - Do they present small cores, i.e., the hourglass effect? What causes them to have this property?
    - Is the core set stable in these DAGs? Do major transitions in the core set happen during the time period in which iGEM has been evolving?

5.2 Dataset

5.2.1 Preliminaries

The International Genetically Engineered Machine (iGEM) is an annual worldwide synthetic biology competition. The competition is between students from diverse backgrounds including biology, chemistry, physics, engineering, and computer science to construct synthetic DNA structures with novel functionalities.

Figure 5.1: The logo of iGEM competition [1]

Every year at the beginning of the summer, there is a "Distribution Kit" handed to teams which includes interchangeable parts (so called "BioBricks") from the Registry of Standard Biological Parts comprising various genetic components such as promoters, terminators, reporter elements, and plasmid backbones. Then, the teams try to use these parts and the new standardized parts of their own in order to build biological systems. Then, they will have them operational in living cells.

The teams can build on previous projects or create completely new parts. At the end of the summer, all teams add their new BioBricks to the registry for further possible reuse in the next year. In each year, the submitted parts are judged based on the quality of synthesis and other rubrics (e.g., for 2018 the rubrics can be found here: `http://2018.igem.org/Judging/Rubric`). According to the iGEM portal, "successful projects produce cells that exhibit new and unusual properties by engineering sets of multiple genes together with mechanisms to regulate their expression".

The iGEM Registry (i.e., the dataset we are working with) includes a set of standard biological parts. A [biological] part is a DNA sequence which encodes a biological function, e.g., a promoter or protein coding sequence. These biological parts are standardized to be easily assembled together and reused with other standardized parts in the registry. A "basic part" is a functional unit of a synthesized DNA that cannot be subdivided into smaller component parts. BBa_R0051 is an example of a promoter basic part. Basic parts have the role of sources in the Lexis setting. A "composite part" is a functional unit of DNA consisting of two or more basic parts assembled together. BBa_I13507 is an example of a composite part, consisting of four basic parts "BBa_B0034 BBa_E1010 BBa_B0010 BBa_B0012". The corresponding webpage for BBa_I13507 shows basic information about the part, such as its history, composition, reuses in other parts and the actual sequence of the part (Fig. 5.2).

The dataset we analyze is the set of all composite parts submitted to the registry from 2003 to 2017. In this dataset, the composite parts are represented by the string of their basic parts (i.e., a non-dividing representation). The sequence of iGEM composite parts can be considered as a sequence of target strings over a set of sources (i.e., basic parts).

5.2.2 Data Collection

We have acquired the iGEM data from `https://github.com/biohubx/igem-data`. All the BioBrick parts were crawled until Dec 28th 2017. We extracted the ".info" files for

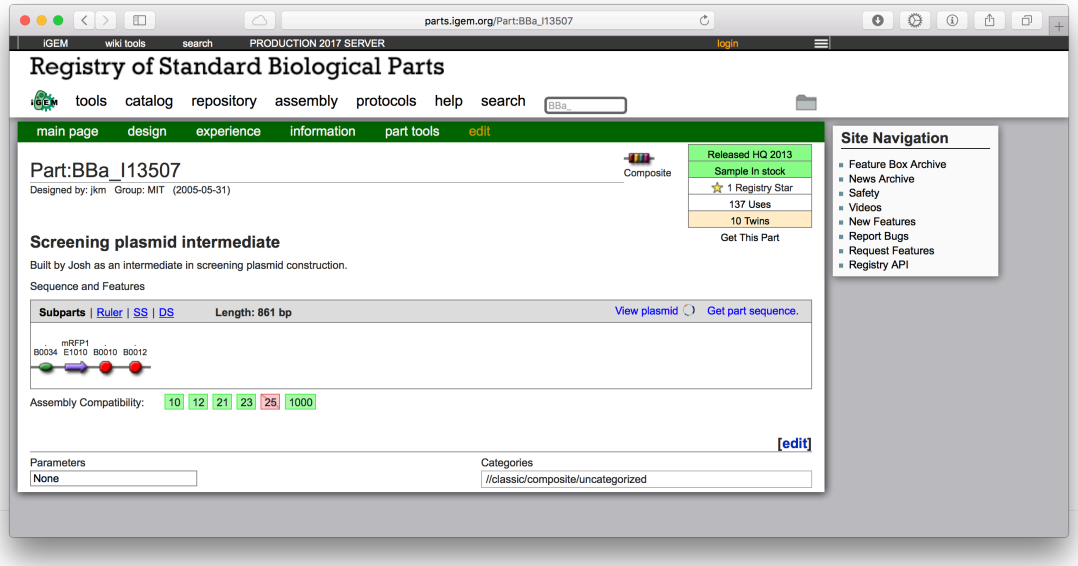


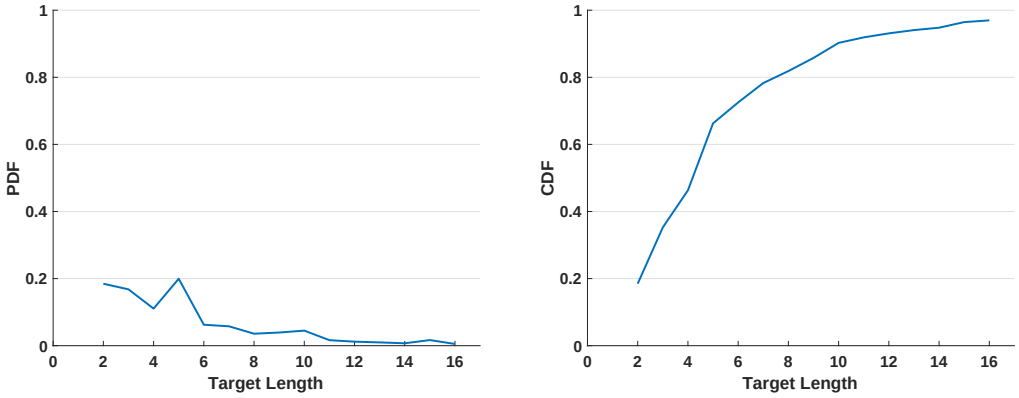
Figure 5.2: Screenshot of the webpage for BBa_I13507.
It shows basic information about the part, such as its history, its composition, its reuses in other parts and the actual sequence of the part.

each part, then used the "subparts" field for forming the target strings and finally, used the year in "creation_date" field for ordering the targets.

The targets are presented in the span of the years 2003 to 2017. In Table 5.1 and Fig.5.3, the preliminary statistics about the dataset can be found.

Table 5.1: Basic statistics on iGEM dataset during 15 years (2003-2017)

# Sources	# Targets	Total Length	Min/Max Target Length
7,889	18,394	107,022	2 / 100



(a) PDF (b) CDF

Figure 5.3: PDF and CDF of target lengths (2003-2017)

As observed in Fig. 5.3, the dataset mostly presents targets of small length. The top 5 categories having the highest fraction of the targets belongs to those of length 5, 2, 3, 4 and 6, accounting for more than 70% of the dataset. Less than 10% of the targets have a length of more than 10.

5.2.3 Considering Annual Batches of Targets

The iGEM competition is conducted annually. Hence, it is reasonable to consider the sequences of targets as annual batches of targets arriving each year. This consideration is in line with the incremental design process in Evo-Lexis.

To show some differences between iGEM and Evo-Lexis, in Fig. 5.4 and 5.5, we can see how the number of sources, the number of targets, the total length of the targets, length statistics and source reuse statistics change over time. We can make the following observations by looking at these figures:

1. The number of sources increases over time (while it was constant in Evo-Lexis).

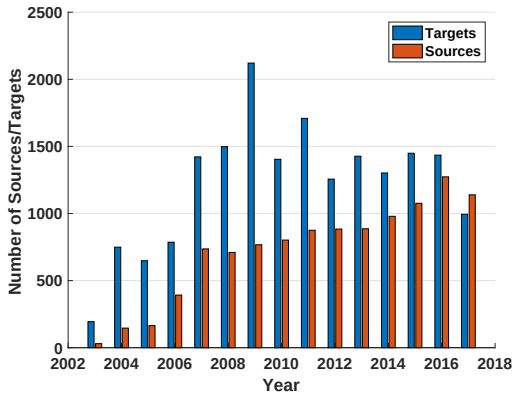
2. In the first four years, the number of targets per year is noticeably small. Later on, the number of targets increases up to 2,000 and then fluctuates around 1,000 to 1,300

targets per year. The total length of the targets follows a similar trend as the number of targets. In Evo-Lexis, the number of targets per batch is constant and they all have the same length.

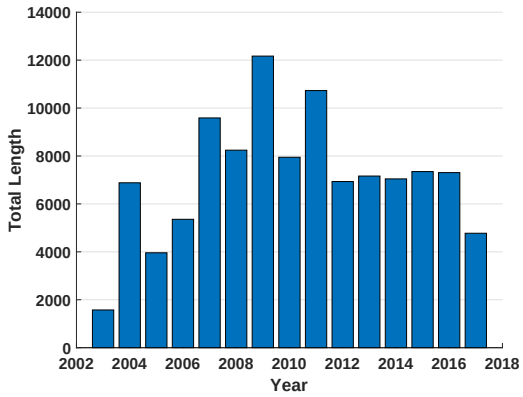
3. The mean and median of target lengths stay in the same range ($\in [5, 7]$) during all 15 years.

4. The reuse of sources (except for the beginning years) is extremely skewed in all years: few sources are used much more often than most of the sources (Fig. 5.5). In Evo-Lexis, all sources are equally likely.

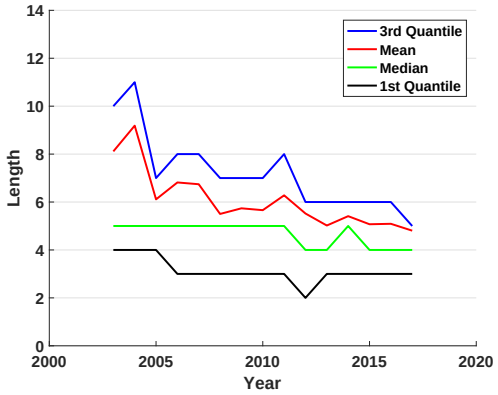
In the following sections, we show that how these differences between iGEM dataset and Evo-Lexis cause differences between the resulting Lexis-DAG hierarchies.



(a) Number of sources and targets per year



(b) Total length of targets per year



(c) Aggregate statistics of length of targets per year

Figure 5.4: Statistics of iGEM dataset when considered as yearly batches

5.3 Analysis of iGEM Dataset in Evo-Lexis Framework

5.3.1 Lexis-DAG Cost Analysis

In this section, we observe how cost efficient the Lexis-DAGs over the iGEM dataset are. We consider an incremental setting similar to Evo-Lexis: In the first year, a clean-slate Lexis-DAG is constructed over the targets of that year. For the targets of the subsequent years, an incremental Lexis-DAG is constructed. Fig. 5.6 shows how the normalized cost of the Lexis-DAGs varies over the years on iGEM. We observe major differences with

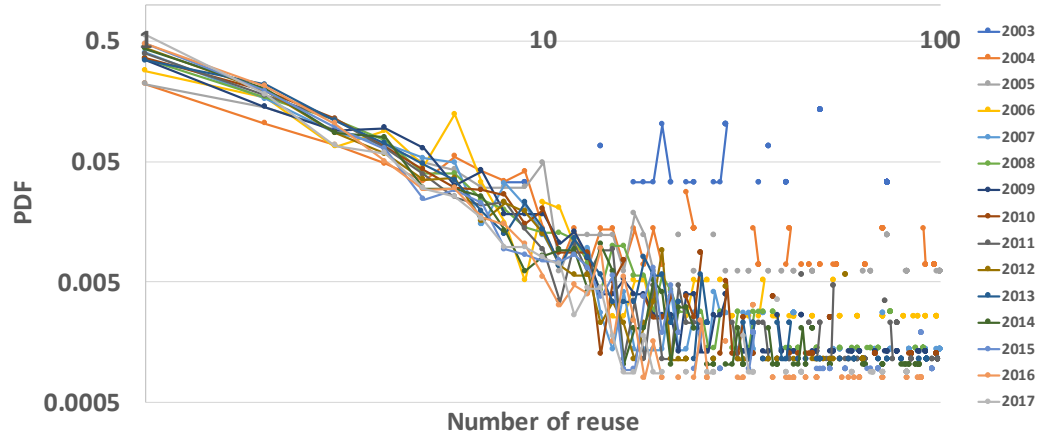


Figure 5.5: PDF of reuse of the sources per year .
Number of reuse is the number of times a source appear in a target in each year.

Evo-Lexis; in Evo-Lexis the normalized cost remains almost constant.

To investigate the reasons for the above observations, in the same Fig. 5.6, we also track the cost reduction performance of the two stages of INC-LEXIS for each batch (as a reminder, in stage-1, we reuse intermediate nodes from previous Lexis-DAG and in stage-2, we further optimize the hierarchy using G-LEXIS). This experiment is done due to our interest in seeing how much stage-1 of INC-LEXIS contributes to the cost reduction on iGEM. There are two observations that we can make:

1. In most batches, more than 50% of the cost reduction is achieved by the stage-1, i.e., reuse stage. The contribution of stage-2 of INC-LEXIS is roughly constant throughout years. This suggests that iGEM targets reuse a significant amount of sequences from previous years in their own submissions.

2. There is an increasing trend in the normalized cost after stage-1. This observation means that the contribution of the reuse stage in INC-LEXIS decreases over the years. As mentioned, the contribution of stage-2 stays mostly constant. Hence, we can relate the increasing trend of the normalized cost to the fact that the amount of reuse reduces from year to year.

We can find the root-cause of the decrease of reuse over time on iGEM to the increase

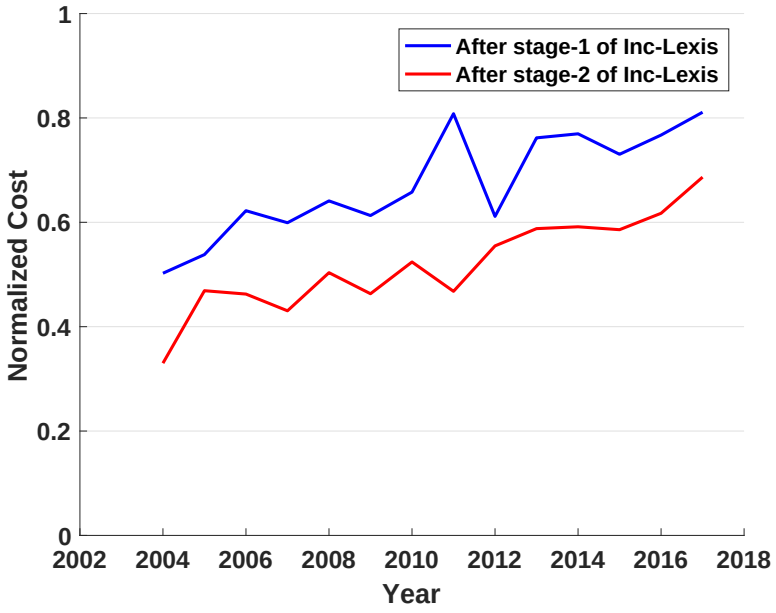


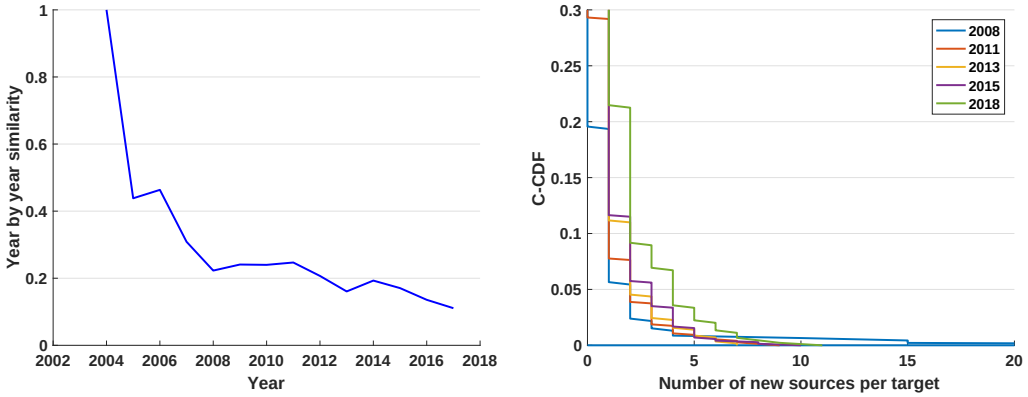
Figure 5.6: The cost reduction performance of the two stages of INC-LEXIS for each batch (i.e., year)

of the size of the set of sources. We have observed in Fig. 5.4a that there are many new sources that get introduced over the years. One of the requirements for reuse from one batch to another in Evo-Lexis is the fact that the set of sources does not drastically change (in fact it is constant in the Evo-Lexis framework). To investigate whether this is true in iGEM, we check the ratio of the sources from one year to the next that remain the same. Specifically, if we have $y_2 = y_1 + 1$, and if S_{y_1} & S_{y_2} are the set of sources in year y_1 & y_2 respectively, we check the ratio below:

$$\frac{|S_{y_1} \cap S_{y_2}|}{|S_{y_1}|} \tag{5.1}$$

This ratio, i.e., *year-by-year similarity*, is the fraction of sources that remain from the previous year. Fig. 5.7a shows how this ratio changes from year to year. By year 2008, the ratio drops significantly to a value around 0.2 which means around 80% of the sources from the previous year are not reused. This reduces the amount of reuse that is possible in

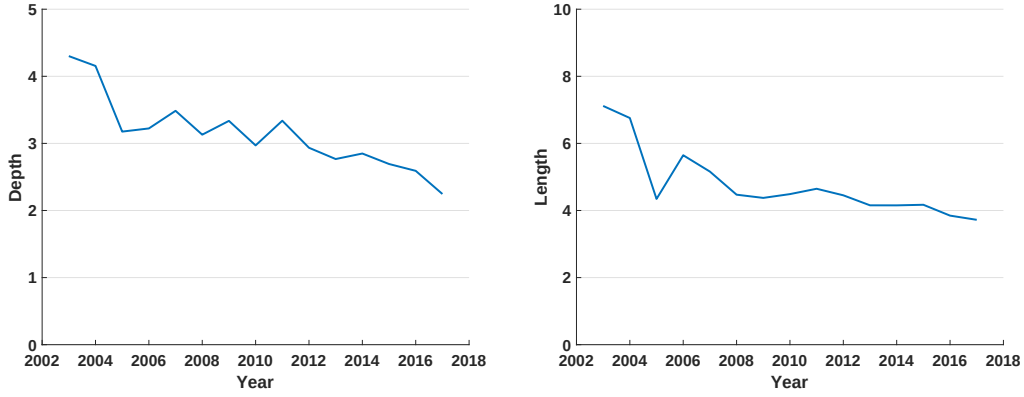
the iGEM dataset. The introduction of new sources is also propagated in individual targets. As Fig. 5.7b shows, we can observe that there is an increase in using new sources per target in each year. As time progresses, there is a higher probability to use more than X number of new sources per target. This observation is a further obstacle for reuse, especially given that the targets in iGEM are often short (5-7 subparts).



(a) Source set similarity between years (b) C-CDF of new sources reuse per target

Figure 5.7: Analysis of the source set of iGEM targets over years

Following the increase of the normalized cost, Fig. 5.8 shows that the DAGs get less deep and have lower average node length as time progresses.



(a) Depth (b) Average Node Length

Figure 5.8: Depth and average node length in iGEM Lexis-DAGs

Overall, the results of this section show a number of differences between iGEM and the Evo-Lexis framework of Chapter 4:

1. In iGEM, the set of sources in each year has low similarity to the previous years, while in Evo-Lexis the source set is constant. The high amount of churn in the set of sources is the primary reason for the lower reuse in iGEM data compared to Evo-Lexis. The fact that the targets are shorter is another contributing factor to a lower potential for reuse of longer intermediate nodes in iGEM.

2. The normalized cost, depth and average node length are all lower in iGEM due to the reduced reuse potential as discussed above.

5.3.2 Hourglass Effect in iGEM

The output of core identification (sequence of most central nodes removed from a Lexis-DAG, see Section 2.4) gives us a good measure of how many intermediate nodes are heavily reused in iGEM. Fig. 5.9 shows the fraction of paths that are covered by the core nodes in each year. This figure shows that in all years, there is a small number of core nodes in the iGEM Lexis-DAGs.

Fig. 5.10 shows that such a small core makes the topology of Lexis-DAGs consistent with an hourglass organization (high H-score values - more than 0.6 in Fig. 5.10c). In Evo-Lexis, we observe similar values of H-score for DAGs constructed using synthetic data. As observed, although the core size increases in iGEM over time, we see a steeper increase in the size of the flat DAG's core mostly due to the increase in set of sources. In Evo-Lexis, the core size shows a decreasing trend while the size of the core of the flat DAG does not significantly change, reflecting similarly high H-score values as in iGEM.

Overall, we can see that the topology of the Lexis-DAGs in iGEM data is in line with the Evo-Lexis model, although the bias in selection of cost-saving nodes is not sufficiently large to cause a non-increasing normalized cost.

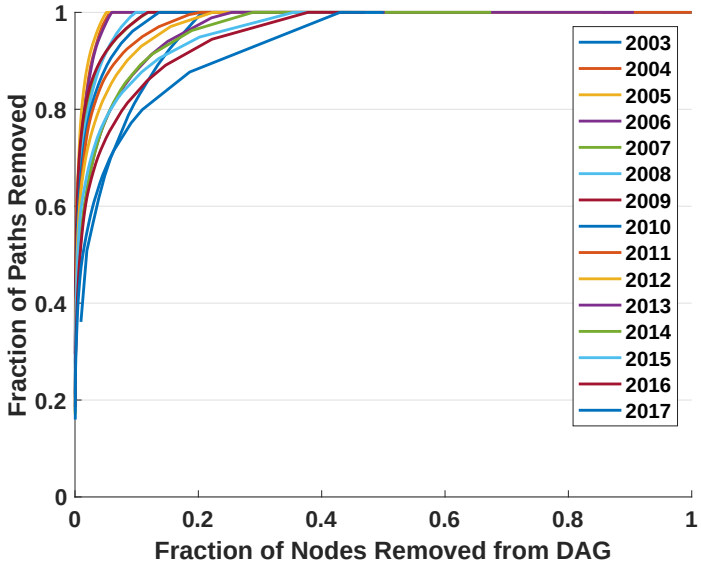


Figure 5.9: Cumulative fraction of paths covered by core nodes in each year of iGEM data.

5.3.3 Diversity among iGEM Targets

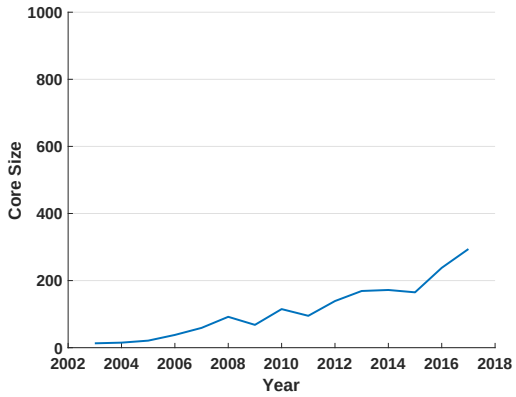
Another question is the degree of diversity among the targets of iGEM over time. Since the length of the targets in iGEM is not constant, we need to modify the definition of the diversity metric in Chapter 4 (Eq. (4.15)). We define the concept of *Normalized Diversity* as follows:

Suppose we have a set of strings $T = \{t_1, t_2, ..., t_n\}$. The goal is to provide a single number that quantifies how dissimilar these elements are to each other.

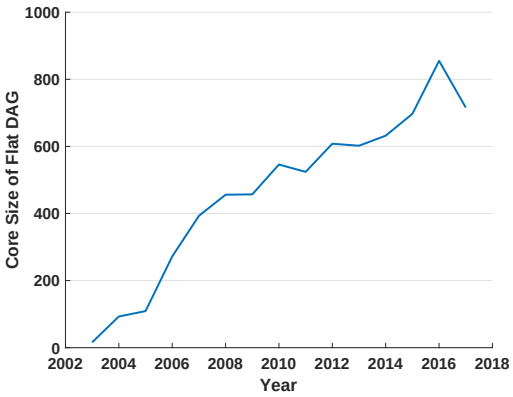
- We first identify the *medoid* $\mathcal{M}_T$ of the set T, i.e., the element that has the lowest average distance from all other elements. We use Levenshtein distance as a measure of distance between targets:

$$\mathcal{M}_T = arg\ min_{m \in T} \sum_{t \in T} LD(t, m) \tag{5.2}$$

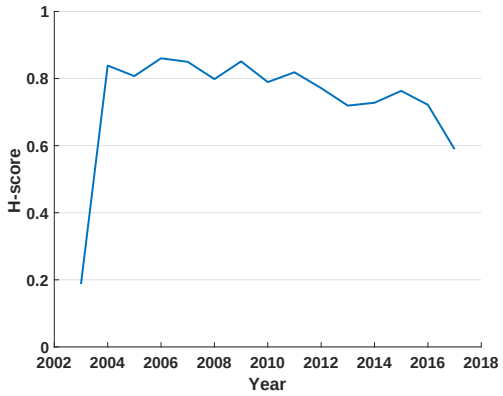
- To compute how diverse the elements are with respect to each other, we average



(a) Core Size



(b) Flat Lexis-DAG Core Size



(c) H-score

Figure 5.10: Core size and H-score in iGEM data over time ($\tau = 0.85$ for core identification)

the normalized distance of all elements from the medoid (distance is normalized by the maximum length of the two sequences in question). We call this measure σ_T, the *Normalized Diversity* of set T. The bigger the metric, the more diverse a set of strings is (because the distance of each target from the medoid can be considered as the number of single-character operations needed to convert any element within the set to the medoid):

$$\sigma_T = \frac{\sum_{t \in T} \frac{LD[t, \mathcal{M}_T]}{max(|t|, |\mathcal{M}_T|)}}{|T|} \tag{5.3}$$

Notice the difference of this metric with the diversity metric in Eq. (4.15).

Fig. 5.11 shows that the normalized diversity metric has a value of more than 0.5 throughout time and reaches up to 0.8 (this means that on average 50% to 80% of a target should be changed so that a target is converted to another in the set of targets in each year). Although such values of diversity are in line with Evo-Lexis, it is understandable that the diversity in iGEM is also partially impacted (towards higher values) by the introduction of new sources discussed in Section 5.3.1.

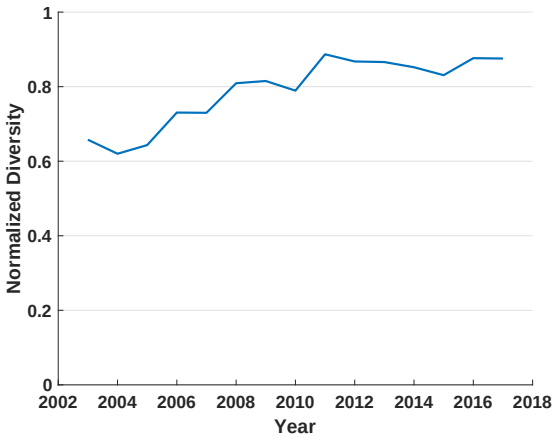


Figure 5.11: Target diversity in iGEM data over years.

5.3.4 Core Stability in iGEM Lexis-DAGs

As the results in Fig. 5.12 show, the core set in iGEM DAGs have relatively high values of the core stability measure (Eq. (4.11)), close to the values we observed in Evo-Lexis. This means that the core nodes stay similar across time, and there are no sudden changes in the content of the core set. One reason for this stability is that the set of core nodes includes several sources, and many of core sources get transferred to the next year. Fig. 5.13 shows the length and path centrality of the core nodes among all nodes of the Lexis-DAGs in selected years. We observe that more than 50% of the core nodes in the selected years are sources.

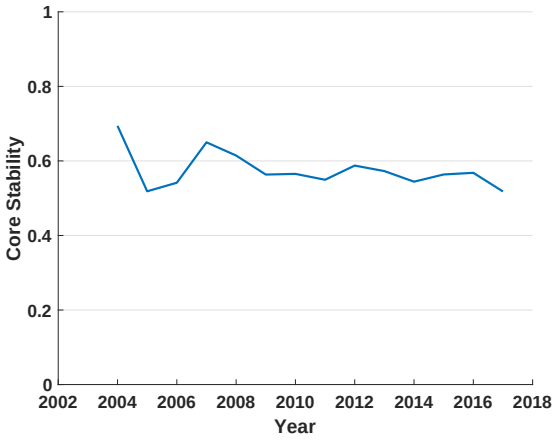
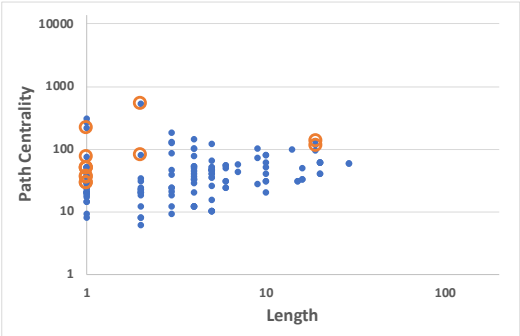


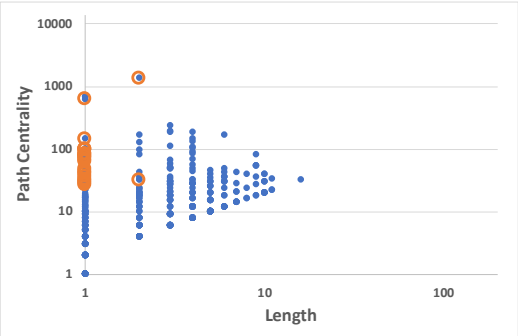
Figure 5.12: Core stability over iGEM dataset.

Additionally, every year the focus of the iGEM designers is on specific parts, most of which are of high path centrality. For example, "BBa_B0010 BBa_B0012" (the most widely used "terminator" part) and "BBa_B0034" are almost always the top-2 central nodes (with the exception of year 2011). Also, some sources such as "BBa_R0011", always appear in the top-20 nodes in the core set. Remember that Fig. 5.5 shows that the reuse distribution of sources is highly skewed. To further quantify the stability of higher centrality nodes, we investigate the stability of the top central core nodes. As a final note, in terms of the analysis for major transitions that we did in Section 4.5, the DAGs do not show a major transition in iGEM. We can relate this to the short time frame that iGEM has been around in comparison to the setting of Evo-Lexis. Specifically, we used the same core stability metric and measured the stability of the top-20 nodes, i.e., *central set 1-20*, and the next 20 central nodes (in core identification, the nodes ranked from 21st to 40th), i.e., *central set 21-40*. The median stability for central set 1-20 and central set 21-40 are 0.707 and 0.248, respectively. Overall, the higher centrality core nodes over the years remain stable and similar to previous years in iGEM data despite the fact that the set of sources changes significantly. This also explains why the size of the core size does not increase drastically in Fig. 5.10a (as opposed to the size of the core of the flat Lexis-DAG in Fig. 5.10b which is a

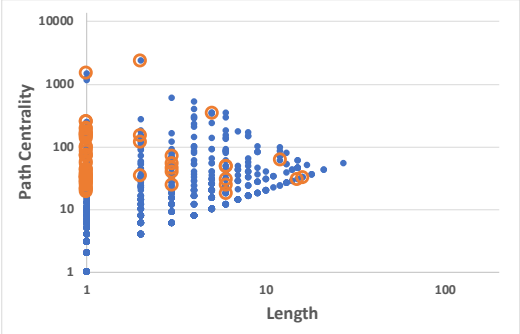
result on the expanding set of sources). In summary, the stability of the core set in iGEM is caused by the same reason with Evo-Lexis, which is the bias and selectivity towards using a specific set of nodes in consecutive years.



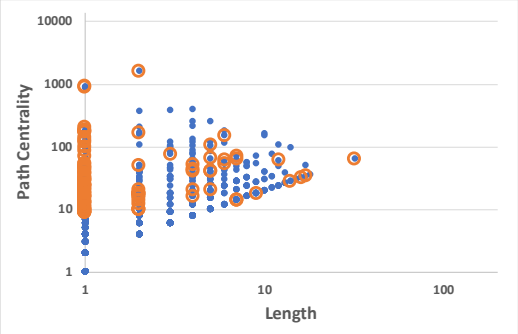
(a) Year 2003



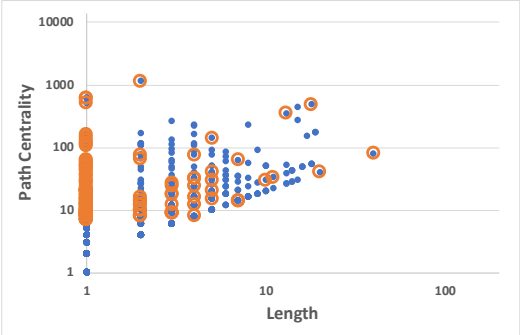
(b) Year 2005



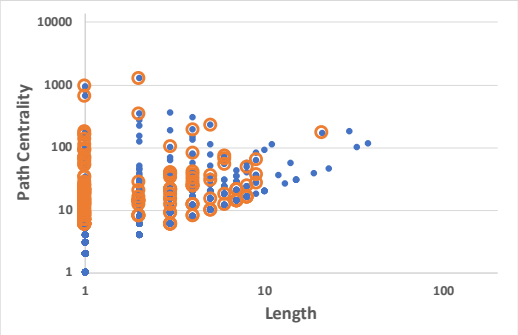
(c) Year 2008



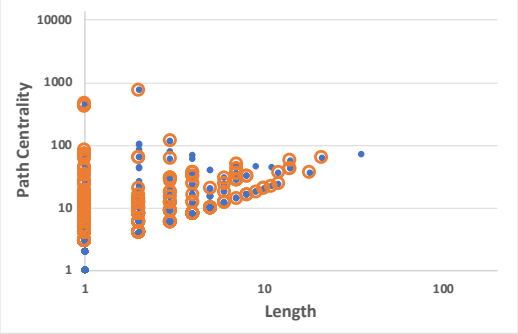
(d) Year 2010



(e) Year 2013



(f) Year 2015



(g) Year 2017

Figure 5.13: Node length and path-centrality in selected years in iGEM dataset. For core selection, we set $\tau = 0.85$. This figure shows that the core sets include a high number of sources.

5.4 Conclusions

iGEM is a dataset that satisfies the basic assumption of Evo-Lexis framework: a sequence of target strings with potential temporal reuse of previously introduced substrings. Because of this compatibility, we chose to use this dataset in a case-study and contrast its qualitative properties with Evo-Lexis. We can summarize the answers to the questions posed in the beginning of this chapter as follows:

- We observe that although incremental design can build efficient hierarchies over the iGEM targets, the normalized cost increases over time. This is due to the fact that the amount of reuse from previous years decreases mainly due to the frequent introduction of new sources over time. The small length of the targets in iGEM is also an additional factor for lowering the potential of reuse of the previously constructed parts in iGEM.

- The increasing normalized cost causes the Lexis-DAGs to become less deep and to contain shorter nodes on average as time progresses. This is different than Evo-Lexis. In addition, there is a high fraction of very short targets in each year in comparison to Evo-Lexis.

- Another important observation in iGEM data is that the Lexis-DAGs present a bias in reusing specific nodes more often than the other nodes. This biased reuse results in the Lexis-DAGs to take the shape of an hourglass with relatively high H-score values and a stable set of core nodes over time. This observation is consistent with the Evo-Lexis modeling results.

- The core sets over the years remain stable and similar to previous years in iGEM data despite the fact that the set of sources changes significantly. Most of the stability is contributed by a small set of central sources and central intermediate nodes that are heavily reused throughout the iGEM registry over time.

- Despite the reuse bias and stability of the core sets, the dataset presents a high degree of diversity among the targets in each year, which is in line with the Evo-Lexis modeling results.
- In the limited temporal span of the iGEM dataset, we do not observe major transitions in the core sets.

CHAPTER 6
LIMITATIONS AND EXTENSIONS

6.1 Limitations

6.1.1 Data Considerations

Lexis, as currently formulated, is only considering strings as the main objects. Also, the framework's specification only considers the concatenation of substrings. However, strings can be the result of many other operators such as those used in measuring Levenshtein distance (e.g., letter removals, substitutions, additions etc.). This limitation implies that generation and modeling of some parts of a dataset may not be captured by Lexis. Furthermore, considering only concatenations makes Lexis more prone to noise in the data.

6.1.2 Further Characterization of Evolutionary Mechanisms

Our modeling allowed us to identify a number of mechanisms within the settings of Evo-Lexis that are sufficient to produce the presented results in this thesis (e.g., the way selection is responsible for hourglass organization). However, it is important to note that we can not be certain whether these mechanisms are also necessary for these qualitative results. We believe that most likely they are not necessary mechanisms (e.g., we can probably come up with other mechanisms for generating an hourglass architecture).

6.1.3 Parameter Space Analysis

By design, there is a large parameter space considered for Evo-Lexis simulation (e.g., length of targets, number of targets, etc.). We have already tested empirically that the presented results do not vary significantly with random initialization (i.e., via testing each result with multiple runs). However, systematic exploration of the Evo-Lexis parameter

space is difficult, and it is not clear with high accuracy what the boundaries of this space are and how the qualitative results of Evo-Lexis are sensitive to different choices of the parameters.

6.1.4 Heuristic Algorithm Design

The Lexis optimization problem and the core identification problem are NP-Hard problems for which we have designed greedy heuristics. While these algorithms serve the purpose of the studies in this research, this thesis does not cover an analysis of the approximation ratio or possibilities for improvement on the runtime of these algorithms.

6.1.5 Additional Mechanisms in Evo-Lexis

Evo-Lexis mainly considers widely accepted mechnisms, namely mutation, recombination and selection as the "forces" on evolution of hierarchies. However, there are many mechanisms that are not modeled by Evo-Lexis such as invention of new sources, expansion or reduction in the number of targets, competition effects between different architectures (hierarchies), etc.

6.2 Extensions

6.2.1 Noisy Data and Approximate-Lexis

In practice, symbolic sequences are always noisy. For instance, a symbolic representation may be "abcde" in one instance, "abbcde" in another instance, and "abced" in a third instance. Lexis, as formulated in this thesis, cannot handle such noise because it lacks the generalization capability to identify that all three sequences are very similar. One can propose to extend the basic Lexis framework so that it can successfully perform such generalizations – we refer to this framework as *Approximate-Lexis* or *A-Lexis*.

We first need a metric for quantifying the similarity between two symbolic sequences. The Levenshtein distance $LD(s,t)$ between two strings s and t is the number of deletions,

insertions, or substitutions required to transform one string to another. The higher the number of required operations, the more distant two strings are from each other [89]. The Levenshtein distance can be normalized by the length of the longer string so that it is always between 0 and 1. Based on a user-specified threshold μ for this metric, we can then determine whether two sequences are sufficiently similar to each other or not.

The next step is to generalize the semantics of vertices and edges in the Lexis-DAG. In the A-Lexis framework, a vertex does not represent a single sequence – instead it represents the centroid of a set of sequences, all of which are within μ from the centroid in terms of Levenstein distance. In other words, each vertex represents a cluster of similar sequences.

Suppose now that that we often see sequences of the form $A_1A_2A_3$, where each A_i can be any sub-sequence in the cluster with centroid A_i. In the A-Lexis DAG, this centroid can be represented with a new vertex, say B, with label $B = A_1A_2A_3$, i.e., the concatenation of the three centroid sequences. B however is also the centroid of a new cluster, capturing any variations that are within μ from the sequence $A_1A_2A_3$. For instance, if the threshold μ allows for single-character "mutations", the vertex B would also capture the sequence $A_1zA_2A_3$ (where z is an individual character).

A major task in future research will be to further develop these preliminary ideas and create efficient and robust algorithms for the construction of A-Lexis DAGs as well as for the analysis of these hierarchies.

6.2.2 Scalable Methods for Hierarchy Inference

The Lexis optimization problem is NP-hard, hence heuristic approaches need to be employed. We have developed a greedy algorithm, called G-LEXIS, which starts from the trivial Lexis-DAG with edges from the alphabet source nodes to each of their occurrences in the target strings, and iteratively searches for the substring that leads to the maximum cost reduction when added as a new intermediate node in the Lexis-DAG. Even with efficient suffix tree data structures, G-LEXIS has quadratic run-time complexity $O(L^2)$, which

is prohibitive for very large datasets.

Faster algorithms can be designed, but they perform worse in terms of cost minimization. In particular, we can leverage the algorithms that have been previously designed for the Smallest Grammar Problem because it is easy to map a straight-line grammar to a Lexis-DAG. For instance, Galle et al. have proposed two heuristics (namely REPAIR and Longest-First-Substitution) that have a linear-time complexity for inferring a straight-line grammar, but they perform much worse than SGP algorithms with quadratic run-time complexity [19]. An open research question that we will focus on is: Can we design an algorithm that runs in faster than quadratic-time (e.g., $O(n \log n)$), while performing almost the same in terms of cost as quadratic-time algorithms?

In future research, one can investigate how to parallelize the data structures (such as suffix arrays) used in the Lexis inference algorithms. Specifically, the research in [102] presents parallel algorithms for distributed memory construction of suffix arrays and longest common prefix (LCP) arrays. A detailed technical survey on parallel suffix arrays can be found at `http://snnynhr.github.io/ParallelSuffixArrays/`.

6.2.3 Other Application Domains for Lexis

In the future, we are aiming to provide broader application contexts for the developed methods in this thesis, especially in the area of analysis of time series for rule discovery and anomaly detection. Recent research shows promising results on structure discovery over discrete representation of time series data, such as SAX [103, 104, 105]. Further, the hierarchical structure is shown to be inherent in such data. What needs to be noted is that there are a few number of works in the literature that focus on hierarchical representations of time series data, and they concern anomaly discovery [103], feature extraction for time series classification [104], and analysis of spatial trajectory data [105]. All of these references rely on, first, symbolic discretization to detect patterns, and second, inference of pattern relationships using SEQUITUR-inspired algorithms [10]. The result is a grammar

representation of the original sequence, which offers insights into its lexical structure [10]. Although a grammar can be represented as a hierarchy of grammar rules, none of these papers exploits such a hierarchical structure to analyze it from a graph mining perspective. A major difference between Lexis and grammar-based methods is that Lexis produces a network representation (a DAG) of the given sequences instead of a grammar. A network representation is preferable because it allows us to apply a large toolbox of graph mining techniques on the discovered network structure, such as various vertex or edge centrality metrics to identify the most important components of the hierarchy or core-periphery analysis methods to identify subsets of intermediate nodes that can capture collectively almost all of the source-to-target (or input-output) relationships [5, 106].

6.2.4 More Realistic Extensions of Evo-Lexis Framework

As mentioned, several assumptions made in Evo-Lexis framework may be unrealistic or inconsistent with real systems. For example, in the case of the analysis of iGEM via Evo-Lexis, it is shown that there are specific dynamics that are not included in the modeling of Evo-Lexis. The increase/decrease of the set of sources is found to have an effect in efficiency and reuse of previously built modules throughout the years of iGEM competition. Introducing such dynamics in the model can be both insightful in the analysis of iGEM (e.g., quantifying the reduction of reuse in iGEM yearly batches), and more realistic modeling in Evo-Lexis (e.g., further what-if scenarios such as how the increase/decrease of the source set affects the hourglass property). However, it should be noted that iGEM is only one dataset and such generalizations of Evo-Lexis must be accompanied and verified with different datasets as well.

CHAPTER 7
CONCLUSIONS

The foundation of this thesis is Lexis, an optimization-based framework for revealing the hierarchical structure of sequence data. In Chapter 2, we motivated the use of Lexis in the analysis of hierarchical structure of sequential data by presenting its use-cases on developing optimized string hierarchies (an immediate use-case is in DNA synthesis), structure discovery over protein data, and feature extraction from textual data.

In Chapter 3, the research over the Lexis framework was expanded and explored by considering another related area of analysis of sequences. The connection of Lexis to the Smallest Grammar Problem (SGP) motivated us to present a generalization of SGP formulation and to develop corresponding algorithms for the newly introduced problems. In addition to meeting new boundaries in compression capabilities of lossless grammar-based codes, such generalization was found to be useful in unsupervised parsing of the structure of natural language.

In Chapter 4, we utilized the Lexis framework in the evolutionary modeling of hierarchical systems. Our proposed model, namely Evo-Lexis, is an attempt to explain the dynamics of such systems in an abstract and general framework. The emergence and evolution of the hourglass property and its relation to the optimization objective of a hierarchical system is one of the main types of analyses that is possible via the proposed model. Evo-Lexis also predicts major transitions in the evolution of both real technological and biological systems.

In the last research effort of this thesis, we analyzed a real dataset from synthetic biology, namely iGEM, in order to identify the strengths and weaknesses of Evo-Lexis modeling approach. Although the analysis shows that Lexis-DAGs on the iGEM data are less cost-efficient and less deep, the dataset exhibits a bias in reusing specific nodes more often

than others. This bias results in the Lexis-DAGs to take the shape of an hourglass with relatively high H-score values and stable set of core nodes, which are consistent with the predictions of Evo-Lexis modeling.

Overall, we first presented an alternative perspective on sequential data. We considered forming optimized graphical structures over such data which enables us to use the rich framework of graph analytics as a proxy for sequential data analysis. We showed such analytical framework can have several promising extensions in sequential data analysis. In addition to these data analysis aspects, we used the same framework in order to provide insights on some general and fundamental queries about evolving hierarchical systems. Our abstract modeling predicts realistic phenomena over hierarchical systems, e.g., hourglass effect and major transitions. We showed that aside from answering general modeling questions, we can use the framework in order to gain insight on evolution of realistic hierarchical systems.

Appendices

APPENDIX A: EXAMPLE OF DIFFERENCE OF NODE SELECTION ORDER IN G-LEXIS AND G-CORE

This example is run on the targets that are submitted during the first year of iGEM dataset (details of the dataset can be found in Chapter 5 of this thesis).

Table A.1: Order of the top-10 nodes identified in G-LEXIS (Numbers on the left show the order the nodes is removed in the algorithm and bio-bricks represent the string representation of the nodes removed)

1	BBa_B0010 BBa_B0012
2	BBa_B0032 BBa_C0040 BBa_B0010 BBa_B0012 BBa_R0040 BBa_B0032 BBa_C0012 BBa_B0010 BBa_B0012 BBa_R0010 BBa_B0032 BBa_C0051 BBa_B0010 BBa_B0012 BBa_R0063 BBa_B0030 BBa_C0062 BBa_B0010 BBa_B0012
3	BBa_B0030 BBa_C0061 BBa_B0010 BBa_B0012 BBa_R0010 BBa_B0030 BBa_C0060 BBa_B0010 BBa_B0012 BBa_R0040 BBa_B0030 BBa_E0022 BBa_B0010 BBa_B0012 BBa_R0040 BBa_B0030 BBa_E0032 BBa_B0010 BBa_B0012
4	BBa_B0034 BBa_C0040 BBa_B0010 BBa_B0012 BBa_R0040
5	BBa_B0034 BBa_C0012 BBa_B0010 BBa_B0012 BBa_R0010 BBa_B0034 BBa_C0051 BBa_B0010 BBa_B0012
6	BBa_B0034 BBa_C0061 BBa_B0010 BBa_B0012 BBa_R0063 BBa_B0034 BBa_C0062 BBa_B0010 BBa_B0012 BBa_R0011 BBa_B0034 BBa_C0060 BBa_B0010 BBa_B0012 BBa_R0040 BBa_B0034 BBa_E0032 BBa_B0010 BBa_B0012
7	BBa_B0012 BBa_B0011
8	BBa_I0500 BBa_B0034 BBa_E0022 BBa_B0010 BBa_B0012 BBa_I0500 BBa_B0034
9	BBa_B0034 BBa_C0012 BBa_B0010 BBa_B0012 BBa_R0011
10	BBa_B0034 BBa_C0051 BBa_B0010 BBa_B0012

Table A.2: Order of the top-10 nodes removed in G-CORE (Numbers on the left show the order the nodes is removed in the algorithm and bio-bricks represent the string representation of the nodes removed)

1	BBa_B0010 BBa_B0012
2	BBa_B0034
3	BBa_B0032 BBa_C0040 BBa_B0010 BBa_B0012 BBa_R0040 BBa_B0032 BBa_C0012 BBa_B0010 BBa_B0012 BBa_R0010 BBa_B0032 BBa_C0051 BBa_B0010 BBa_B0012 BBa_R0063 BBa_B0030 BBa_C0062 BBa_B0010 BBa_B0012
4	BBa_B0012 BBa_B0011
5	BBa_R0040
6	BBa_B0030 BBa_C0061 BBa_B0010 BBa_B0012 BBa_R0010 BBa_B0030 BBa_C0060 BBa_B0010 BBa_B0012 BBa_R0040 BBa_B0030 BBa_E0022 BBa_B0010 BBa_B0012 BBa_R0040 BBa_B0030 BBa_E0032 BBa_B0010 BBa_B0012
7	BBa_C0012
8	BBa_C0051
9	BBa_R0011
10	BBa_C0040

REFERENCES

[1] "Igem.org/main_page."

[2] C. Y. Baldwin and K. B. Clark, *Design rules: The power of modularity volume 1*. Cambridge, MA, USA: MIT Press, 1999, ISBN: 0262024667.

[3] E. Ravasz and A.-L. Barabási, "Hierarchical organization in complex networks," *Phys. Rev. E*, vol. 67, p. 026 112, 2 2003.

[4] H. Mengistu, J. Huizinga, J.-B. Mouret, and J. Clune, "The evolutionary origins of hierarchy," *PLOS Computational Biology*, vol. 12, no. 6, pp. 1–23, Jun. 2016.

[5] K. M. Sabrin and C. Dovrolis, "The hourglass effect in hierarchical dependency networks," *Network Science*, vol. 5, no. 4, pp. 490–528, 2017.

[6] C. R. Myers, "Software systems as complex networks: Structure, function, and evolvability of software collaboration graphs," *Phys. Rev. E*, vol. 68, p. 046 116, 4 2003.

[7] N. Sánchez, S. Park, and J. M. Finley, "Evidence of energetic optimization during adaptation differs for metabolic, mechanical, and perceptual estimates of energetic cost," *Scientific Reports*, vol. 7, no. 1, p. 7682, 2017.

[8] J. Clune, J.-B. Mouret, and H. Lipson, "The evolutionary origins of modularity," *Proceedings of the Royal Society of London B: Biological Sciences*, vol. 280, no. 1755, 2013.

[9] D. Gusfield, "Algorithms on strings, trees, and sequences: Computer science and computational biology," Cambridge Univ. Press, 1997.

[10] C. Nevill-Manning and I. Witten, "Identifying hierarchical structure in sequences: A linear-time algorithm," *J. Artif. Int. Res.*, vol. 7, no. 1, pp. 67–82, 1997.

[11] J. Kleinberg, "Bursty and hierarchical structure in streams," in *KDD*, Edmonton, Alberta, Canada, 2002, pp. 91–101, ISBN: 1-58113-567-X.

[12] H. Mengistu, J. Huizinga, J. Mouret, and J. Clune, "The evolutionary origins of hierarchy," *CoRR*, vol. abs/1505.06353, 2015.

[13] S. Miyagawa, R. Berwick, and K. Okanoya, "The emergence of hierarchical structure in human language," *Frontiers in Psychology*, vol. 4, no. 71, 2013.

[14] W. Dubitzky, M. Granzow, and D. P. Berrar, *Fundamentals of data mining in genomics and proteomics*. Springer Science & Business Media, 2007.

[15] J. Blakes, O. Raz, U. Feige, J. Bacardit, P. Widera, T. Ben-Yehezkel, E. Shapiro, and N. Krasnogor, "Heuristic for maximizing DNA re-use in synthetic DNA library assembly," *ACS Synthetic Biology*, vol. 3, no. 8, pp. 529–542, 2014.

[16] D. Densmore, T. Hsiau, J. Kittleson, W. DeLoache, C. Batten, and J. Anderson, "Algorithms for automated DNA assembly," *Nucleic Acids Res.*, vol. 38, no. 8, pp. 2607–2616, 2010.

[17] C. G. Nevill-Manning and I. H. Witten, "Identifying hierarchical strcture in sequences: A linear-time algorithm," *J. Artif. Intell. Res.(JAIR)*, vol. 7, pp. 67–82, 1997.

[18] M. Charikar, E. Lehman, D. Liu, R. Panigrahy, M. Prabhakaran, A. Sahai, and A. Shelat, "The smallest grammar problem," *Information Theory, IEEE Transactions on*, vol. 51, no. 7, pp. 2554–2576, 2005.

[19] M. Gallé, "Searching for compact hierarchical structures in DNA by means of the smallest grammar problem," PhD thesis, Université Rennes 1, 2011.

[20] R. Eyraud, "Inférence grammaticale de langages hors-contextes," PhD thesis, université Jean Monnet, 2006.

[21] P. Siyari, B. Dilkina, and C. Dovrolis, "Lexis: An optimization framework for discovering the hierarchical structure of sequential data," in *Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '16, San Francisco, California, USA: ACM, 2016, pp. 1185–1194, ISBN: 978-1-4503-4232-2.

[22] M. Charikar, E. Lehman, D. Liu, R. Panigrahy, M. Prabhakaran, A. Sahai, and A. Shelat, "The Smallest Grammar Problem," *IEEE Trans. on Inf. Theory*, vol. 51, no. 7, 2005.

[23] J. Kieffer and E. Yang, "Grammar based codes: A new class of universal lossless source codes," *IEEE Trans. on Inf. Theory*, vol. 46, p. 2000, 2000.

[24] A. Apostolico and S. Lonardi, "Off-line compression by greedy textual substitution," *Proceedings of the IEEE*, vol. 88, no. 11, pp. 1733–1744, 2000.

[25] G. Brodal, R. Lyngso, A. Ostlin, and C. Pedersen, "Solving the string statistics problem in time o(n log n).," in *ICALP*, 2002, pp. 728–739.

[26] M. Gallé, "Searching for compact hierarchical structures in DNA by means of the Smallest Grammar Problem," PhD thesis, Université Rennes 1, 2011.

[27] M. Farach, "Optimal suffix tree construction with large alphabets," in *Proceedings 38th Annual Symposium on Foundations of Computer Science*, 1997, pp. 137–143.

[28] S. Inenaga, T. Funamoto, M. Takeda, and A. Shinohara, "Linear-time off-line text compression by longest-first substitution," in *String Processing and Information Retrieval*, 2003, pp. 137–152.

[29] M. Newman, *Networks: An introduction*. Oxford University Press, Inc., 2010, ISBN: 0199206651, 9780199206650.

[30] K. M. Sabrin, "The hourglass effect in source-target dependency networks," PhD thesis, Georgia Institute of Technology, 2018.

[31] A. A. Hagberg, D. A. Schult, and P. J. Swart, "Exploring network structure, dynamics, and function using NetworkX," in *Proceedings of the 7th Python in Science Conference*, 2008, pp. 11–15.

[32] "Drive5.com/usearch/manual/uclust_algo.html."

[33] R. C. Edgar, "Search and clustering orders of magnitude faster than BLAST," *Bioinformatics*, vol. 26, no. 19, pp. 2460–2461, 2010.

[34] A. Apostolico, M. E. Bock, and S. Lonardi, "Monotony of surprise and large-scale quest for unusual words," in *Proceedings of the Sixth Annual International Conference on Computational Biology*, ser. RECOMB '02, Washington, DC, USA: ACM, 2002, pp. 22–31, ISBN: 1-58113-498-3.

[35] F. Cunial, *Analysis of the subsequence composition of biosequences*. Georgia Institute of Technology, 2012.

[36] H. T. Lam, F. Mörchen, D. Fradkin, and T. Calders, "Mining compressing sequential patterns," *Statistical Analysis and Data Mining*, vol. 7, no. 1, pp. 34–52, 2014.

[37] J. Zhang, Y. Wang, C. Zhang, and Y. Shi, "Mining contiguous sequential generators in biological sequences," *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, vol. PP, no. 99, pp. 1–1, 2015.

[38] N. Larsson and A. Moffat, "Offline dictionary-based compression," in *Data Compression Conference, 1999. Proceedings.*, 1999, pp. 296–305.

[39] J. Zhang, Y. Wang, and D. Yang, "CCSpan: Mining closed contiguous sequential patterns," *Knowledge-Based Systems*, vol. 89, pp. 1 –13, 2015.

[40] N. Tatti and J. Vreeken, "The long and the short of it: Summarising event sequences with serial episodes," in *KDD*, 2012, pp. 462–470, ISBN: 978-1-4503-1462-6.

[41] M. Gallé, "The bag-of-repeats representation of documents," in *ACM SIGIR Conference*, ACM, 2013.

[42] H. S. Paskov, R. West, J. C. Mitchell, and T. Hastie, "Compressive feature learning," in *NIPS*, 2013, pp. 2931–2939.

[43] H. S. Paskov, J. C. Mitchell, and T. J. Hastie, "Data representation and compression using linear-programming approximations," in *ICLR*, 2016.

[44] J. Lanctot, M. Li, and E. Yang, "Estimating DNA sequence entropy," in *ACM-SIAM Symposium on Discrete Algorithms*, 2000.

[45] S. Akhshabi and C. Dovrolis, "The evolution of layered protocol stacks leads to an hourglass-shaped architecture," in *Dynamics On and Of Complex Networks, Volume 2*, Springer, 2013.

[46] P. Siyari and M. Gall, "The generalized smallest grammar problem," in *Proceedings of The 13th International Conference on Grammatical Inference*, S. Verwer, M. van Zaanen, and R. Smetsers, Eds., ser. Proceedings of Machine Learning Research, vol. 57, Delft, The Netherlands: PMLR, 2017, pp. 79–92.

[47] J. C. Kieffer and E. Yang, "Grammar-based codes: A new class of universal lossless source codes," *Information Theory, IEEE Transactions on*, vol. 46, no. 3, pp. 737–754, 2000.

[48] Z. S. Harris, "Distributional structure," *Word*, vol. 10, no. 2-3, pp. 146–162, 1954.

[49] R. Carrascosa, F. Coste, M. Gallé, and G. Infante-Lopez, "The smallest grammar problem as constituents choice and minimal grammar parsing," *Algorithms*, vol. 4, no. 4, pp. 262–284, 2011.

[50] ——, "Searching for smallest grammars on large sequences and application to DNA," *Journal of Discrete Algorithms*, vol. 11, pp. 62–72, 2011.

[51] ——, "Choosing word occurrences for the smallest grammar problem," in *Language and Automata Theory and Applications*, 2010, pp. 154–165.

[52] F. Benz and T. Kötzing, "An effective heuristic for the smallest grammar problem," in *Proceedings of the 15th Annual Conference on Genetic and Evolutionary Computation*, ser. GECCO '13, Amsterdam, The Netherlands, 2013, pp. 487–494.

[53] R. Yoshinaka, "Identification in the limit of k, l-substitutable context-free languages," in *Grammatical Inference: Algorithms and Applications*, Springer, 2008, pp. 266–279.

[54] A. Clark and R. Eyraud, "Polynomial identification in the limit of context-free substitutable languages," *Journal of Machine Learning Research*, vol. 8, pp. 1725–1745, 2007.

[55] F. M. Luque and G. Infante-Lopez, "Pac-learning unambiguous k, l-NTS languages," in *Grammatical Inference: Theoretical Results and Applications*, 2010, pp. 122–134.

[56] F. Coste, G. Garet, and J. Nicolas, "Local substitutability for sequence generalization," in *International Conference on Grammatical Inference*, vol. 21, 2012, pp. 97–111.

[57] M. Van Zaanen, "ABL: Alignment-based learning," in *International Conference on Computational Linguistics*, 2000.

[58] Z. Solan, D. Horn, E. Ruppin, and S. Edelman, "Unsupervised learning of natural languages," *Proceedings of the National Academy of Sciences*, 2005.

[59] J. Scicluna and C. De La Higuera, "Pcfg induction for unsupervised parsing and language modelling.," in *EMNLP*, 2014, pp. 1353–1362.

[60] D. J. Cook and L. B. Holder, "Substructure discovery using minimum description length and background knowledge," *Journal of Artificial Intelligence Research*, pp. 231–255, 1994.

[61] B. Keller and R. Lutz, "Evolving stochastic context-free grammars from examples using a minimum description length principle," in *1997 Workshop on Automata Induction Grammatical Inference and Language Acquisition*, Citeseer, 1997.

[62] M. Nederhof and G. Satta, "Parsing non-recursive context-free grammars," in *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, 2002, pp. 112–119.

[63] G. Navarro, "A guided tour to approximate string matching," *ACM computing surveys (CSUR)*, vol. 33, no. 1, pp. 31–88, 2001.

[64] X. Chen, S. Kwong, and M. Li, "A compression algorithm for DNA sequences," *IEEE Engineering in Medicine and Biology*, vol. 20, no. 4, pp. 61–66, 2001.

[65] M. Van Zaanen, "Bootstrapping structure into language: Alignment-Based Learning," phd, University of Leeds, Leeds, UK, 2002.

[66] F. Dorr and F. Coste, "Compressing (genomic) sequences grammar inference," INRIA, Tech. Rep., 2014.

[67] C. G. Nevill-Manning and I. H. Witten, "On-line and off-line heuristics for inferring hierarchies of repetitions in sequences," *Proceedings of the IEEE*, vol. 88, no. 11, pp. 1745–1755, 2000.

[68] C. G. Nevill-Manning, "Inferring sequential structure," PhD thesis, University of Waikato, 1996.

[69] M. Marcus, G. Kim, M. Marcinkiewicz, R. MacIntyre, A. Bies, M. Ferguson, K. Katz, and B. Schasberger, "The Penn Treebank: Annotating predicate argument structure," in *Proceedings of the Workshop on Human Language Technology*, ser. HLT '94, Plainsboro, NJ, 1994, pp. 114–119.

[70] O. Vinyals, Ł. Kaiser, T. Koo, S. Petrov, I. Sutskever, and G. Hinton, "Grammar as a foreign language," in *Advances in Neural Information Processing Systems*, 2015, pp. 2755–2763.

[71] D. Klein, "The unsupervised learning of natural language structure," PhD thesis, Stanford University, 2005.

[72] W. Miller, "The hierarchical structure of ecosystems: Connections to evolution," *Evolution: Education and Outreach*, vol. 1, no. 1, pp. 16–24, 2008.

[73] N. Kashtan and U. Alon, "Spontaneous evolution of modularity and network motifs," *Proceedings of the National Academy of Sciences of the United States of America*, vol. 102, no. 39, pp. 13 773–13 778, 2005.

[74] N. Kashtan, E. Noor, and U. Alon, "Varying environments can speed up evolution," *Proceedings of the National Academy of Sciences*, vol. 104, no. 34, pp. 13 711–13 716, 2007.

[75] W. Callebaut and D. Rasskin-Gutman, *Modularity: Understanding the development and evolution of natural complex systems*, ser. Vienna series in theoretical biology. MIT Press, 2005, ISBN: 9780262033268.

[76] G. P. Wagner, M. Pavlicev, and J. M. Cheverud, "The road to modularity," *Nature Reviews Genetics*, vol. 8, 921 EP –, 2007, Review Article.

[77] S. Akhshabi and C. Dovrolis, "The evolution of layered protocol stacks leads to an hourglass-shaped architecture," in *Proceedings of the ACM SIGCOMM 2011 Conference*, ser. SIGCOMM '11, Toronto, Ontario, Canada: ACM, 2011, pp. 206–217, ISBN: 978-1-4503-0797-0.

[78] G. E. Hinton and R. R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," *Science*, vol. 313, no. 5786, pp. 504–507, 2006.

[79] T. Casci, "Hourglass theory gets molecular approval," *Nature Reviews Genetics*, vol. 12, 76 EP –, 2010.

[80] R. Tanaka, M. Csete, and J. Doyle, "Highly optimised global organisation of metabolic networks," *IEE Proceedings - Systems Biology*, vol. 2, no. 4, pp. 179–184, 2005.

[81] S Akhshabi, S Sarda, C Dovrolis, and S Yi, "An explanatory evo-devo model for the developmental hourglass [version 1; referees: 1 approved, 2 approved with reservations]," *F1000Research*, vol. 3, no. 156, 2014.

[82] T. Friedlander, A. E. Mayo, T. Tlusty, and U. Alon, "Evolution of bow-tie architectures in biology," *PLOS Computational Biology*, vol. 11, no. 3, pp. 1–19, Mar. 2015.

[83] M. Galle and M. Tealdi, "Xkcd-repeats: A new taxonomy of repeats defined by their context diversity," *Journal of Discrete Algorithms*, vol. 48, pp. 1 –16, 2018.

[84] T. C. Bell, J. G. Cleary, and I. H. Witten, *Text compression*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1990, ISBN: 0-13-911991-4.

[85] R. Hershberg, "Mutation–The Engine of Evolution: Studying Mutation and Its Role in the Evolution of Bacteria," *Cold Spring Harb Perspect Biol*, vol. 7, no. 9, a018077, 2015.

[86] W. B. Arthur, *The nature of technology: What it is and how it evolves*. Free Press, 2009.

[87] H. Youn, D. Strumsky, L. M. A. Bettencourt, and J. Lobo, "Invention as a combinatorial process: Evidence from us patents," *Journal of The Royal Society Interface*, vol. 12, no. 106, 2015.

[88] J. Schot and F. W. Geels, "Niches in evolutionary theories of technical change," *Journal of Evolutionary Economics*, vol. 17, no. 5, pp. 605–622, 2007.

[89] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to algorithms, third edition*, 3rd. The MIT Press, 2009, ISBN: 0262033844, 9780262033848.

[90] A. Barabási and M. Pósfai, *Network science*. Cambridge University Press, 2016, ISBN: 9781107076266.

[91] J. Rexford and C. Dovrolis, "Future internet architecture: Clean-slate versus evolutionary research," *Commun. ACM*, vol. 53, no. 9, pp. 36–40, Sep. 2010.

[92] S. Jain and S. Krishna, "Large extinctions in an evolutionary model: The role of innovation and keystone species," *Proceedings of the National Academy of Sciences*, vol. 99, no. 4, pp. 2055–2060, 2002.

[93] J. Smith and E. Szathmary, *The major transitions in evolution*. OUP Oxford, 1997, ISBN: 9780198502944.

[94] S. Valverde and R. V. Solé, "Punctuated equilibrium in the large-scale evolution of programming languages," *Journal of The Royal Society Interface*, vol. 12, no. 107, 2015.

[95] W. B. Arthur and W. Polak, "The evolution of technology within a simple computer model," *Complexity*, vol. 11, no. 5, pp. 23–31, 2006.

[96] D. Kim, D. B. Cerigo, H. Jeong, and H. Youn, "Technological novelty profile and invention's future impact," *EPJ Data Science*, vol. 5, no. 1, p. 8, 2016.

[97] S. Bakhshi and C. Dovrolis, "The price of evolution in incremental network design (the case of ring networks)," in *Bio-Inspired Models of Networks, Information, and Computing Systems: 6th International ICST Conference, BIONETICS 2011, York, UK, December 5-6, 2011, Revised Selected Papers*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 1–15, ISBN: 978-3-642-32711-7.

[98] S. Bakhshi and C. Dovrolis, "The price of evolution in incremental network design: The case of mesh networks," in *2013 IFIP Networking Conference*, 2013, pp. 1–9.

[99] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson, "How transferable are features in deep neural networks?" In *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2*, ser. NIPS'14, Montreal, Canada: MIT Press, 2014, pp. 3320–3328.

[100] A. M. Sharp, "Incremental algorithms: Solving problems in a changing world," AAI3276789, PhD thesis, Ithaca, NY, USA, 2007, ISBN: 978-0-549-19843-7.

[101] A. Borodin and R. El-Yaniv, *Online computation and competitive analysis*. New York, NY, USA: Cambridge University Press, 1998, ISBN: 0-521-56392-5.

[102] P. Flick and S. Aluru, "Parallel distributed memory construction of suffix and longest common prefix arrays," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '15, Austin, Texas: ACM, 2015, 16:1–16:10, ISBN: 978-1-4503-3723-6.

[103] P. Senin, J. Lin, X. Wang, T. Oates, S. Gandhi, A. P. Boedihardjo, C. Chen, and S. Frankenstein, "Time series anomaly discovery with grammar-based compression," in *Proc. of International Conference on Extending Database Technology*, 2015.

[104] X. Wang, J. Lin, P. Senin, T. Oates, S. Gandhi, A. P. Boedihardjo, C. Chen, and S. Frankenstein, "RPM: representative pattern mining for efficient time series classification," in *Proc. of International Conference on Extending Database Technology*, 2016.

[105] T. Oates, A. P. Boedihardjo, J. Lin, C. Chen, S. Frankenstein, and S. Gandhi, "Motif discovery in spatial trajectories using grammar inference," in *Proc. of ACM International Conference on Information & Knowledge Management*, 2013.

[106] V. Ishakian, D. Erdös, E. Terzi, and A. Bestavros, "A framework for the evaluation and management of network centrality," in *SIAM International Symposium on Data Mining*, 2012.