SYNTHESIS OF SIGNAL TEMPORAL LOGIC FORMULAS WITH GENETIC ALGORITHMS

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

SERTAÇ KAĞAN AYDIN

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
COMPUTER ENGINEERING

MAY 2019

Approval of the thesis:

SYNTHESIS OF SIGNAL TEMPORAL LOGIC FORMULAS WITH GENETIC ALGORITHMS

submitted by **SERTAÇ KAĞAN AYDIN** in partial fulfillment of the requirements for the degree of **Master of Science in Computer Engineering Department, Middle East Technical University** by,

Prof. Dr. Halil Kalıpçılar
Dean, Graduate School of **Natural and Applied Sciences** _____

Prof. Dr. Halit Oğuztüzün
Head of Department, **Computer Engineering** _____

Assist. Prof. Dr. Ebru Aydın Göl
Supervisor, **Computer Engineering, METU** _____

Examining Committee Members:

Prof. Dr. Halit Oğuztüzün
Computer Engineering, METU _____

Assist. Prof. Dr. Ebru Aydın Göl
Computer Engineering, METU _____

Assist. Prof. Dr. Engin Demir
Computer Engineering, Hacettepe University _____

Date:

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Surname: Sertaç Kağan Aydın

Signature :

ABSTRACT

SYNTHESIS OF SIGNAL TEMPORAL LOGIC FORMULAS WITH GENETIC ALGORITHMS

Aydın, Sertaç Kağan
M.S., Department of Computer Engineering
Supervisor: Assist. Prof. Dr. Ebru Aydın Göl

May 2019, 68 pages

Signal Temporal Logic (STL) is used to reason about the behavior of continuous signals. Due to its expressivity and algorithms to generate signal monitors, it is used to define monitoring rules for hardware and software systems. However, it is a hard task to define an STL formula that describes the system behavior. An expert of the monitored system should write the formula and optimize its parameters to minimize the monitoring errors (false alarms and missed alarms). To simplify and disseminate the use of formal monitors, its necessary to automate the formula writing process, which motivates this study. In this dissertation, we study the problem of synthesizing STL formulas from a labeled set of system traces.

The datasets from different fields may vary in size and can be quite large. In order to handle such cases, we developed a parallel formula synthesis method based on genetic algorithms. The proposed method does not require any expertise guidance; it generates formula structure and optimizes the formula parameters simultaneously. In addition, in order to find the local optimum around an individual, we developed a randomized search over the parameter space, which is shown to speed up the conver-

gence of the proposed method.

Keywords: Formal Methods, STL, Requirement Mining, Genetic Algorithms

ÖZ

SİNYAL ZAMANSAL MANTIK FORMÜLLERİNİN SENTEZLENMESİNDE GENETİK ALGORİTMALARIN KULLANIMI

Aydın, Sertaç Kağan
Yüksek Lisans, Bilgisayar Mühendisliği Bölümü
Tez Yöneticisi: Dr. Öğr. Üyesi. Ebru Aydın Göl

Mayıs 2019 , 68 sayfa

Sinyal zamansal mantık(Signal Temporal Logic, STL), devamlı sinyallerin davranışlarını anlamlandırmak için kullanılır. Anlatım gücü ve algoritmaları sayesinde, sinyal izleme prosedürleri için, donanım ve yazılım sistemlerinde kullanılacak izleme kurallarını tanımlamada kullanılır. Yine de, bir sistemin davranışını tanımlayan STL formülünü yazmak basit bir iş değildir. İzlenilecek sistemin bir uzmanının formülü üretip, ürettiği bu formülün parametrelerini, izleme anında doğabilecek hataları(yanlış ya da kaçırılan alarmlar) asgariye indirecek şekilde optimize etmesi gerekir. Dolayısıyla, formel izlemeyi basitleştirmek ve yaygınlaştırmak adına, formül oluşturma sürecinin otomatize edilme ihtiyacı vardır, ki bu çalışmanın temel motivasyonunu da bu ihtiyaç oluşturur.

Farklı alanlardan gelen farklı veri kümeleri birbirlerinden boyutca farklılıklar gösterirler, ve hatta çok çok büyüyebilirler de. Bu tip durumları da düşünerek, genetik algoritma bazlı paralel bir formül sentezleme metodu üzerine çalıştık. Sunduğumuz çözüm, bir uzman yardımına ihtiyaç duymaksızın, gerekli formülü türetip parametreleri optimize edebilmekte. Dahası her bir formül yapısına ait lokal optimum para-

metreleri bulabilmek adına, parametre uzayı üzerinde çalışan rastlantısal bir arama metodu geliştirdik, ve bu sayede önerilen formülün daha az tekrarda amaçlanan formüle yakınsamasını sağladık.

Anahtar Kelimeler: Formel Metotlar, STL, Gereksinim Türetme, Genetik Algoritmalar

to Güneş

ACKNOWLEDGMENTS

I am grateful to have the opportunity to have the experience of this thesis study and I want to thank all those people who made this possible.

First and foremost, I would like to thank and express my candid indebtedness to my advisor Ebru Aydın Göl. She supported me with great patience and guided like no one else can do. Moreover it is an honor to share her wisdom and learn from her.

Additionaly, I would like to thank the examining committee members Prof. Dr. Halit Oğuztüzün and Assist. Prof. Dr. Engin Demir for their kind attention to my study and wise comments that they made. I also express my gratitude to the members of cyber-physical systems research group for their suggestions and friendship during the preparation of this thesis.

I also want to thank my former company Sebit Eğitim ve Bilgi Teknolojileri A.Ş. and all co-workers that I worked with for their intellectual support.

I would like to thank my friends who never give up believing in me. Whenever I experience hard times I always felt their supports and camaraderie. I would like to mention Anıl Güven Yüksel, Ziya Can and Ceren Özcan who became a part of my life and and always offered my more than friendship.

I would like to thank my parents Osman and Kudret Aydın as well as Murat and Sibel Güneş for their incredible indulgence and support. I also need to mention my sister Aysu for her endless love and fondness.

The last but not the least I would like to thank my beloved wife, my best friend and supportive spouse Bade for all the things that she does. The effect of her is beyond this study and it is even not possible to express it by using words. I could not achieve this study or anything else without you. Thank you.

TABLE OF CONTENTS

LIST OF TABLES

TABLES

LIST OF FIGURES

FIGURES

LIST OF ABBREVIATIONS

Alg	Algorithm
Eq	Equation
Fig	Figure
GA	Genetic Algorithms
GP-UCB	Gaussian Process Upper Confidence Bound
GMM	Gaussian Mixture Model
LTL	Linear Temporal Logic
PSTL	Parametric Signal Temporal Logic
ptSTL	Past Time Signal Temporal Logic
ROI	Region of Interest
SMC	Statistical Model Checking
STL	Signal Temporal Logic
SVM	Support Vector Machine

CHAPTER 1

INTRODUCTION

1.1 Motivation

By the growing need in automation each day more and more human controlled systems are replaced by automated systems. Computer aided methods are used in many systems from traffic control to automobile industry. With these methods, the delivery speed is, yet need for monitoring and alert systems became inevitable because of the fact that accessibility and reliability of these services are crucial [1].

Increasing speed in delivery, makes the systems that are supervised by humans, inprofitable. Considering computer aided traffic control system which is supervised by humans, there is a need for large amount of supervisors, which eventually increases the cost of the system.

This fact derives the industry to use centralized monitoring systems, in which a supervisor may monitor many systems remotely. Monitoring systems are useful but still a little bit unrewarding considering the hunger in industry for fast delivery. This a a huge waste of time to employ a person to sit on his/her table to perceive something is going wrong on one sector in one of the giant monitors. It seems that the weak(slow) link in the chain is the human control. Moreover, computer aided systems does not have any problems almost 99 percent of the time. Hence, whether the person responsible to monitor the systems is useless most of the time, or he/she is responsible for so many systems that he/she is unable to react any situation in a reasonable amount of time.

Another approach to solve such problem is to create alerting systems. So, there is

no need to monitor the system all the time, instead there is another computer aided system that monitors the main system, to alert human beings in case of a problem. This method seems much more profitable, especially considering the need for human employees. Since there is no need to monitor the system second by second, the need for human employees decreases dramatically, by using alerting systems alongside with monitoring system.

However, it not an easy task to define alerting conditions in a proper way. The systems that are being monitored are huge systems with lots of subsystems in it. Moreover both the systems and the requirements that the systems are responsible to meet changes everyday. So, even having great expertise on the area, it is highly likely to miss situations that needs to be alerted. Or it is not a long shot to create extra-anxious alerting systems, that alerts so often and redundantly which results in humans to ignore real alerting situations.

In this thesis we are trying to build alerting rules for monitoring systems form the labeled dataset of the system behavior. To achieve so, we used signal temporal logic formulas considering the expressive power and human readability of signal temporal logic. In the coming sections we demonstrated novel examples from the literature, aiming to solve similar problems, but unlike them we do not need expert guidance. Moreover, in this dissertation, we proposed a model which provides a complete solution for the formula synthesis problem for system monitoring.

1.2 Research Objectives and Scope

In control theory, complex models are tested by using a simple set of specifications. However, in this approach there is no formal proof for whether the system meets the requirements or not. To specify and verify such systems, formal methods offers mathematically based languages, techniques and tools [2].

One of the most widely used and well known mathematics based language is linear temporal logic(LTL). In this dissertation, we are going to explain the details of LTL and two extensions of it, namely signal temporal logic(STL) and past time signal temporal logic(ptSTL). Signal temporal logic is a rich specification language which

is used to define temporal behaviors of signals [3, 4]. Considering large systems, although formal methods introduce a definite solution for the verification problem [5], application of the methods are difficult due to the system complexity [6, 7]. Signal temporal logic formulas on the other hand, provide an efficient way of checking system traces [8]. Moreover it is possible to generate STL(or ptSTL) formulas in an automated manner [4]. Furthermore, STL formulas are expressive to represent rich specification of the system not only in a precise way, but also in a human readable form [9, 10]. In this dissertation we build up an efficient method for generating ptSTL formulas from labeled system traces. The whole method is unsupervised, in other words there is no need for an expert of the system to determine the monitoring rules or their structures. Furthermore, since the ptSTL formulas are as expressive as STL formulas [11] and also does not depend on future values of the system traces, the developed system is an early warning system. We propose a ptSTL formula synthesis method based on genetic algorithms.

1.3 Structure of Thesis

This chapter is the introduction section for the thesis, in which we define our motivation, research objectives and scope and structure of the thesis in an informal way. In Chapter 2, we build a base for our study by providing formal background on "Signal Temporal Logic" and "Genetic algorithms". In the Chapter 3 we introduced novel examples from the literature, which works on similar problems. Since the problem is to synthesize signal temporal logic formulas, we divided the chapter into two sections, which covers the studies that are aiming to optimize a signal temporal logic formula's parameters and studies that both build up the structure of the formula and optimize its parameters simultaneously. In Chapter 4, we formally defined the formula synthesis problem. Moreover, we outlined the proposed approach. Chapter 5 is reserved for genetic algorithms, the application of the genetic algorithms in our problem. In Chapter 6 we proposed a method for optimization of ptSTL logic formula parameters. The proposed method is examined in three sections belongs to that chapter. In Chapter 7, we combined the methods that are introduced in Chapter 5 and Chapter 6 to introduce a complete solution. We integrated optimization methods in genetic algorithms as a

new phase which we call adaptation. Chapter 8 is the part that we realize our proposed solutions by using case studies. We gave detailed information about how we adapt our solution to different problem spaces and provide results of the algorithms that we run on that datasets. Lastly in Chapter 9 we summarize the work done in this dissertation, and draw a brief conclusion.

CHAPTER 2

PRELIMINARY INFORMATION ON STL AND GENETIC ALGORITHMS

In this dissertation, we developed methods to synthesize "Signal Temporal Logic" formulas by using "Genetic Algorithms". The background information about these topics is given in this chapter.

2.1 Temporal Logics

2.1.1 Linear Temporal Logic(LTL)

Linear Temporal Logic (LTL) formulas are constructed from a set of atomic propositions, Boolean operators, and temporal operators. Boolean true, negation and conjunction are denoted by True, $\neg$ and $\wedge$ respectively. Other Boolean operators such as disjunction($\vee$) and implication($\implies$) can be obtained by using the combinations of these three. Until (U) and next (X) are the temporal operators. The formal definition of the syntax of an LTL formula ϕ over a given set of propositions O is recursively defined as follows:

$$\phi = o \mid true \mid \phi_1 \wedge \phi_2 \mid \neg\phi \mid X\ \phi \mid \phi_1 U\ \phi_2 \tag{21}$$

where $o \in O$ is an atomic proposition and ϕ,ϕ_1 and ϕ_2 are LTL formulas.

Two additional temporal operators Globally(G) and Eventually(F) are commonly used in literature. However they did not take place in 21 since they can be obtained as follows:

$$F\ \phi = trueU\ \phi \tag{22}$$

$$G\ \phi = \neg F\ \neg\phi \tag{23}$$

The satisfaction of formula ϕ over a set of propositions O at position $k \in N^+$ by word $w = w(1)w(2)w(3)... \in (2^P)^\omega$, denoted by $w(k) \models \phi$, is defined recursively as follows [12]:

- $w(k) \models true$
- $w(k) \models o$ for some o if $o \in w(k)$
- $w(k) \models \neg\phi$ if $w(k) \not\models \phi$
- $w(k) \models \phi_1 \wedge \phi_2$ if $w(k) \models \phi_1$ and $w(k) \models \phi_2$
- $w(k) \models X\ \phi$ if $w(k+1) \models \phi$
- $w(k) \models \phi_1 U\ \phi_2$ if there exists $j \geq k$ such that $w(j) \models \phi_2$ and for all $k \leq i < j$, we have $w(i) \models \phi_1$

A word w satisfies ϕ if $w(0) \models \phi$. In order to provide a better insight informal definitions of the LTL operators is shown below:

1. $X\ \phi$ holds at the current step if ϕ is satisfied at he next step
2. $\phi_1 U\ \phi_2$ holds if ϕ_1 is satisfied until ϕ_2 is satisfied
3. $G\ \phi$ holds if ϕ is satisfied at each step
4. $F\ \phi$ is satisfied if ϕ is satisfied at some future step (i.e., ϕ is "eventually" satisfied).

2.1.2 Signal Temporal Logic(STL)

Signal Temporal Logic(STL) is an extension of LTL with real valued signals and real valued constraints. STL is equipped with quantitative semantics, which defines the degree of satisfaction (or violation) of the formula [13].

The syntax of STL formula is obtained by introducing a time bound to the temporal operators. In addition, linear inequalities are used instead of propositions.

$$\phi = true \mid x < c \mid \phi_1 \wedge \phi_2 \mid \neg\phi \mid G_{\ [a,b)}\phi \mid F_{\ [a,b)}\phi \mid \phi_1 U_{\ [a,b)}\phi_2 \tag{24}$$

Boolean true($true$), negation($\neg$) and conjunction($\wedge$) operators are the same in LTL formula definition. x is variable and c is any real valued constant, i.e. $c \in \mathbb{R}$. $a, b \in \mathbb{R}^+$ are time bounds. Informally, the semantics of bounded operators are defined as follows:

- $F_{\ [a,b)}\phi$ is satisfied at time t if ϕ is satisfied at some point in time between $[t+a, t+b)$
- $G_{\ [a,b)}\phi$ is satisfied at time t if ϕ is satisfied at every time in $[t+a, t+b)$
- $\phi_1 U_{\ [a,b)}\phi_2$ is satisfied at time t if there exists $t+a \leqslant i \leqslant t+b$ such that ϕ_2 is satisfied at time i and ϕ_1 is satisfied every time in $[t+a, i)$.

The semantics of STL is defined over real valued signals. A piecewise continuous n-dimensional signal is represented as:
$T = (t_0, s_0), (t_1, s_1), (t_2, s_2), \ldots, (t_{N-1}, s_{N-1}), (t_N, s_N)$.
In this notation $t_i \in \mathbb{R}_+$ are time labels and $s_i \in \mathbb{R}^n$ is a vector consist of system metrics. The valuation of the signal at time t is denoted by (T, t). For a piece-wise continuous signal $\{t_i\}_{i=0,\ldots,N}$ is an increasing series and $(T, t) = (T, t_i)$ is valid for all $t \in [t_i, t_{i+1})$

The robustness degree is the quantitative evaluation of STL formula. The quantitative valuation $\rho(\phi, T, t)$ of an STL formula ϕ at time t is computed as follows:

$$
\begin{aligned}
\rho(true, T, t) &= +\infty \\
\rho(x < c, T, t) &= c - (T, t)[x] \\
\rho(\phi_1 \wedge \phi_2, T, t) &= \min(\rho(\phi_1, T, t), \rho(\phi_2, T, t)) \\
\rho(\phi_1 \vee \phi_2, T, t) &= \max(\rho(\phi_1, T, t), \rho(\phi_2, T, t)) \\
\rho(F_{\ [a,b)}\phi, T, t) &= \max_{t' \in [t+a, t+b)} \rho(\phi, T, t') \\
\rho(G_{\ [a,b)}\phi, T, t) &= \min_{t' \in [t+a, t+b)} \rho(\phi, T, t') \\
\rho(\phi_1 U_{\ [a,b)} \phi_2, T, t) &= \max_{t' \in [t+a, t+b)} \min(\rho(\phi_2, T, t'), \\
&\qquad \min_{t'' \in [t', t]} \rho(\phi_1, T, t''))
\end{aligned}
\tag{25}
$$

The quantitative valuation provides information about both the binary satisfaction information and the robustness of satisfaction or violation. $\rho(\phi, T, t) > 0$ means that the signal T satisfies ϕ at time t. The greater the $\rho(\phi, T, t)$ the more robust the satisfaction is. Conversely $\rho(\phi, T, t) < 0$ means that the signal T violates ϕ at time t. For equality, $\rho(\phi, T, t) = 0$, we need to resort back the qualitative semantics.

2.1.3 Past Time Signal Temporal Logic(ptSTL)

LTL and STL use future temporal operators. However, in some cases, including the system monitoring, it is more intuitive and practical to reason about the past behavior, e.g., the available information up to the current time.

Past time Signal Temporal Logic (ptSTL) is defined to express specifications over the past behavior. A ptSTL formula is defined using the past temporal operators and Boolean operators are defined as:

$$
\phi = true \mid x < c \mid \phi_1 \wedge \phi_2 \mid \neg\phi \mid G^{-}_{\ [a,b)}\phi \mid F^{-}_{\ [a,b)}\phi \mid \phi_1 S_{\ [a,b)} \phi_2 \tag{26}
$$

Informally, the semantics of past time operators:

- $F^-_{\ [a,b)}\phi$ is satisfied at time t if ϕ is satisfied at some point in time between $[t-b, t-a)$
- $G^-_{\ [a,b)}\phi$ is satisfied at time t if ϕ is satisfied at every time in $[t-b, t-a)$
- $\phi_1 S_{\ [a,b)} \phi_2$ is satisfied at time t if ϕ_2 is satisfied at some time in $[t-b, t-a)$ and ϕ_1 is satisfied since that time.

Similar to STL ptSTL is also equipped with quantitative semantics. To simplify the notation, we use the following function for the interval computation:

$$I(t, [a, b)) = (t-b, t-a] \cap [0, t] \tag{27}$$

The quantitative valuation ρ of a signal T at time t for a formula is recursively computed as follows.

$$\begin{aligned}
\rho(true, T, t) &= +\infty \\
\rho(x < c, T, t) &= c - (T, t)[x] \\
\rho(\phi_1 \wedge \phi_2, T, t) &= \min(\rho(\phi_1, T, t), \rho(\phi_2, T, t)) \\
\rho(\phi_1 \vee \phi_2, T, t) &= \max(\rho(\phi_1, T, t), \rho(\phi_2, T, t)) \\
\rho(F^-_{\ [a,b)}\phi, T, t) &= \sup_{t' \in I(t,[a,b))} \rho(\phi, T, t') \\
\rho(G^-_{\ [a,b)}\phi, T, t) &= \inf_{t' \in I(t,[a,b))} \rho(\phi, T, t') \\
\rho(\phi_1 S_{\ [a,b)} \phi_2, T, t) &= \sup_{t' \in I(t,[a,b))} \min(\rho(\phi_2, T, t'), \\
&\qquad \inf_{t'' \in [t', t]} \rho(\phi_1, T, t''))
\end{aligned} \tag{28}$$

Parametric signal temporal logic is another extension of STL in which parameters are used in predicates and intervals instead of constant numerics. Given a parametric ptSTL formula ψ and valuations v for its parameters, a ptSTL formula $\psi(v)$ is obtained by replacing the parameters with the corresponding values. For example, $\psi = F^-_{\ [0,p_1]} x < p_2$ is a parametric formula and $\psi(25, 6) = F^-_{\ [0,25]} x < 6$ is the ptSTL formula obtained from ψ with parameter valuation $p_1 = 25$ and $p_2 = 6$.

2.2 Genetic Algorithms

Genetic algorithms is one of the most popular evolutionary computation techniques. According to Mitchell, most of methods that is called as genetic algorithm, commonly have the following aspects at minimum: representation of candidate solutions as chromosomes, fitness measurement of the solutions, selection, crossover, reproduction of new offspring and random mutation [14]. A very general genetic algorithm is drawn as given in Algorithm 1

Algorithm 1 Simple Genetic Algorithm

1: Initialization: Start with a randomly generated population of n chromosomes
2: Compute fitness: Compute the fitness of each chromosome x in the population.
3: **while** A certain number of steps or until the fitness converges **do**
4: Selection
5: Crossover
6: Mutation
7: Create the new generation
8: Fitness computation of the new generation.
9: **end while**

2.2.1 Representation

Genetic representation is the task to map the phenotype of the data in the problem space into a genotype. The better the representation, the higher possibility of combination of crossover and mutation operations to generate increasingly better individuals (chromosomes). Furthermore, better representation results in a better chance of the set of all possible genotypes to cover the whole problem space. Moreover, evolvability dramatically increases or decreases with the representation [15].

2.2.2 Initial Population

Initial population is the set of chromosomes that represents a sample space of the problem defined. The initial population should be large and diverge enough to rep-

resents different minorities of the problem space. Otherwise the algorithm may converge to local extremum, so that misses the optimal solutions [15].

2.2.3 Crossover

Crossover is the process of chosen chromosomes to form new offsprings that inherits characteristics from both parents. The chosen chromosomes are randomly paired and some part of their genotypes are swapped [15].

2.2.4 Mutation

Mutation is the random modifications in the genotype of an individual. The purpose of the mutation operation is to reach every point in the space. Mutation is useful to avoid local minima and to achieve better progress in highly converged populations [15].

2.2.5 Selection

Selection is the process of choosing individuals from a population. The selected individuals will transfer their genomes to the new generations. The selected individuals are going to be paired for crossover. The selection works in favor of the evolution principle: best fit to the nature survives.

2.2.6 Fitness Function

Fitness function shows how well an individual fit to the given problem. The main aspects to occupy a fitness functions are the choice and combination of fitness components and the way that the function is evaluated [15].

2.3 Binary Classification

The binary classification is the task of mapping the elements of a set into two groups, negative(0) or positive(1). A training set for a binary classification is defined as :

$$\mathcal{D} = \{(X_i, L^{X_i}) \mid L^{X_i} \in \{0, 1\} \text{ for each } i = 0, \ldots, N\},$$

where X_i is a data point and L^{X_i} is its binary class label. A binary classifier P($\cdots$) maps X_i to a predicted value $P(X_i) \in \{0, 1\}$. The success of a binary classifier is computed based on its performance over dataset $\mathcal{D}$ [16].

Since the success of such binary classification task rely on values $P(X_i) \in \{0, 1\}$ and $L^{X_i} \in \{0, 1\}$, it introduces four different classes with respect to the original label and the predicted one. These classes are shown in Table 2.1.

Table 2.1: Binary classification

Original/Predicted	**Positive(+)**	**Negative(-)**
Positive(+)	True Positive(TP)	False Negative(FN)
Negative(-)	False Positive(FP)	True Negative(TN)

For a dataset $\mathcal{D}$, and classifier P the number of instances for each class from Table 2.1 is computed as follows:

$$C_{TP}(P, \mathcal{D}) = \sum_{i=1}^{N} \begin{cases} 1 \text{ if } P(X_i) == 1 \text{ and } L^{X_i} == 1 \\ 0 \text{ otherwise} \end{cases} \tag{29}$$

$$C_{FP}(P, \mathcal{D}) = \sum_{i=1}^{N} \begin{cases} 1 \text{ if } P(X_i) == 1 \text{ and } L^{X_i} == 0 \\ 0 \text{ otherwise} \end{cases} \tag{210}$$

$$C_{TN}(P, \mathcal{D}) = \sum_{i=1}^{N} \begin{cases} 1 \text{ if } P(X_i) == 0 \text{ and } L^{X_i} == 0 \\ 0 \text{ otherwise} \end{cases} \tag{211}$$

$$C_{FN}(P,\mathcal{D}) = \sum_{i=1}^{N} \begin{cases} 1 \text{ if } P(X_i) == 0 \text{ and } L^{X_i} == 1 \\ 0 \text{ otherwise} \end{cases} \tag{212}$$

To measure the success, the ratio of correctly classified instances ($\frac{TP+TN}{N}$) can be used. Different success measures are used according to the problem domain. Commonly used ones are defined as follows. Sensitivity is measured by recall value while specificity is measured by precision [17]. Recall is sensitive to true positive cases that are correctly classified as positive while, precision is the proportion of predicted positive cases that are correctly real positives. F-score is used to balance these two measures.[17]. These functions may also be utilized in genetic algorithms as fitness functions. In Table 2.2 mathematical definitions of these functions are given.

Table 2.2: The commonly used fitness functions and their formulations.

Name	**Abbreviation**	**Equation**
Inverse Mismatch Ratio	IMR	$\frac{1}{\mathcal{C}_{FP}+\mathcal{C}_{FN}}$
Precision	P	$\frac{\mathcal{C}_{TP}}{\mathcal{C}_{TP}+\mathcal{C}_{FP}}$
Recall	R	$\frac{\mathcal{C}_{TP}}{\mathcal{C}_{TP}+\mathcal{C}_{FN}}$
F1-Score	F1	$\frac{2*R*P}{R+P}$

CHAPTER 3

RELATED WORK

STL formulas can be used to generate monitoring rules of a system. However, it is not a easy task to obtain STL formulas that serve as a monitoring rule. Therefore, it is important to automate the formula writing process.

The formula synthesis problem is emerged in different forms. Some of the studies are interested in finding the formula that best defines the properties of a given set of signals, while some of them are focusing on "good" and "bad" system behaviors and synthesizing formulas that make a distinction between the "good" and the "bad" behavior[18].

The formula synthesis problems can also be classified with respect to the required information regarding the formula structure. There exists studies focusing on synthesizing parameters for a predefined formula template, whereas some studies aim at generating both the structure and the parameters related to that formula structure. In this chapter, we divided these approaches in to two sections namely "Parameter Synthesis" and "Formula Synthesis" accordingly.

3.1 Parameter Synthesis

Asarin proposed a construction of STL formula in [19]. He introduced Parametric Signal Temporal Logic (PSTL) to distinguish the definition of formula structure and parameters. For many cases it is important to be able to describe the STL structure for unknown parameters and then synthesize the parameters for the structure which best describes the dataset [20]. In [21], authors defined a set of PSTL formulas to

represent certain bahaviours of the system, such as "spike" or "settling time" and optimize parameters for the template formulas that represents the certain behaviors.

The parameters that are used in PSTL formulas are mainly divided into two groups namely magnitude parameters $\{p_1, \cdots, p_g\}$ and timing parameters $\{s_1, \cdots, s_h\}$ to form hyper-rectangles $\mathbb{R}^g$ and $\mathbb{R}^h$ [19]. The parametric formula, in other words the structure of the STL formula is formed by using these hyper-rectangles. An example PSTL formula for the STL formula structure is shown in Equation 31.

$$\phi = G_{\ [0,s_1)}(((x_0 > p_1) U_{\ [0,s_2)} (x_0 > p_2))) \tag{31}$$

The valuation of parameter $(u, v) \in \mathbb{R}^g$ and $\mathbb{R}^h$ generates an STL formula $\phi^\star$ from PSTL formula ϕ in the form of Equation 32.

$$\phi^\star = G_{\ [0,v_1)}(((x_0 > u_1) U_{\ [0,v_2)} (x_0 > u_2))) \tag{32}$$

For example, an expert knows that if the blood sugar exceeds a certain amount, $p_1\ mg/dl$, it should drop to a lower level, $p_2\ mg/dl$, in a certain amount of time $s_1\ min$. Then the parametric STL formula become:

$$G\ (blood\_sugar < p_1 \wedge F_{\ [0,\epsilon)} blood\_sugar > p_1) \implies G_{\ [\epsilon,s_1) blood\_sugar < p_2} \tag{33}$$

The purpose of the parameter synthesis problem is to gather parameters corresponds to Equation 33 from the given dataset. After the synthesizing phase it is expected to obtain an STL formula in the form of:

$$G\ (blood\_sugar < 140 \wedge F_{\ [0,\epsilon)} blood\_sugar > 140) \implies G_{\ [\epsilon,60) blood\_sugar < 120} \tag{34}$$

Different studies in literature occupies different methods to synthesize parameters for predefined formula template. In their study Jin, Donze, Deshmukh and Seshia mine the STL requirement from Simulink [22] models. After generating the parameters, they run a falsification algorithm to obtain counter example traces that violate the mined formula. Then they added them to their dataset, which they use as simulation traces in next iterations. If falsification algorithm they developed failed to find a

falsifying trace, the mining algorithm terminates [23]. Their case studies are from automotive systems and they synthesized parameters for requirements such as automatic transmission, air-fuel ratio and settling time.

A similar study uses stochastic optimization methods to automatically synthesize parameters of a PSTL formula. After generating the parameters, the trace is analyzed by a robutness analyzer to obtain best robustness value so far. Since the parametric robustness function they use is parametric, they behave their problem as either a minimization or maximization problem [7]. They used S-TaLiRo [24] for falsification, which is a tool containing a number of stochastic optimization methods [25, 26, 27]. Afterwards Hoxha extended the solution to work with multidimensional parameter range [20]. Hence, they developed a solution to multi-parameter mining problem which is in the form of Pareto front [28].

Two other studies occupy a statistical model checking (SMC) procedure to evaluate candidate parameters that belong to PSTL formula [18, 29]. SMC is a procedure that simulates the system for finitely many executions to provide a statistical evidence for the violation or the satisfaction of the specification with respect to the samples [30]. Their aim is also to gather the optimum result with as few function evaluations as possible. To achieve so, they also used the Gaussian process upper confidence bound (GP-UCB) algorithm to deal with continuous optimization of STL parameters. GP-UCB algorithm is a variant of Gaussian process optimization performs Bayesian updates to increase efficiency [31].

The sixth study to mention is using unsupervised machine learning algorithms to extract features from PSTL templates for timed traces [32]. They defined a projection operation that maps given timed trace to the unique parameter valuations with properties "tightness" and "uniqueness" [32]. They used two unsupervised learning techniques which are already implemented in the scikit-learn toolkit [33]: Gaussian Mixture Models (GMMs) [34] and Support Vector Machine (SVM) [35].

TeLEx [36] approach offered by Jha learns PSTL parameters only from positive examples by using a tightness metric [36]. Learning from positive examples has been studied by many researchers in different domains[37, 38, 39]. Since the tool has no access to the negative examples, to avoid over-generalization they used a tightness

metric to measure how tight the STL formula is satisfied by the trace. They also used a gradient descent based method by using the slickness of the offered tightness metric [36].

3.2 Formula Synthesis

The other significant research topic is to synthesize the STL formula that describes the traces. In other words, researches aimed to synthesize both the PSTL formula which serves as STL formula structure, and parameters of it. This type of studies are mainly on domains that the structure of the solution is vague or unknown.

In [40] and [41], the authors created a directed acyclic graph (DAG) where a path exists between possible formula structures. They construct the graph, generate parameters for each candidate structure using simulated annealing [42]. After the first set of execution is applied, they prune the nodes in DAG with high costs and grow the promising nodes in DAG till they come up with a sufficiently satisfying formula [40]. However, different than [40], authors developed a tightness function which punishes the size of the formula in [41] to avoid to search for a formula that probably have a poor performance. Furthermore, in [40], they introduced a concept of causality by searching for formulas in the form of $\phi_{cause} \implies \phi_{effect}$, and given ϕ_{cause}, searching for ϕ_{effect} or vice versa.

In another work, authors approached the STL synthesis problem as a two-class classification problem, and proposed a desicion-tree based approach to learn STL classifiers [9]. They developed a framework of decision tree learning. Their algorithm consists of two parts. First they build a decision tree from the given trace. Since learning optimum decision tree problem is NP-complete [43], they introduced some predefined stopping conditions. Than convert the created decision tree into an STL formula.

Yoo and Belta also followed a decision tree based approach in [10]. They first extract the spatial predicates from the provided signal. In other words, their method obtains region of interest (ROI) from a set of signal where there is a peak level of desired/undesired behaviors. Then they construct a set of signal formulas using ROIs which

they call logical extraction. Finally, they use a decision tree based algorithm similar to [9] to obtain STL formula.

In [44], Aydin Gol developed a grid search based algorithm to solve PSTL formula structure generation and parameter synthesis of the formula structure generated. Since the set of PSTL formulas are finite, the model developed is looking for all possible PSTL structures as a candidate solution. However, domain of the parameters are in continuous space, defining a step size for magnitude and timing parameters became inevitable.

A recent study also occupies decision tree techniques to mine STL formula. However, unlike [9, 10], the whole system trace is not marked as "good" or "bad", instead every data point has an associated label. Their algorithm consists of three main steps. They firstly convert the time series data set via windowing to obtain new features. Then, they used Apache WEKA [45] implementation of Quinlan's C4.5 decision tree algorithm [46] to generate the decision tree. Lastly, they converted the generated decision tree into a ptSTL formula [47].

CHAPTER 4

PROBLEM DEFINITION AND PROPOSED APPROACH

4.1 Problem Definition

In this dissertation, our goal is to synthesize a monitoring rule as a ptSTL formula such that the qualitative valuation of the formula along the traces will mimic the labels (alerting conditions). In our set-up, we assume that the system produces the values of n metrics with time-stamps. We have a set of labeled system traces in the following form:

$$\begin{aligned}\mathcal{D} = \{(T, L^T) \mid T =&(t_0, s_0), (t_1, s_1), \ldots, (t_{m^T}, s_{m^T}),\\ L^T =&(t_0, l_0), (t_1, l_1), \ldots, (t_{m^T}, l_{m^T}), m^T \in \mathbb{N}\\ \text{and } s_i \in&\mathbb{R}^n, l_i \in \{0, 1\} \text{ for each } i = 0, \ldots, m^T\},\end{aligned} \tag{41}$$

where T denote a system trace that is a piecewise-affine signal and L^T denote its label. $L^T(t_i) = 1$ is the positive label at time t_i. Unlike other studies at the literature, the label is not comprise the whole trace, instead each time point in a trace has its own label.

To formalize, the aim of the study is to solve Problem 1.

Problem 1. Given a set of labeled system traces $\mathcal{D}$ as in (41), find a ptSTL formula ϕ^* maximizes the given fitness value among $\mathcal{D}$, i.e., find ϕ^* such that

$$\phi^* = \text{argmax}_{\phi \text{ is a ptSTL formula}}\, \mathcal{C}(\mathcal{D}, \phi). \tag{42}$$

In Problem 1, $\mathcal{C}(\mathcal{D}, \phi)$ is a fitness value representing how well the formula ϕ describes

the dataset $\mathcal{D}$. In Chapter 2, we propose candidate fitness functions and present corresponding computation methods. An example for Problem 1 is given in Example 4.1.1.

Example 4.1.1. An example case for STL synthesis problem is CPU rate. The purpose of CPU rate case is to obtain a ptSTL formula, which will serve as the monitoring rule for CPU usage. The data being used is generated synthetically by the following rules. The duration between two consecutive data points is randomly picked from interval $[1, 5]$. For each signal, we randomly pick a high rate duration $hr^d \in [30, 100]$ and a start time st for the high rate. If $hr^d > 50$, we mark the signal during $[st + 50, st + hr^d]$ as violating (label 1). For $[st, st + hr^d]$, we generate data points randomly from Gaussian distribution with mean 35 and deviance 6 denoted as $G(35, 6)$. For the rest of the signal, at each time point, we generate data points from $G(15, 6)$ with 0.98 probability, and we generate data points from $G(45, 6)$ with 0.02 probability. The latter part is to mimic the errors in the labeling process. In addition, we limit the generated values to range $[0, 100]$ for the CPU rate. An example trace for CPU rate is given in Figure 4.1.

In the figure the CPU signal is marked with blue color where, its label is marked with orange. For a better view the label is scaled. We generated 500 different labeled traces for dataset $\mathcal{D}$.

4.2 Proposed Approach

Synthesizing ptSTL formula from a given labeled dataset is not an one of a kind, but still a contemporary problem. Most of the studies in the literature commonly worked on datasets such that, whole signal is labeled as 1 or 0. However, as seen in (41) and Figure 4.1, we attack datasets that contains signals which are labeled as 1 or 0 at each point. Moreover our aim is not only to synthesize parameters for a predefined formula template, but also to gather the ptSTL formula with its structure and parameters, which defines the given dataset. To achieve so, we used genetic algorithms, in order to work with continuous parameters in the search space, and also to minimize huge run times that occur in methods which occupies grid search methods. Certain design decision that we draw about genetic algorithms to solve synthesizing ptSTL problem

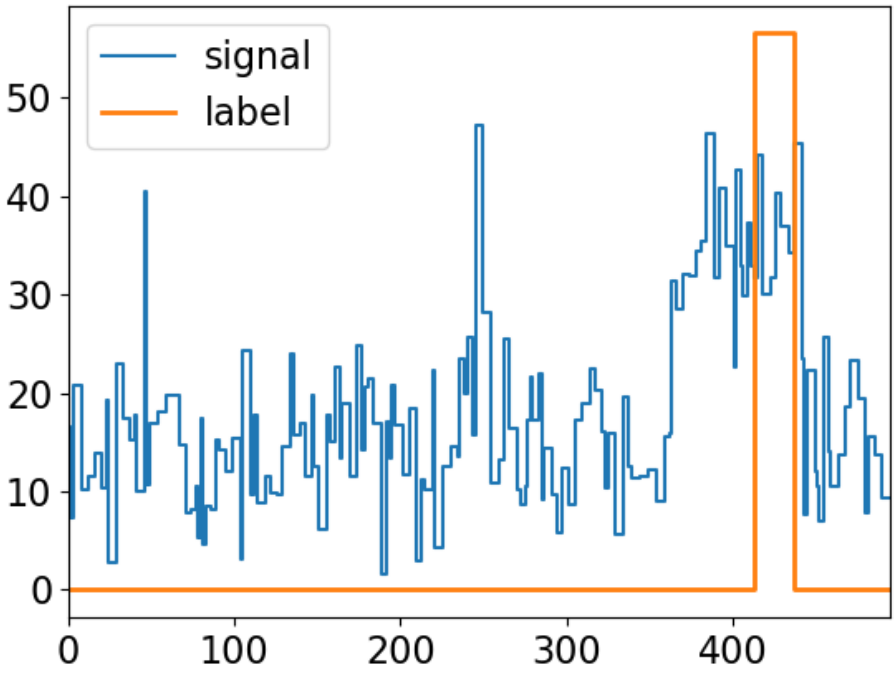

Figure 4.1: A sample signal for CPU rate case and its label. The label is scaled for better view.

are explained in the coming sections.

As an expansion we occupied parameter optimization which we call "adaptation" later on. Our aim is to develop a novel extension to genetic algorithms, where the expectation is to avoid extinction of candidate solutions. Moreover, by this extension we are able to obtain suitable formulas, from early stages of evolutionary process.

CHAPTER 5

GENETIC ALGORITHMS FOR STL SYNTHESIS

To use genetic algorithms for ptSTL formula synthesis, the first step is to encode ptSTL formulas as chromosomes, which require us to determine a representation. The choice of relevant genetic representation is not a trivial problem [15]. The most commonly used representations techniques are integer or real valued vectors, or tree-based structures. For our problem, we chose tree-based representation, since temporal logic formulas can be represented by their syntax trees. To limit the search space for the genetic algorithms, we define a set of parameters, which are shown in Table 5.1.

Table 5.1: Parameters that genetic algorithms use for STL synthesis

Parameter	Definition	Example
Operator Count	a tuple that defines the minimum and the maximum number of operators that an STL formula have	[0,2]
Parameter Domains for Past Globally	a tuple that defines the upper and lower limit for G^- operators time bound	[5,30]
Parameter Domains for Past Eventually	a tuple that defines the upper and lower limit for F^- operators time bound	[5,30]
Parameter Domains for Since	a tuple that defines the upper and lower limit for S operators time bound	[5,30]
Parameter Domains for Variables	a set of tuples that defines the upper and lower limit for each STL magnitude parameter	$\{(x^0, [0, 20]), (x^1, [15, 35])\}$

5.1 Representation

Each ptSTL formula is considered as an individual and formulas are represented by their syntax trees. The syntax tree of the formula $x_0 < 10\ S\ _5(x_1 = 1 \wedge x_0 > 0)$ is shown in Figure 5.1

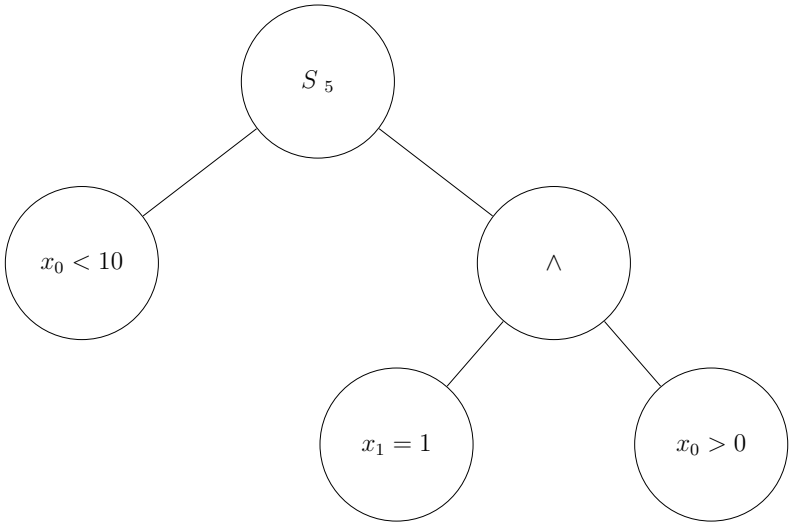

Figure 5.1: Syntax Tree for $x_0 < 10\ S\ _5(x_1 = 1 \wedge x_0 > 0)$

To generate the syntax tree, first the formula in infix form is converted to a formula in prefix form. For example, the prefix representation of formula $x_0 < 10\ S\ _5(x_1 = 1 \wedge x_0 > 0)$ is $S\ _5\ x_0 < 10\ (\wedge = x_1 1 > x_0 0)$. Then the syntax tree is constructed by following the algorithm given in Algorithm 2. From the beginning of the formula we obtain the operator and the parameters related to it as the root of the syntax tree and recursively generated child trees as the formula continues. Operands and Binary operators are given in table 5.2

Algorithm 2 $GenerateSyntaxTree(\tau, \phi, \mathbf{i}, \mu, \theta, \beta)$

INPUT: syntax_tree τ that is generated through recursive calls, prefix_formula ϕ, index **i**, list of metric operators μ, list of ptSTL operators θ, list of binary operators β
OUTPUT: post_index **j**

1: $items \leftarrow split(\phi)$;
2: **if** $\neg(items[0] \in \mu)$ **then**
3: $\quad \tau.operator \leftarrow items[\mathbf{i}+1]$
4: $\quad \tau.parameter \leftarrow items[\mathbf{i}+2]$
5: $\quad$ **return i** $+ 3$
6: **else**
7: $\quad \tau.operator \leftarrow items[\mathbf{i}]$
8: $\quad \mathbf{j} \leftarrow GenerateSyntaxTree(\tau.left\_child, items, \mathbf{i}+3, \mu, \theta, \beta)$
9: $\quad$ **if** $items[\mathbf{i}] \in \theta$ **then**
10: $\quad\quad \mathbf{j} \leftarrow GenerateSyntaxTree(\tau.right\_child, items, \mathbf{j}, \mu, \theta, \beta)$
11: $\quad$ **end if**
12: $\quad$ **if** $items[\mathbf{i}] \in \beta$ **then**
13: $\quad\quad \mathbf{j} \leftarrow GenerateSyntaxTree(\tau.right\_child, items, j, \mu, \theta, \beta)$
14: $\quad$ **end if**
15: $\quad$ **return j**
16: **end if**

5.2 Initial Population

Individuals (syntax trees) form the population. $\mathcal{G}_i$ is used to express the population in ith generation. Initial population should be crowded and diverse enough to represent individuals having different properties [15]. To achieve this, we first formalize the parametric formula space. Given a set of system variables $\mathcal{V}$ (metrics), the set of all parametric ptSTL formulas over $\mathcal{V}$ with oc operators is denoted by $\mathcal{F}^{oc}$ and recursively defined as:

Table 5.2: The operator classes that are used in ptSTL formulas

Operator	Binary Operator	STL Operator	Metric Operator
F^-	X	✓	X
G^-	X	✓	X
S	✓	✓	X
$\vee$	✓	X	X
$\wedge$	✓	X	X
$\neg$	X	X	X
$=$	✓	X	✓
$>$	✓	X	✓
$<$	✓	X	✓

$$
\begin{aligned}
\mathcal{F}^0 &= \{x \sim p_x \mid c \in \mathbb{R}, \sim \in \{<, >\}, x \in \mathcal{V}\} \cup \{true\} \\
\mathcal{F}^n &= \{\neg\phi \mid \phi \in \mathcal{F}^{n-1}\} \cup \{\mathbf{X}_{[p_a,p_b)}\phi \mid \phi \in \mathcal{F}^{n-1}, \mathbf{X} \in \{G^-, F^-\}\} \\
&\quad \bigcup_{i=1}^{n-1} \{\phi_1 \; op \; \phi_2 \mid \phi_1 \in \mathcal{F}^i, \phi_2 \in \mathcal{F}^{n-i-1}, op \in \{\vee, \wedge\}\} \\
&\quad \bigcup_{i=1}^{n-1} \{\phi_1 S_{\;[p_a,p_b)} \phi_2 \mid \phi_1 \in \mathcal{F}^i, \phi_2 \in \mathcal{F}^{n-i-1}\} \\
\mathcal{F}^{\leq n} &= \cup_{i=0}^{n} \mathcal{F}^i
\end{aligned}
\tag{51}
$$

where p_a, p_b and p_x denote the parameters and $\mathcal{F}^{\leq oc}$ denotes the set of parametric ptSTL formulas with at most oc operators. To form the initial population with n individuals, n formulas are sampled from this set and random values are assigned to each parameter.

5.3 Fitness and Selection

Fitness and selection are highly related concepts, since the selection function is chosen with respect to fitness. Hence the design decision about fitness directly effects the selection procedure also.

5.3.1 Fitness

Fitness is a value that represents how an individual is likely to be selected to reproduce in order to transfer its genes to the next generation. The fitness functions presented in Section 2.2.6, such as recall or precision, can be used in Algorithm 1. Any fitness function is applicable to our algorithm.

We simply count for 4 different cases that are introduced in Table 2.1, since our set-up introduces a dataset $\mathcal{D}$ 41 which contains different systems traces with timestamps and their labels. The computation of the total number of true positives, false positives, true negatives and false negatives over a trace T are shown in Equations 52,53 54, and 55 respectively.

$$C_{TP}(T, L^T, \phi) = \sum_{i=1}^{m^T} \begin{cases} 1 \text{ if } \rho(T, t_i, \phi) > 0 \text{ and } L^T(t_i) == 1 \\ 0 \text{ otherwise} \end{cases} \tag{52}$$

$$C_{FP}(T, L^T, \phi) = \sum_{i=1}^{m^T} \begin{cases} 1 \text{ if } \rho(T, t_i, \phi) > 0 \text{ and } L^T(t_i) == 0 \\ 0 \text{ otherwise} \end{cases} \tag{53}$$

$$C_{TN}(T, L^T, \phi) = \sum_{i=1}^{m^T} \begin{cases} 1 \text{ if } \rho(T, t_i, \phi) \leq 0 \text{ and } L^T(t_i) == 0 \\ 0 \text{ otherwise} \end{cases} \tag{54}$$

$$C_{FN}(T, L^T, \phi) = \sum_{i=1}^{m^T} \begin{cases} 1 \text{ if } \rho(T, t_i, \phi) \leq 0 \text{ and } L^T(t_i) == 1 \\ 0 \text{ otherwise} \end{cases} \tag{55}$$

The computation of the number of TP instances over a dataset $\mathcal{D}$ and formula ϕ is shown in (56). The number of instances over a dataset for other classes can be computed similarly for other classes.

$$\mathcal{C}_{TP}(\mathcal{D}, \phi) = \sum_{(T, L^T) \in \mathcal{D}} C_{TP}(T, L^T, \phi) \tag{56}$$

In other words, the success of a ptSTL formula ϕ does not only rely on a single trace T, it is computed over the whole dataset $\mathcal{D}$

5.3.2 Selection

In selection procedure, individuals that will transfer their genes to next generation are selected. Many procedures such as tournament selection, roulette wheel are already introduced [15] in literature. We used proportionate selection for formula synthesis problem to choose the formulas to transfer their genes to next generations, that are more likely to be the solution when choosing individuals.

The probability for each individual ϕ in generation $\mathcal{G}_i$ to be selected is defined as:

$$o_\phi = \frac{\mathcal{C}(\phi, \mathcal{D})}{\sum_{\varphi \in \mathcal{G}_i} \mathcal{C}(\varphi, \mathcal{D})} \tag{57}$$

$\mathcal{C}(\phi, \mathcal{D})$ is the fitness function for formula ϕ over dataset $\mathcal{D}$.

The overall flow of the selection procedure is shown in Algorithm 3. In line 3, the fitness values of the items in the set sums up to obtain the total fitness value of the set. Then in line 5, a random value between 0 and total fitness value is generated. Then fitness value of each item is subtracted from total fitness. Whenever a value less than or equal to generated number is obtained, the formula is chosen. As it can be observed higher fitness values results in higher probability to be chosen to reproduce.

Algorithm 3 *Selection*

INPUT: The set of formulas and their fitness values in generation
$G, S = \{(\phi, f) \mid \phi \in G, f = C(\phi, \mathcal{D})\}$
OUTPUT: Selected ptSTL formula ϕ

1: $\mathbf{f} \leftarrow 0$ (initialize total fitness to zero)
2: **for all** $(\phi^i, \mathbf{f}^i) \in \mathbb{S}$ **do**
3: $\quad \mathbf{f} \leftarrow \mathbf{f} + \mathbf{f}^i$
4: **end for**
5: $rand \leftarrow r \mid 0 <= r <= \mathbf{f}$
6: **for all** $(\phi^i, \mathbf{f}^i) \in \mathbb{S}$ **do**
7: $\quad \mathbf{f} \leftarrow \mathbf{f} - \mathbf{f}^i$
8: $\quad$ **if** $\mathbf{f} <= rand$ **then**
9: $\quad\quad$ **return** ϕ^i
10: $\quad$ **end if**
11: **end for**

5.4 Reproduction

Reproduction is the process of individuals to exposed to crossover and mutation to create a new generation. The individuals that are selected with respect to their fitness values are exposed to crossover and to mutation with a predefined probability.

5.4.1 Crossover

Crossover is the process of generating two new individuals (offsprings) from two individuals (parents). For syntax trees, we randomly chose a node from each parent syntax tree and swap the corresponding subtrees to form the offsprings. Since each subtree is a valid syntax tree, it is guaranteed that the offsprings represent valid ptSTL formulas. Crossover is demonstrated in the Algorithm 4 and the Figure 5.2 where randomly selected nodes that are highlighted with red and green, are swapped to form the individuals shown in the second row. In the algorithm random selection is shown

in lines 3 and 4. The "selectnode" function randomly selects a node of the input tree. After the selection selected nodes of each tree interchanges.

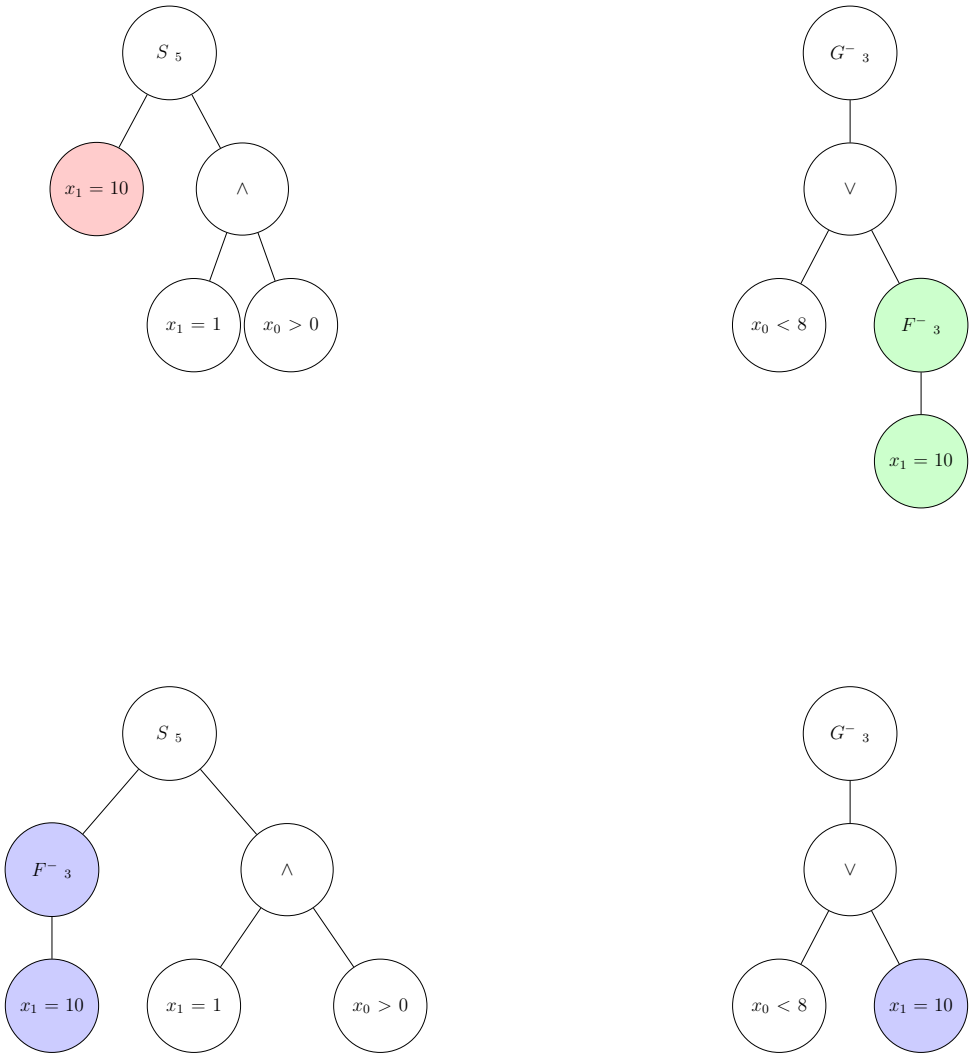

Figure 5.2: Generation of offsprings via crossover

Algorithm 4 $Crossover(\phi_1, \phi_2, \mu, \theta, \beta)$

INPUT: formula ϕ_1, formula ϕ_2, list of metric operators μ, list of ptSTL operators θ, list of binary operators β

OUTPUT: syntax tree τ_1, τ_2

1: $\tau_1 \leftarrow GenerateSyntaxTree(\phi_1, \mu, \theta, \beta)$
2: $\tau_2 \leftarrow GenerateSyntaxTree(\phi_2, \mu, \theta, \beta)$
3: $node_1 \leftarrow selectnode(\tau_1)$
4: $node_2 \leftarrow selectnode(\tau_2)$
5: $temp \leftarrow node_1$
6: $node_1 \leftarrow node_2$
7: $node2 \leftarrow temp$
8: **return** τ_1, τ_2

5.4.2 Mutation

Mutations are random alterations that occur in an individual. Mutations create small changes that can improve the fitness of the individual. To create such small changes for the formula synthesis problem, we defined a two-phase mutation process: operator mutation and parameter mutation. The operator mutations are defined for each operator. For temporal operators, G^- becomes F^-, and F^- becomes G^- when the operator is mutated. Note that in mutations for G^- and F^-, only the operator in the mutated node changes and the rest of the tree remains the same. On the other hand, for S operator, two types of mutations are defined considering its semantics. 1) It becomes G^- and the right subtree is dropped (G^- is unary, whereas S is binary) as a minimization over left formula is performed in the semantics of S that is similar to G^-. 2) It becomes F^- and left subtree is dropped and right subtree becomes left subtree as a maximization over the right formula is performed in the semantics of S that is similar to F^-. For inequalities (e.g. $x < c$), the direction of the inequality changes. The Boolean operators $\wedge$ and $\vee$ interchange for mutation.

The parameter mutations are defined for the thresholds of the inequalities and time bounds for the temporal operators. In this case, the new parameter is sampled from a

normal distribution whose mean is the previous value of the parameter. The operator and parameter mutations are summarized in Table 5.3.

Table 5.3: Parameter and operator mutations.

Node	Mutated Node		
	Parameter	Operator	
$x > c$	$x > c'$	$x < c$	—
$x < c$	$x < c'$	$x > c$	—
$\varphi_1 \wedge \varphi_1$	—	$\varphi_1 \vee \varphi_1$	—
$\varphi_1 \vee \varphi_1$	—	$\varphi_1 \wedge \varphi_1$	—
$\varphi_1 S_{\ p} \varphi_2$	$\varphi_1 S_{\ p'} \varphi_2$	$G^-_{\ p} \varphi_1$	$F^-_{\ p} \varphi_2$
$F^-_{\ p} \varphi$	$G^-_{\ p'} \varphi$	$G^-_{\ p} \varphi$	—
$G^-_{\ p} \varphi$	$G^-_{\ p'} \varphi$	$F^-_{\ p} \varphi$	—

For an individual, whether to be mutated or not is determined by a certain probability μ. If an individual is selected for mutation, a node is chosen randomly and it is either exposed to operator mutation or parameter mutation. An operator mutation is illustrated in Figure 5.3.

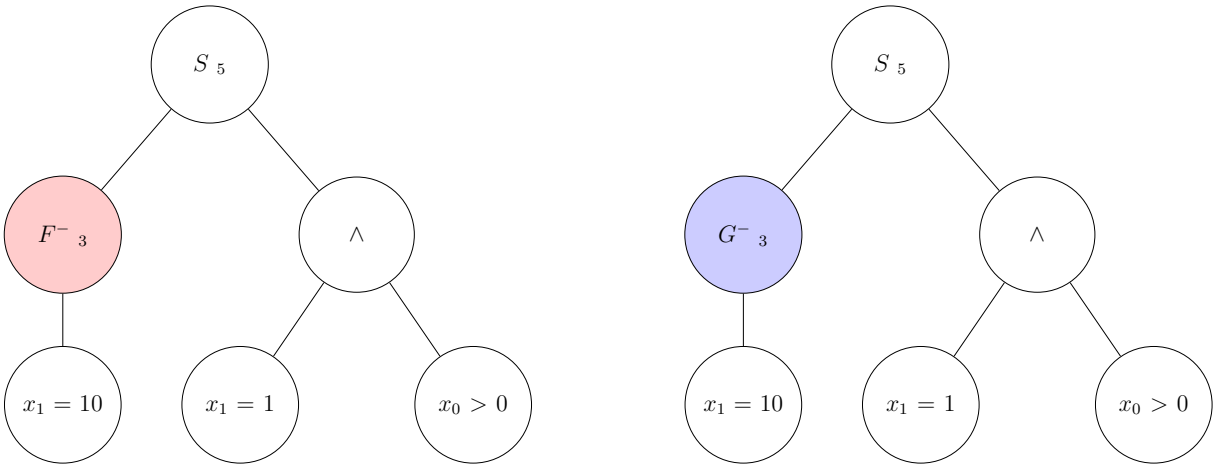

Figure 5.3: Operational Mutation

5.4.3 New Generation

A new generation is formed by iteratively selecting two individuals as parents from the previous generation, performing crossover to these individuals to generate offsprings and finally applying mutation to each offspring with probability μ. These steps are repeated until sufficient number of individuals are added to the next generation.

5.5 Termination

Genetic algorithms are highly randomized and non-deterministic algorithms. As convergence is not guaranteed, it is necessary to define termination conditions clearly. In our study, we define termination condition as having 1.0 fitness score (the highest value for considered fitness functions). For the cases in which the algorithm does not converge to the highest fitness value, we maintain the best fitness value among generations and terminate the algorithm when the best fitness does not change for a predefined number of iterations.

We run genetic algorithms on Problem 4.1.1. Details and results for the application is given in Example 5.5.1.

Example 5.5.1. We run Algorithm 1 on the dataset $\mathcal{D}$ for the CPU rate case in Example 4.1.1. The values for the parameters that is given is Table 5.1 is shown in Table 5.4. We have generated 400 different random formulas, in other words, 400 random formula out of all possible formula space is chosen as the initial population in genetic algorithms. We used defined F1-Score as the fitness function to avoid a bias towards minimization of false negatives and false positives. In reproduction, an individual can be picked more than once or it can be eliminated. After crossover, each individual is mutated with probability 0.1 ($\mu = 0.1$). The individuals that are selected for mutation are exposed to parameter mutation with probability 0.9 and operator mutation with probability 0.1.

The F1-Score for the best formula $\phi^{\star,1.1} = G^{-}_{\ [0,49)}(((x_0 > 18.0)S_{\ [0,41)}(x_0 > 30.67)) \vee (x_0 > 27.44))$ that is generated by Algorithm 1 is 0.904. The formula

is found in the 69th iteration and hold still till 119th iteration, the algorithm halts when the best fitness does not change for 50 iteration. The best fitness values over the generations is shown in Figure 5.4.

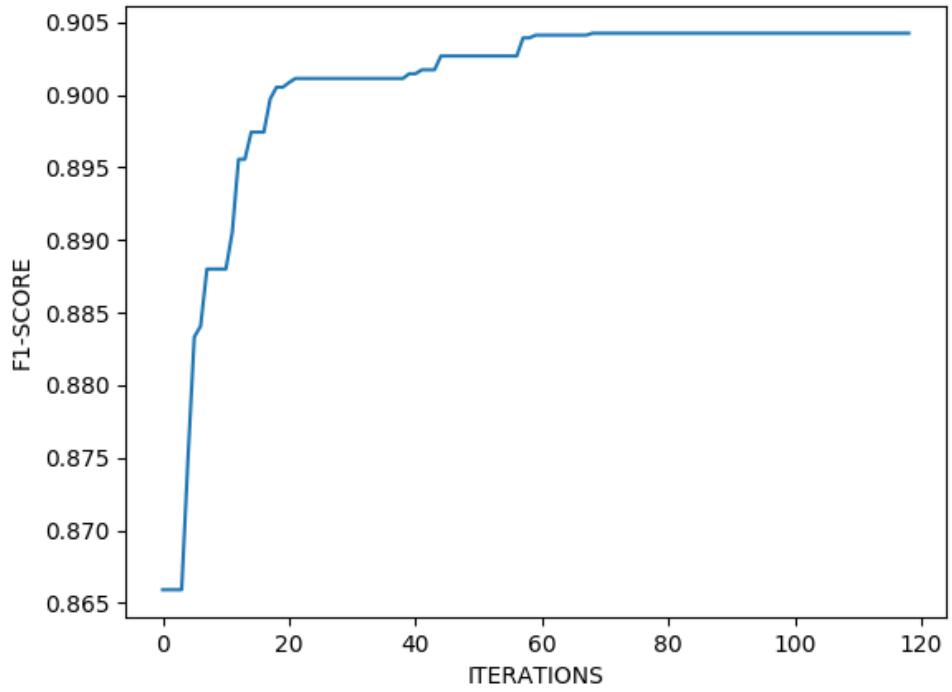

Figure 5.4: The F1-Score of the best formula over generations for pure genetic algorithms.

The overall computation took approximately 100 minutes on a PowerEdge T430 with single Intel Xeon E5-2650 12C/24T processor. As the optimization is based on parallel computing, the computation time can be reduced by increasing the processor cores.

The accuracy of the generated formula shows that the proposed synthesis approach of using genetic algorithms to synthesize ptSTL formulas is able to generate common monitoring rules in an automated way.

The same dataset was used in [44], where a grid search was used to synthesize a ptSTL formula instead of genetic algorithms. In addition, the number of mismatches between dataset labels and formula valuation (i.e. $\mathcal{C}_{FN}(\mathcal{D}, \phi) + \mathcal{C}_{FP}(\mathcal{D}, \phi)$) was used

as the formula fitness. To compare results with [44], we calculated the mismatch count for $\phi^{\star,1}$, which is 656, whereas it was 697 for the best formula over the same dataset. Although the difference seems negligible, it should be noted that the run time of the algorithm reduced significantly. The grid search algorithm [44] took 8 hours while the algorithm proposed here found the solution in 100 minutes. Moreover, the fitness function for our case is not chosen as minimum mismatch count. This is because of the fact that the dataset that we generated is not balanced and only 4% of the data points are labeled with 1. To use an unbiased method, we used F1-Score instead of mismatch count. The F1-Score for the best formula obtained with grid search method [44] is 0.89. In addition, when fitness is defined as mismatch count (with minimization), we were able to decrease the mismatch count to 555 with genetic algorithms [48]. To summarize, the increase in the fitness value and the significant decrease in the computation time show the effectiveness of the proposed genetic algorithm based method.

Table 5.4: Genetic algorithm parameters.

Parameter	Abbreviation	Parameter Domain
operator count	oc	$[0, 3]$
cpu signal	x_0	$[10, 40]$
Past Globally Parameter Domain	G^-	$[20, 80]$
Past Eventually Parameter Domain	F^-	$[20, 80]$
Since Parameter Domain	S	$[20, 80]$

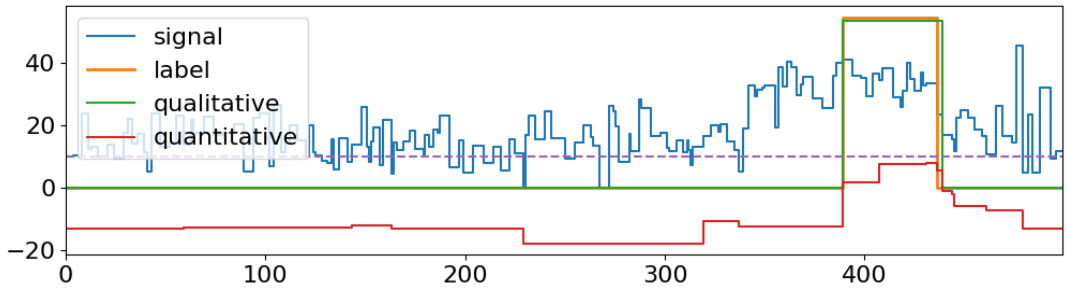

Figure 5.5: A sample signal for Case Study 1, its label, quantitative and qualitative valuations for formula $\phi^{\star,1}$. The label and quantitative valuations are scaled for better view.

CHAPTER 6

LOCAL OPTIMIZATION FOR PARAMETER SYNTHESIS

In order to solve STL synthesis problem, there exists studies in the literature that takes a different track and optimize parameters of an STL formula whose template is predefined by an expert of the domain[23]. In this dissertation, we also studied the parameter synthesis problem. As in STL synthesis problem, most of the studies focused on signals that have a single label defining the good or bad behavior of the the signal [18, 9, 23]. We propose a parameter synthesis method for signals that have label at each time point.

In the proposed method, we randomly generate values for time bounds of temporal operators and constants in the inequalities. Then iteratively, we slightly change these values to obtain better results. Considering the run time performance of the proposed method we offer three versions of the solution. All of them, generate a tree-based structure whose root is the initial formula, but have different strategies for expanding and pruning the tree.

Local optimization begin with the initialization phase. Each and every parameter in the template formula is initialized to a random variable. An expert of the system can assign likely values to the parameters as well. Afterwards, the parameter values are incremented/decremented by a predefined step size to generate new STL formulas. The new STL formulas are added as child nodes to form the search tree. Essentially, in a single step of the search, $2 \times m$ new formulas can be obtained from a formula with m parameters. These new formulas are added to the tree as the child nodes of the initial formula. The procedures are repeated for each of the new $2 \times m$ formula to downwards. A sample tree is shown in Figure 6.1 for the formula template $\mathbf{A}_{p_1}(x < p_2)$, initialized with $p_1 = 40, p_2 = 20$.

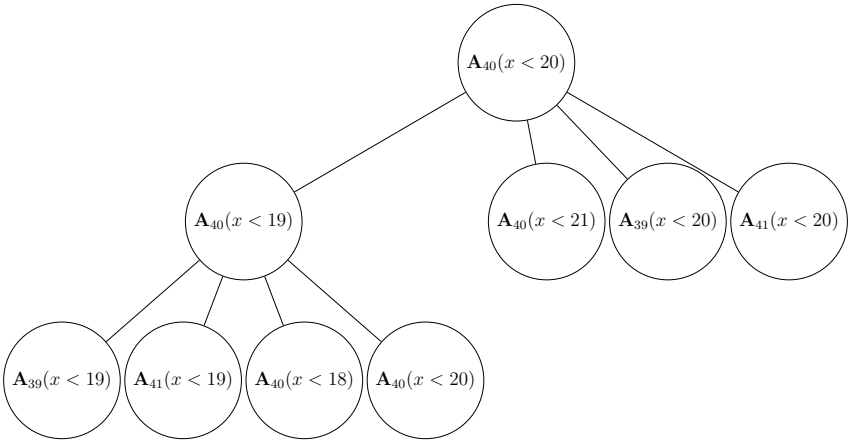

Figure 6.1: Local Search Tree

The variations of the local search method is generated with respect to the pruning strategy. These variations are local search, restricted local search and guided search.

6.1 Local Search

Local search is the base version. There is no restriction and the search tree grows in every direction. The only pruning strategy that applied in this version is to prune the nodes which are already visited. Other than that, starting from the initial formula, the tree expands to the each and every child node. Considering the run time performance, a maximum depth limit is set to stop the grow of the search tree.

Evaluation techniques such as precision, recall or F-1 score are not monotonous in parameter domain. Hence the algorithm moves on both directions on the domain of the each parameter to find a local optimum value. However, the cost of evaluating the STL formula over the signal set is very high. In the local search method with no restriction, the number of evaluation of the STL formula is exponential in the depth of the tree. An STL formula with m parameters have to be evaluated with different parameters for $\frac{(2m)^{h+1}-1}{2m-1}$ times with iteration limit h. This high costs drives us to find more efficient local parameter search methods.

We run local search on Problem 4.1.1. Details and results for the application is given in Example 6.1.1.

Example 6.1.1. From Example 5.5.1, we know that the parametric formula of the solution is in the form:

$$\phi = G^{-}{}_{[0,s_1)}(((x_0 > p_1) S{}_{[0,s_2)}(x_0 > p_2)) \vee (x_0 > p_3)) \tag{61}$$

So we randomly initialized the parameters with values given in the Table 6.1 to obtain the initial formula. So our initial formula become $\phi^{\star,1.2} = G^{-}{}_{[0,5)}(((x_0 > 11) S{}_{[0,12)}(x_0 > 24)) \vee (x_0 > 21))$. We run local search with maximum depth limit of 10. Local search obtained the formula $\phi^{\star,1.2.2} = G^{-}{}_{[0,27)}(((x_0 > 11) S{}_{[0,12)}(x_0 > 24)) \vee (x_0 > 21))$. To obtain the formula, algorithm evaluates a candidate formula over 500 trace dataset 5730 times and it lasts 5182 seconds. F1-Score of the obtained formula is 0.534 where the F1-Score for the initial formula is 0.271.

Table 6.1: Randomly initialized values for parametric formula 61

Parameter	Value
p_1	11
p_2	24
p_3	21
s_1	5
s_2	12

6.2 Restricted Local Search

In this version of the local search, in each iteration, before expanding the search tree, the leaf nodes (formulas) whose fitness score is less then their ancestor are pruned. Thus the search only explores the "promising" directions with an increase on the fitness value instead of all directions.

As seen in line 5 of Algorithm 5, a node is added if its valuation is higher than its ancestor. Note that, without the condition in line 5, we obtain the unrestricted local search defined in the previous section. Even though only a fraction of the full search tree is explored, the complexity of the restricted search is still exponential with the

Algorithm 5 RestrictedLocalSearch($\phi, \mathbf{v}, \mathbf{d}, h$)

Input: Template formula ϕ, parameter initialization vector $\mathbf{v} = [v_1, \ldots, v_m]$, step size vector $\mathbf{d} = [d_1, \ldots, d_m]$, iteration limit h

Output: Formula tree, one of the leaf node is the optimum formula

1: $root = \phi(\mathbf{v})$ (formula is generated by assigning the related values to the parameters)
2: **if** h=0 **then return** root
3: **for** $i = 1$ to m and $\sim \in \{-, +\}$ **do**
4: $\quad \mathbf{v}' = \mathbf{v} \sim (\mathbf{d} * e^i)$ (e^i, vector only the ith input is 1)
5: $\quad$ **if** $Success(\mathbf{v}') > Success(\mathbf{v})$ **then**
6: $\quad\quad root.appendChild(RestrictedLocalSearch(\phi, \mathbf{v}', \mathbf{d}, h-1))$
7: $\quad$ **end if**
8: **end for**
9: **return** root

iteration limit. However, in each iteration it is expected to prune the half of the children, which results in an increase in run time efficiency.

Example 6.2.1. For restricted local search, we used the same template formula that is given in Equation 61 and initialized the parameters with values given in Table 6.1. In other words, again we gathered the initial formula $\phi^{\star,1.2} = G^{-}{}_{[0,5)}(((x_0 > 11)S_{[0,12)}(x_0 > 24)) \vee (x_0 > 21))$. We run restricted local search with maximum depth limit of 10. The construction of the search tree lasted for 2766 seconds and restricted local search algorithm obtained the formula: $\phi^{\star,1.2.2} = G^{-}{}_{[0,27)}(((x_0 > 11)S_{[0,12)}(x_0 > 24)) \vee (x_0 > 21))$. To obtain the formula, algorithm evaluates a candidate formula over 500 trace dataset 1438 times. F1-Score of the obtained formula is 0.534 where the F1-Score for the initial formula is 0.271.

6.3 Guided Search

The guided search method is similar to the restricted local search method. In the guided search the aim is also to move towards to the promising directions. However,

for a child node to be defined as a promising direction, unlike restricted local search, it is not enough to have a better fitness than its ancestor. In guided search, a promising child is also a better fit than its siblings. This means that, in every iteration, among the child nodes, the one with the highest success value survives and all of the other child nodes are being pruned. Hence, the structure of the tree is just look like a path instead of a tree. Figure 6.2 is an example for a template formula $\mathbf{A}_{p_1}(x < p_2)$, being initialized with the values $p_1 = 40$ ve $p_2 = 20$.

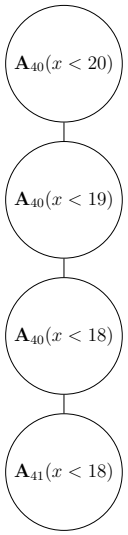

Figure 6.2: Guided Search

Example 6.3.1. Unlike local search and guided local search, we did not define a maximum depth limit for guided search since we expect guided search to be much more efficient. We used the same template formula that is given in Equation 61 and initialized the parameters with values given in Table 6.1. The resulting formula is $\phi^{\star,1.2.3} = G^{-}{}_{[0,49)}(((x_0 > 18)S{}_{[0,12)}(x_0 > 25)) \vee (x_0 > 25))$, and its F-1 score is 0.900

Guided search took 35 evaluations to obtain the parameters. It took 230 seconds to compute the 35 evaluations in the same computer as guided and restricted guided search methods. The number of iterations vs. F1-Score graph of the guided search for CPU rate case is given in Figure 6.3.

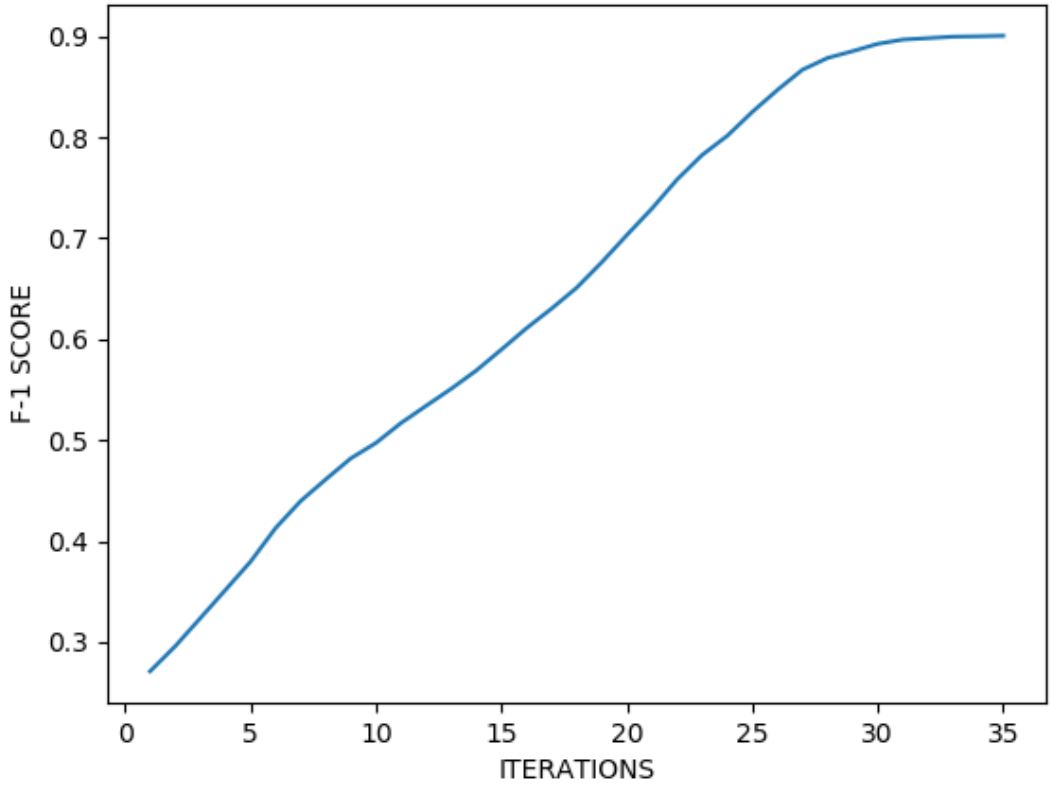

Figure 6.3: The F1-Score of the formulas generated by guided search.

CHAPTER 7

GENETIC ALGORITHMS FOR STL SYNTHESIS EMPOWERED WITH LOCAL OPTIMIZATION

A novel contribution of this thesis is to integrate the local search to the proposed formula synthesis technique which is based on genetic algorithms. This is called the adaptation step. In biology adaptation is defined as a feature of an organism which improves success in reproductions, and evolved by natural selection [49]. As in biology we also define adaptation as adaptive modifications in genome, when the individual meets the environment. Just like the biology, in our problem, after the mutation state, each individual develops an adaptation to the environment. This adaptation demonstrates itself as local parameter optimization.

The main idea is to find local optimums quickly, and continue to generate new generations from individuals which are better fits to their environment. Genetic adaptation provides better fitness to the environment whereas it may diminish fitness to the other environments [49]. Moreover, it takes place after the genotype is formed. Considering these two key concepts, we find it suitable to name this phase as adaptation.

All tree local optimization methods introduced in Chapter 6 are suitable for adaptation. Considering the efficiency we chose guided search, but it is also possible to adapt either local or restricted local search. While applying the local optimization, we did not separate the procedure from the flow of the genetic algorithms. Hence, we integrated the adaptation process into genetic algorithms. The overall flow of the genetic algorithms with the adaptation phase is presented by Algorithm 6. We simply add a new step as adaptation in line 7 after the execution of mutation phase. The other phases such as crossover or mutation are the same as in the Chapter 5

Algorithm 6 Genetic Algorithm Empowered with Local Optimization

1: Initialization: Start with a randomly generated population of n chromosomes
2: Compute fitness: Compute the fitness of each chromosome x in the population.
3: **while** A certain number of steps or until the fitness converges **do**
4: Selection
5: Crossover
6: Mutation
7: Adaptation
8: Create the new generation
9: Fitness computation of the new generation.
10: **end while**

We run genetic algorithms empowered with local optimization on Example 4.1.1. Details and results for the application is given in Example 7.0.1.

Example 7.0.1. We used same dataset as in 5.5.1 for the genetic algorithms empowered with local optimization. Since algorithm still uses genetic algorithms, we also run Algorithm 6 with the parameters in Table 5.4. We have generated 400 different random formulas, in other words, 400 random formula out of all possible formula space is chosen as the initial population in genetic algorithms empowered with local optimization also.. We used defined F1-Score as the fitness function to avoid a bias towards minimization of false negatives and false positives.

The F1-Score for the best formula $\phi^{\star,1.3} = (x_0 > 23.19) \vee G^{-}{}_{[0,50)}(((x_0 > 15.54) S_{[0,22)} (x_0 > 28.04)))$ generated by Algorithm 6 is 0.920. The formula is found in the 266th iteration and hold still till 316th iteration, the algorithm halts when the best fitness does not change for 50 iteration. The algorithm ends up with a better score than the algorithm that occupies pure genetic algorithms, but also it takes more iteration to converge. Hence it is better to compare the Algorithm 1 and the Algorithm 6 with respect to their access time to same results. Algorithm 6 gathers the F1-Score of 0.904 at iteration number 25 whereas Algorithm 1 gets the same result at 69th iteration. Hence, although the algorithm works non deterministic and these results can have been change slightly in each run, genetic algorithms empowered with local optimization is able to produce formulas with higher fitness scores faster compared to Algorithm 1.

The best fitness values over the generations for genetic algorithms empowered with local optimization is shown in Figure 7.1.

The overall computation took approximately 354 minutes on a PowerEdge T430 with single Intel Xeon E5-2650 12C/24T processor. As the optimization is based on parallel computing, the computation time can be reduced by increasing the processor cores.

The accuracy of the generated formula show that the proposed synthesis approach can generate common monitoring rules in an automated way.

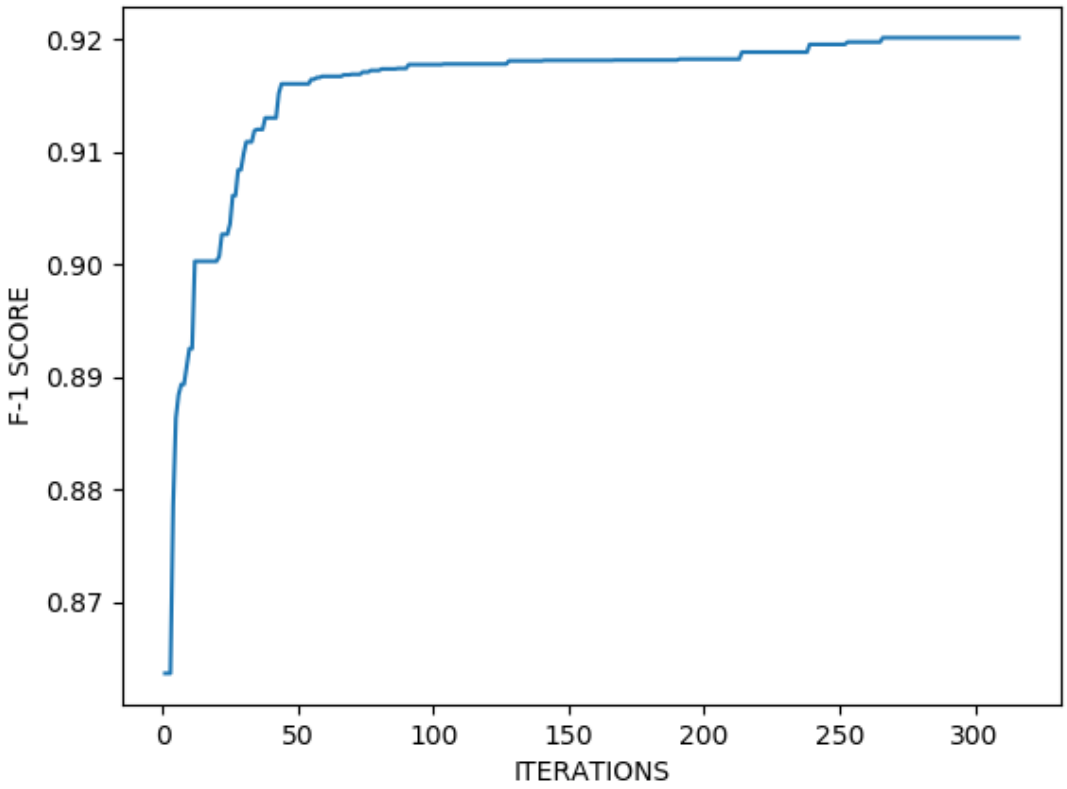

Figure 7.1: The F1-Score of the best formula over generations for genetic algorithms empowered with local optimization.

CHAPTER 8

EXPERIMENTS AND RESULTS

All computations are performed on a PowerEdge T430 with single Intel Xeon E5-2650 12C/24T processor. As the optimization is based on parallel computing, the reported computation times can be reduced by increasing the processor cores.

8.1 Case Study 1: Restart Rate

As a case study, our aim is to find a ptSTL formula which will be serve as a monitoring rule for the restart rate of set of servers. We used restart rate(r) and on-going system update(push p), since the restart rate is expected to increase during the system updates. As in the Example 4.1.1, we generated dataset with labels randomly. For each trace, we randomly picked the following for a system update: a high rate $r^p \in [20, 30]$ (uniform distribution), push duration $d^p \in [30, 50]$, and start time st^p. We also randomly picked the following for the case of a high update rate without system update (violation): a high rate $r^v \in [15, 25]$, a duration $d \in [30, 50]$, and start time st with $[st^p, st^p + d^p] \cap [st^v, st^v + d] = \emptyset$. The following rules are used to generate the data points according to the time stamps t (l is label):

$$
\begin{aligned}
t \in [st^p, st^p + d^p] &: l = 0, p = 1, r = r^p \\
t \in [st^v, st^v + d] &: l = 1, p = 0, r = r^v \\
\text{otherwise} &: l = 0, p = 1, r \in [0, 10] \text{ with probability } 0.04 \\
& \qquad\quad r = 0 \text{ with probability } 0.96
\end{aligned}
$$

8.1.1 Using Pure Genetic Algorithms

We generated 500 labeled traces for dataset $\mathcal{D}$ and run Algorithm 1 with $oc = 2$ and the following parameter domains: $p_r \in \{6 + 2i \mid i = 0, \ldots, 10\}$ for linear predicates over r, $p_p \in \{0, 1\}$ for linear predicates over p, $p_b \in \{4 + 4i \mid i = 0, \ldots, 12\}$ for $G^{-}{}_{[0,p_b)}$, $F^{-}{}_{[0,p_b)}$, and $S_{[0,p_b)}$. In this case study, we generated an initial population with 40 individuals. In three iterations the Algorithm 1 generated a formula with 1.0 F-1 Score and terminated. The formula is: $\phi^{\star,2.1} = ((p < 0.68) \wedge (r > 11.0)) \vee (r > 32.0)$. Notice that $(r > 11.0) \vee (r > 32.0)$ semantically equals to $(r > 11.0)$. The computation took 11 seconds on the same computer as case study 1. The valuation of $\phi^{\star}$ over a sample trace is shown in Figure 8.1. This example shows that the proposed unsupervised synthesis approach is capable of identifying the relations for a multi-dimensional example.

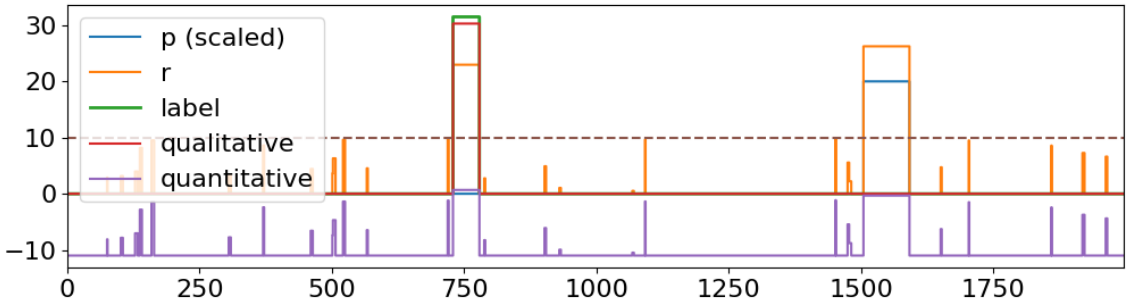

Figure 8.1: A sample signal for Case Study 2. The label, quantitative and qualitative valuations for formula $\phi^{\star,2.1} = ((p < 0.68) \wedge (r > 11.0)) \vee (r > 32.0)$, where r stands for restart rate and p stands for push. p value, label and qualitative valuations are scaled as well for better view.

8.1.2 Using Local Search

Although the first step of the algorithm is random initialization of the parameters of the template formula, in order to be fair to all three approaches, we used the same initial values of the parameters for all three versions which is $\phi\prime = (p < 0) \wedge (r > 3)$.

8.1.2.1 Local Search

Local search synthesized the formula $\phi^{\star,2.2.1} = (p < 1) \wedge (r > 11)$ whose F1-Score is 1.0. Local search took 9792 evaluations to obtain parameters. It took 15615 seconds to compute the 9792 evaluations. The maximum depth limit for local search is 10, however in this case, the algorithm is able to generate the optimum formula before exceeding the maximum depth limit.

8.1.2.2 Restricted Local Search

Restricted local search synthesized the formula $\phi^{\star,2.2.2} = (p < 1) \wedge (r > 11)$ whose F-1 Score is 1.0. Restricted local search took 9 evaluations to obtain parameters. It took 18 seconds to compute the 9 evaluations. The maximum dept limit of 10 is not exceeded and the algorithm reached the optimum solution, as in local search.

8.1.2.3 Guided Search

Guided search synthesized the formula $\phi^{\star,2.2.2} = (p < 1) \wedge (r > 11)$ whose F-1 Score is 1.0. Guided search took 8 evaluations to obtain the parameters. It took 17 seconds to compute the 8 evaluations. For the restart rate case number of iterations are the same for guided and restricted local search. This is because of the fact that there is no promising path in the search tree except the one guided search occupies. The Iteration F1-Score graph of the guided search for restart rate case is given in Figure 8.2.

8.1.3 Using Genetic Algorithms Empowered With Local Optimization

As in CPU case, we also used both pure genetic algorithms and genetic algorithms empowered with local optimization for restart rate case also. Algorithm 6 generated a formula with 1.0 F-1 Score and terminated in the first iteration. The generated formula is $\phi^{\star,2.3} = ((p < 1.0) \wedge (r > 11.0))$. As a results, empowerment of genetic algorithms with local optimization is efficient for multi-dimentional examples also.

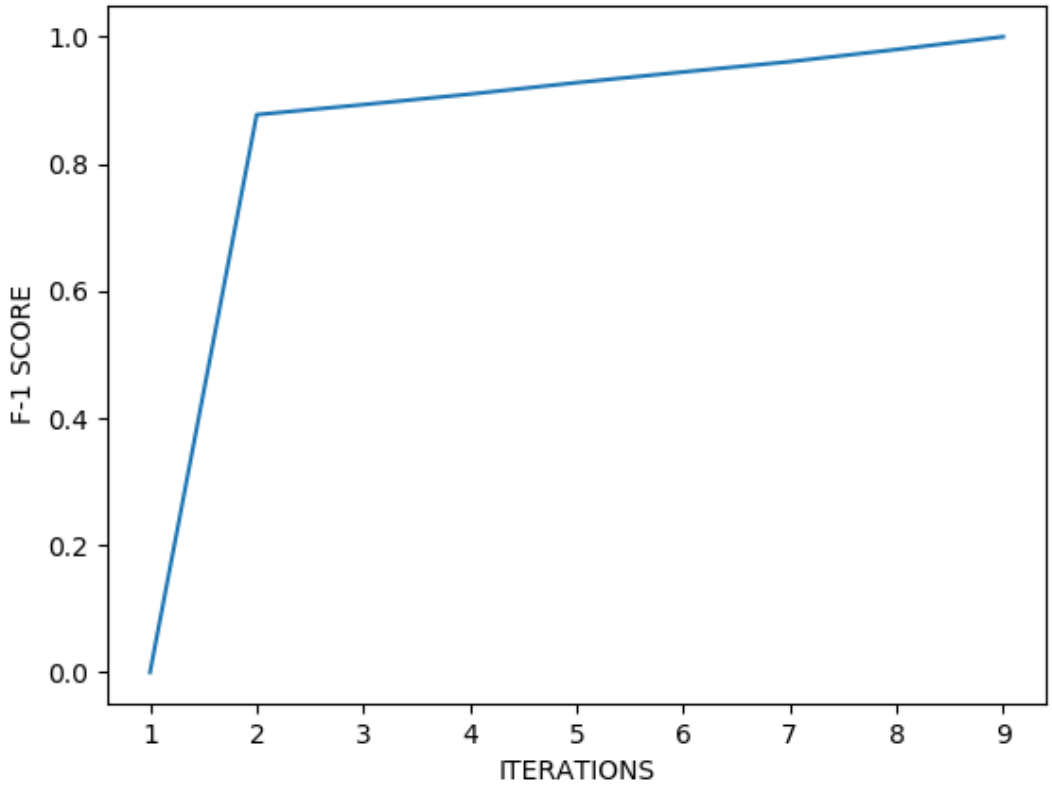

Figure 8.2: The F1-Score of the formulas generated by guided search.

8.2 Case Study 2:Ports Dataset

In this case study we used Open Research Maritime Vessl Tracking Dataset which is provided by Marine Traffic [50]. The dataset contains maritime surveillance data of the ships which are transporting in the Mediterranean Sea. There are total of 775 trips in the data set, which have destination port as an attribute. We aimed to synthesize a ptSTL rule for the ships that are headed to Tuzla port as soon as possible.

The dataset set contains "speed", "longitude", "latitude", "course", "heading", "ship id", "ship type", "departure port name" and "reported draught" attributes. Duration between two consecutive timestamps is one minute. Since, our aim is to generate a rule for the ships that heads to Tuzla port, we labeled second half of the datapoints in a trip to Tuzla with 1, and 0 otherwise.

The same dataset is also used in [47] to generate a ptSTL formula which defines ships that heads to Tuzla port. Considering the results in [47], we used F-10 Score instead of F-1 Score. Because of the fact that, Ketenci and Gol used TP-rate for fitness function, we also chose our fitness function in favor of sensitivity (recall). Hence, we used F-10 score which gives more importance to recall than precision to compare the results with [47].

In order to run Algorithm 1 and Algorithm 6 for the ports dataset, the values for the parameters that is given is Table 5.1 is shown in Table 8.1

The domains of the parameters that are used for parameter synthesis for ports data set are given in Table 8.1

Table 8.1: Genetic algorithm parameters.

Parameter	Abbreviation	Parameter Domain
operator count	oc	$[0, 3]$
speed	x_0	$[5, 55]$
longitude	x_1	$[5, 55]$
latitude	x_2	$[5, 55]$
course	x_3	$[5, 100]$
heading	x_4	$[5, 500]$
Past Globally Parameter Domain	G^-	$[10, 110]$
Past Eventually Parameter Domain	F^-	$[10, 110]$
Since Parameter Domain	S	$[10, 110]$

8.2.1 Using Pure Genetic Algorithms

The F10-Score for the best formula $\phi^{\star,3.3.1} = (x2 > 24.0)S_{\ [0,86)}(G^-_{\ [0,86)}(G^-_{\ [0,86)}((x2 < 29.509) \wedge (F^-_{\ [0,86)}(x1 > 35.0)))))$ generated by Algorithm 1 is 0.978, where $x1$ represents latitude attribute while $x2$ represents longitude. The recall of the solution is 0.999 while precision is 0.315. Genetic algorithms reached the formula in 109th iteration and hold still till 139th iteration. The best fitness values over the generations is shown in Fig. 8.3.

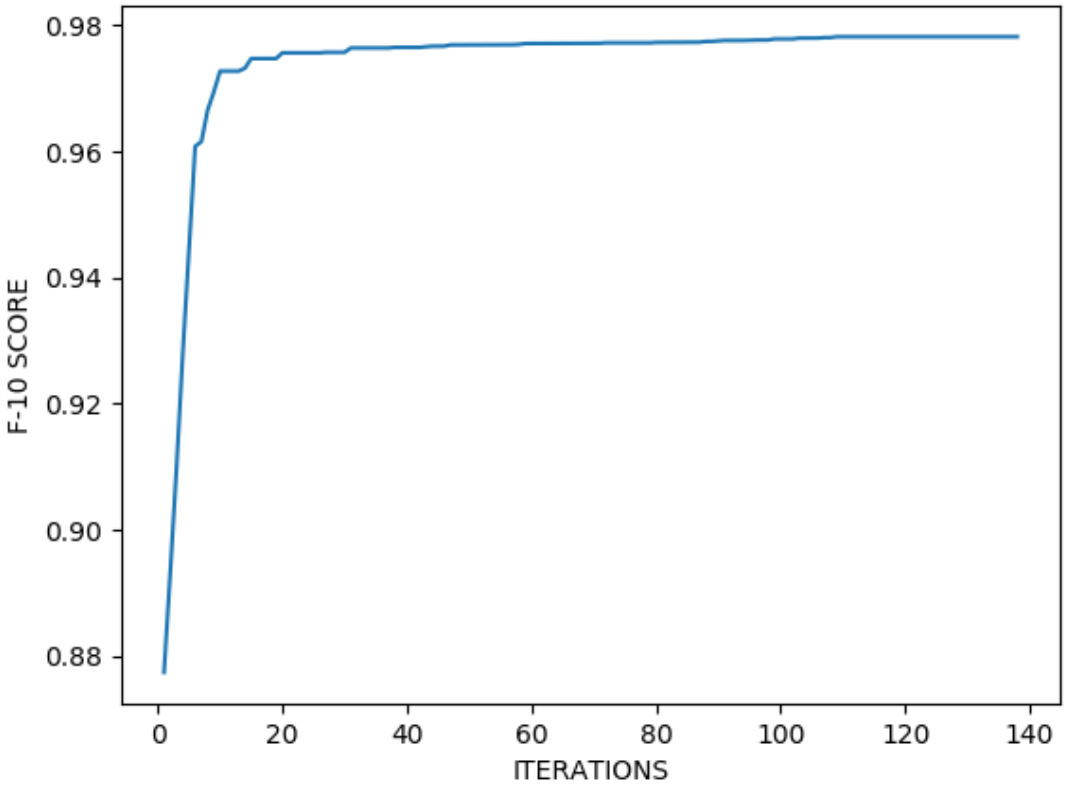

Figure 8.3: The F10-Score of the best formula over generations for genetic algorithms.

The same dataset was used in [47]. They developed a decision tree based model to generate formula $(G^{-}{}_{[0,100)}x2 > 25.36) \wedge (F^{-}{}_{[0,80)}x2 < 29.44) \wedge (G^{-}{}_{[0,100]}x1 < 40.88)$ and its F-10 Score is 0.958. The precision of the solution is 0.318 while the recall is 0.978. Considering the accuracy of both algorithms, genetic algorithms are also capable of generating ptSTL formula with respect to different fitness functions.

Comparing the results with the study [47], F-10 score and recall of the solution is far better than [47], on the other hand precision of two solutions are nearly the same. Hence, we demonstrated that genetic algorithms solution is valid for not only synthetic datasets, but also real datasets. The overall computation took approximately 3442 minutes.

8.2.2 Using Local Search

The parametric formula is given in the following form:

$$\phi\prime = (x2 > p_1)S_{[0,s_1)}(G^{-}_{[0,s_2)}(G^{-}_{[0,s_3)}((x2 < p_2) \wedge (F^{-}_{[0,s_4)}(x1 > p_3))))) \quad (81)$$

And we initialized parameters with values given in the Table 8.2 to obtain the initial formula. So our initial formula become $\phi\prime = (x2 > 20)S_{[0,20)}(G^{-}_{[0,17)}(G^{-}_{[0,17)}((x2 < 34) \wedge (F^{-}_{[0,23)}(x1 > 27)))))$, and its F-10 Score is 0.954.

Table 8.2: Randomly initialized values for Parametric Formula 81

Parameter	Value
p_1	20
p_2	34
p_3	27
s_1	20
s_2	17
s_3	17
s_4	23

8.2.2.1 Local Search

The maximum depth limit for local search is 5. Local search obtained the formula $(x2 > 24)S_{[0,20)}(G^{-}_{[0,17)}(G^{-}_{[0,17)}((x2 < 32) \wedge (F^{-}_{[0,23)}(x1 > 27)))))$ after evaluating 1313 different candidate formulas. The whole computation took 8512 seconds and F-10 score of the obtained formula is 0.971

8.2.2.2 Restricted Local Search

The maximum depth limit for restricted local search is also 5. To build the search tree, restricted local search evaluated a candidate formula for 201 times. The resulting ptSTL formula is $(x2 > 24)S_{[0,20)}(G^{-}_{[0,17)}(G^{-}_{[0,17)}((x2 < 32) \wedge (F^{-}_{[0,23)}(x1 >$

$27)))))$. The F-10 Score of the obtained formula is 0.971. The process took 2317 seconds to complete.

8.2.2.3 Guided Search

The maximum depth limit for guided search is also 5. Guided search calculated the F10 score of a canditate 65 times and obtains the results $(x2 > 24)S_{[0,20)}(G^{-}_{[0,17)}(G^{-}_{[0,17)}((x2 < 32) \wedge (F^{-}_{[0,23)}(x1 > 27)))))$ whose F-10 score is 0.971. The process took 1047 seconds to complete.

8.2.3 Using Genetic Algorithms Empowered With Local Optimization

The F10-Score for the best formula $\phi^{\star,1.3.3} = ((x_2 < 30.0) \wedge (((x_1 > 34.0) \wedge (x_2 > 25.0))|(x_2 > 24.0)))S_{[0,99]}(x1 > 35.0497)$ generated by Algorithm 6 is 0.976. The recall value is 0.998 and precision is 0.304. The formula is found in the 19th iteration and hold still till 49th iteration, the algorithm halts when the best fitness does not change for 30 iteration. The algorithm ends up with a slightly worse score than the algorithm that occupies pure genetic algorithms, but also it takes less iteration to converge. Hence it is better to compare the Algorithm. 1 and the Algorithm 6 with respect to their access time to same results. Algorithm 6 gathers the F10-Score of 0.976 at iteration number 19 whereas Algorithm 1 gets the same result at 31st iteration. Hence, although the algorithm works non deterministic and these results may have been change slightly in each run, it may be concluded that local optimization empowers genetic algorithms to reach local minimum/maximums to diminish number of generations.

The best fitness values over the generations for genetic algorithms empowered with local optimization is shown in Figure 8.4.

The accuracy of the generated formula show that the proposed synthesis approach can generate common monitoring rules in an automated way.

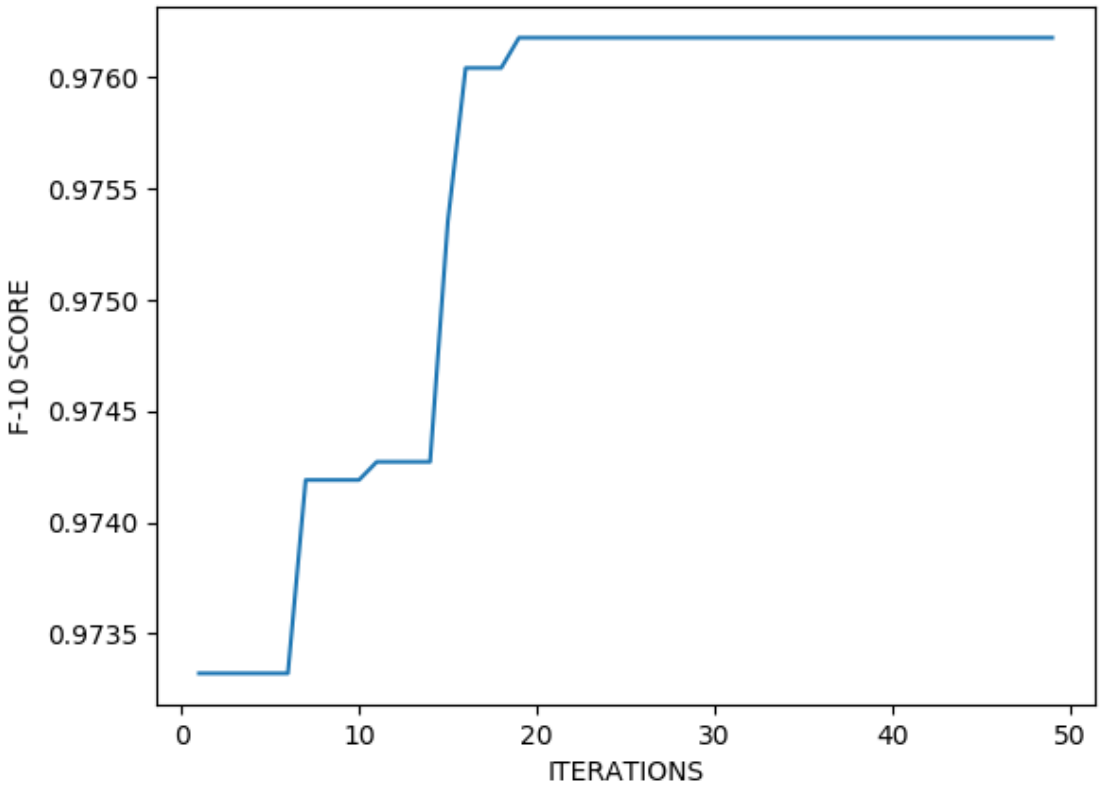

Figure 8.4: The F10-Score of the best formula over generations for genetic algorithms empowered with local optimization.

CHAPTER 9

CONCLUSION

In this dissertation, we focused on synthesizing temporal logic formulas to define given labeled dataset. We mainly studied on the ptSTL extension of temporal logic, in which the bounds of the temporal operators are defined in past time. We presented two main approaches on solving the formula synthesizing problem. Firstly, we proposed a genetic algorithm based solution in order to obtain a formula structure and parameters related to it. Secondly, we developed a method to optimize parameters of a given STL formula template, which occupies three different local search versions. Moreover, we integrated two different research topics and proposed a contribution to genetic algorithms method with "adaptation" empowerment.

In genetic algorithms, we adapt concepts of genetic algorithms such as representation, fitness function, selection, crossing over, mutation and generation into our problem domain. Since the temporal logic formulas can be represented by their syntax trees, we chose a tree base chromosome representation for ptSTL formulas. The genetic algorithms method we proposed is capable of working on any well defined fitness function. The most widely used fitness functions in literature are precision, recall and f-score. In our case studies, we used f-score to show that, in different domains it is possible to be on the side of recall or precision. For instance, in a fire detection system, it is vital not to miss any hazardous situation; however, it may not be very important to have false alarms time to time. So, it is possible to have a sensitive ptSTL rule by just increasing the β value in f-score.

For selection, we used proportionate selection to select the ptSTL formulas, that are likely to be the solution for the given dataset, to pass their genetic codes to the new generations. We defined crossing over as a swap operation between randomly se-

lected branches of syntax trees of ptSTL formulas, and mutation operation occurs as a random modification in a branch's operation or parameter value.

Parameter optimization method we proposed works on a search tree, whose root is a random initialization of the predefined ptSTL formula template. In each depth, we tune parameters to find better solutions to the problem. Considering the performance of the algorithm we developed three different versions all of which occupies different expanding and pruning strategies.

Lastly, we integrated the two solutions we made under the name of "Genetic Algorithms Empowered with Local Optimization". Our purpose is to reach local maximum/minimum values rapidly to diminish numbers of generations in genetic algorithms. Moreover, we wanted the eliminate the possibility of loosing promising formula templates because of the bad parameter assignment. We also discovered that, the method we proposed is interior to the evolutionary process and we called the step as "Adaptation".

In our case studies, we demonstrated that usage of the genetic algorithms for STL formula problem is an efficient solution. There were studies in the literature which treats formula synthesis problem as a classification problem, but most of them implements decision tree based solutions. To the best of our knowledge, when we proposed genetic algorithms for formula synthesizing problem in [48], there were no such method in literature. Afterwards, Nenzi et.al.(2018) [51] suggest the usage of genetic algorithms, there were very few details on the implementation of the genetic algorithms and the benefits.

Another contribution of this thesis is that we developed a simple parameter optimization method. The method we proposed works on a search tree to obtain parameters efficiently. We first proposed the method in [52]. It is an end to end solution for parameter optimization for STL formulas problem; on the other hand, it can work favor of a method that synthesize the ptSTL formula and parameters together to increase efficiency. To show so, we integrated our solution with genetic algorithms solution and demonstrated the decrease in number of generations.

The last but not the least, we proposed a new phase to the genetic algorithms. In

biology evolutionary process continues after the birth with adaptation. Unlike mutation, living creatures adapts to the nature after realizing the environmental settings. To adapt such concept in genetic algorithms, we offer an optimization considering the dataset that we try to fit in. We called the new phase as adaptation because of the fact that, both processes are similar in biology and computer science, and as far as we know there were no such concept in genetic algorithms before.

REFERENCES

[1] A. Birolini, *Reliability Engineering, Theory and Practice*. Springer-Verlag Berlin Heidelberg, 2014.

[2] E. M. Clarke and J. M. Wing, "Formal methods: State of the art and future directions," *ACM Comput. Surv.*, vol. 28, no. 4, pp. 626–643, Dec. 1996. [Online]. Available: http://doi.acm.org/10.1145/242223.242257

[3] A. Donzé, T. Ferrère, and O. Maler, "Efficient robust monitoring for STL," in *CAV 2013. LNCS vol 8044*, N. Sharygina and H. Veith, Eds. Springer Berlin Heidelberg, 2013, pp. 264–279.

[4] A. Donzé, "On signal temporal logic," in *RV 2013, LNCS 8174.*, A. Legay and S. Bensalem, Eds. Springer Berlin Heidelberg, 2013, pp. 382–383.

[5] E. M. Clarke, D. Peled, and O. Grumberg, *Model checking*, 1999.

[6] K. Havelund and G. Roşu, "Efficient monitoring of safety properties," *International Journal on Software Tools for Technology Transfer*, vol. 6, no. 2, pp. 158–173, Aug 2004.

[7] H. Yang, B. Hoxha, and G. Fainekos, "Querying parametric temporal logic properties on embedded systems," in *ICTSS 2012, LNCS, vol 7641*, B. Nielsen and C. Weise, Eds. Springer Berlin Heidelberg, 2012, pp. 136–151.

[8] T. Nghiem, S. Sankaranarayanan, G. Fainekos, F. Ivancić, A. Gupta, and G. J. Pappas, "Monte-carlo techniques for falsification of temporal properties of non-linear hybrid systems," ser. HSCC '10. New York, NY, USA: ACM, 2010, pp. 211–220.

[9] G. Bombara, C.-I. Vasile, F. Penedo, H. Yasuoka, and C. Belta, "A decision tree approach to data classification using signal temporal logic," in *Proceedings of the 19th International Conference on Hybrid Systems: Computation and Control*, ser. HSCC '16. New York, NY, USA: ACM, 2016, pp. 1–10.

[10] C. Yoo and C. Belta, "Rich time series classification using temporal logic." in *Robotics: Science and Systems*, 2017.

[11] D. Gabbay, "The declarative past and imperative future," in *Temporal Logic in Specification. LNCS, vol 398*. Springer Berlin Heidelberg, 1989, pp. 409–448.

[12] C. Belta, B. Yordanov, and E. A. Gol, *Formal methods for discrete-time dynamical systems*. Springer, 2017, vol. 89.

[13] V. Raman, A. Donzé, D. Sadigh, R. M. Murray, and S. A. Seshia, "Reactive synthesis from signal temporal logic specifications," in *Proceedings of the 18th international conference on hybrid systems: Computation and control*. ACM, 2015, pp. 239–248.

[14] M. Mitchell, *An Introduction to Genetic Algorithms*. Cambridge, MA, USA: MIT Press, 1998.

[15] D. Floreano and C. Mattiussi, *Bio-inspired artificial intelligence: theories, methods, and technologies*. MIT press, 2008.

[16] S. Lessmann, B. Baesens, C. Mues, and S. Pietsch, "Benchmarking classification models for software defect prediction: A proposed framework and novel findings," *IEEE Trans. Softw. Eng.*, vol. 34, no. 4, pp. 485–496, Jul. 2008. [Online]. Available: http://dx.doi.org/10.1109/TSE.2008.35

[17] D. M. Powers, "Evaluation: from precision, recall and f-measure to roc, informedness, markedness and correlation," 2011.

[18] E. Bartocci, L. Bortolussi, and G. Sanguinetti, "Data-driven statistical learning of temporal logic properties," in *FORMATS 2014, LNCS, vol 8711*. Springer International Publishing, 2014, pp. 23–37.

[19] E. Asarin, A. Donzé, O. Maler, and D. Nickovic, "Parametric identification of temporal properties," in *International Conference on Runtime Verification*. Springer, 2011, pp. 147–160.

[20] B. Hoxha, A. Dokhanchi, and G. Fainekos, "Mining parametric temporal logic properties in model-based design for cyber-physical systems," *International*

Journal on Software Tools for Technology Transfer, vol. 20, no. 1, pp. 79–93, Feb 2018. [Online]. Available: https://doi.org/10.1007/s10009-017-0447-4

[21] J. Kapinski, X. Jin, J. Deshmukh, A. Donze, T. Yamaguchi, H. Ito, T. Kaga, S. Kobuna, and S. Seshia, "St-lib: a library for specifying and classifying model behaviors," SAE Technical Paper, Tech. Rep., 2016.

[22] Simulink, *version 8.0 (R2012b)*. Natick, Massachusetts: The MathWorks Inc., 2012.

[23] X. Jin, A. Donze, J. V. Deshmukh, and S. A. Seshia, "Mining requirements from closed-loop control models," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 34, no. 11, pp. 1704–1717, Nov 2015.

[24] Y. Annpureddy, C. Liu, G. Fainekos, and S. Sankaranarayanan, "S-taliro: A tool for temporal logic falsification for hybrid systems," in *Tools and Algorithms for the Construction and Analysis of Systems*, P. A. Abdulla and K. R. M. Leino, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 254–257.

[25] S. Sankaranarayanan and G. Fainekos, "Falsification of temporal properties of hybrid systems using the cross-entropy method," in *Proceedings of the 15th ACM international conference on Hybrid Systems: Computation and Control*. ACM, 2012, pp. 125–134.

[26] Y. S. R. Annapureddy and G. E. Fainekos, "Ant colonies for temporal logic falsification of hybrid systems," in *IECON 2010-36th Annual Conference on IEEE Industrial Electronics Society*. IEEE, 2010, pp. 91–96.

[27] T. Nghiem, S. Sankaranarayanan, G. Fainekos, F. Ivancić, A. Gupta, and G. J. Pappas, "Monte-carlo techniques for falsification of temporal properties of non-linear hybrid systems," in *Proceedings of the 13th ACM international conference on Hybrid systems: computation and control*. ACM, 2010, pp. 211–220.

[28] R. H. Myers, D. C. Montgomery, and C. M. Anderson-Cook, *Response surface methodology: process and product optimization using designed experiments*. John Wiley & Sons, 2016.

[29] E. Bartocci, L. Bortolussi, L. Nenzi, and G. Sanguinetti, "System design of stochastic models using robustness of temporal properties," *Theoretical Computer Science*, vol. 587, pp. 3–25, 2015.

[30] A. Legay, B. Delahaye, and S. Bensalem, "Statistical model checking: An overview," in *Runtime Verification*, H. Barringer, Y. Falcone, B. Finkbeiner, K. Havelund, I. Lee, G. Pace, G. Roşu, O. Sokolsky, and N. Tillmann, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 122–135.

[31] N. Srinivas, A. Krause, S. M. Kakade, and M. Seeger, "Gaussian process optimization in the bandit setting: No regret and experimental design," *arXiv preprint arXiv:0912.3995*, 2009.

[32] J. V. Deshmukh, X. Jin, S. Seshia *et al.*, "Learning auditable features from signals using unsupervised temporal projection." HSCC, 2017.

[33] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg *et al.*, "Scikit-learn: Machine learning in python," *Journal of machine learning research*, vol. 12, no. Oct, pp. 2825–2830, 2011.

[34] D. Reynolds, "Gaussian mixture models," *Encyclopedia of biometrics*, pp. 827–832, 2015.

[35] J. A. Suykens and J. Vandewalle, "Least squares support vector machine classifiers," *Neural processing letters*, vol. 9, no. 3, pp. 293–300, 1999.

[36] S. Jha, A. Tiwari, S. A. Seshia, T. Sahai, and N. Shankar, "Telex: learning signal temporal logic from positive examples using tightness metric," *Formal Methods in System Design*, pp. 1–24, 2019.

[37] F. Denis, "Learning regular languages from simple positive examples," *Machine Learning*, vol. 44, no. 1-2, pp. 37–66, 2001.

[38] B. Zhang and W. Zuo, "Learning from positive and unlabeled examples: A survey," in *2008 International Symposiums on Information Processing*. IEEE, 2008, pp. 650–654.

[39] S. Muggleton, "Learning from positive data," in *International Conference on Inductive Logic Programming*. Springer, 1996, pp. 358–376.

[40] Z. Kong, A. Jones, A. Medina Ayala, E. Aydin Gol, and C. Belta, "Temporal logic inference for classification and prediction from data," in *Proceedings of the 17th international conference on Hybrid systems: computation and control*. ACM, 2014, pp. 273–282.

[41] A. Jones, Z. Kong, and C. Belta, "Anomaly detection in cyber-physical systems: A formal methods approach," in *53rd IEEE Conference on Decision and Control*. IEEE, 2014, pp. 848–853.

[42] P. J. Van Laarhoven and E. H. Aarts, "Simulated annealing," in *Simulated annealing: Theory and applications*. Springer, 1987, pp. 7–15.

[43] H. Laurent and R. L. Rivest, "Constructing optimal binary decision trees is np-complete," *Information processing letters*, vol. 5, no. 1, pp. 15–17, 1976.

[44] E. Aydin Gol, "Efficient online monitoring and formula synthesis with past stl," in *5th IEEE International Conference on Control, Decision and Information Technologies (Codit)*, 2018.

[45] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, "The weka data mining software: an update," *ACM SIGKDD explorations newsletter*, vol. 11, no. 1, pp. 10–18, 2009.

[46] S. L. Salzberg, "C4. 5: Programs for machine learning by J. Ross Quinlan. Morgan Kaufmann Publishers, INC., 1993," *Machine Learning*, vol. 16, no. 3, pp. 235–240, 1994.

[47] A. Ketenci and E. A. Gol, "Synthesis of monitoring rules via data mining," in *2009 American control conference*. IEEE, 2009, pp. 7–pp.

[48] S. K. Aydin and E. A. Gol, "On the use of genetic algorithms for synthesis of signal temporal logic formulas," in *2018 26th Signal Processing and Communications Applications Conference (SIU)*, May 2018, pp. 1–4.

[49] S. F. Elena and R. E. Lenski, "Microbial genetics: evolution experiments with

microorganisms: the dynamics and genetic bases of adaptation," *Nature Reviews Genetics*, vol. 4, no. 6, p. 457, 2003.

[50] (2018) Open research maritime vessel tracking dataset. [Online]. Available: https://www.marinetraffic.com/research/open-research-maritime-vessel-tracking-dataset/

[51] L. Nenzi, S. Silvetti, E. Bartocci, and L. Bortolussi, "A robust genetic algorithm for learning temporal specifications from data," in *International Conference on Quantitative Evaluation of Systems*. Springer, 2018, pp. 323–338.

[52] S. K. Aydin and E. A. Gol, "Optimizing parameters of signal temporal logic formulas with local search," in *2019 27th Signal Processing and Communications Applications Conference (SIU)*, April 2019, pp. 1–4.