İSTANBUL TEKNİK ÜNİVERSİTESİ ★ FEN BİLİMLERİ ENSTİTÜSÜ

ÜST-SEZGİSELLER İÇİN ALANA ÖZGÜ DİL TASARIMI

YÜKSEK LİSANS TEZİ

Hilal Kevser CORA

Bilgisayar Mühendisliği Anabilim Dalı

Bilgisayar Mühendisliği Programı

HAZİRAN 2013

İSTANBUL TEKNİK ÜNİVERSİTESİ ★ FEN BİLİMLERİ ENSTİTÜSÜ

ÜST-SEZGİSELLER İÇİN ALANA ÖZGÜ DİL TASARIMI

YÜKSEK LİSANS TEZİ

Hilal Kevser CORA
(504101508)

Bilgisayar Mühendisliği Anabilim Dalı

Bilgisayar Mühendisliği Programı

Tez Danışmanı: Doç. Dr. Şima UYAR

HAZİRAN 2013

İTÜ, Fen Bilimleri Enstitüsü'nün 504101508 numaralı Yüksek Lisans Öğrencisi **Hilal Kevser CORA**, ilgili yönetmeliklerin belirlediği gerekli tüm şartları yerine getirdikten sonra hazırladığı **"ÜST-SEZGİSELLER İÇİN ALANA ÖZGÜ DİL TASARIMI"** başlıklı tezini aşağıdaki imzaları olan jüri önünde başarı ile sunmuştur.

Tez Danışmanı :	**Doç. Dr. Şima UYAR** İstanbul Teknik Üniversitesi	.....
Jüri Üyeleri :	**Prof. Dr. Hasan DAĞ** Kadir Has Üniversitesi	.....
	Yrd. Doç. Dr. Feza BUZLUCA İstanbul Teknik Üniversitesi	.....

Teslim Tarihi : **3 Mayıs 2013**
Savunma Tarihi : **7 Haziran 2013**

ÖNSÖZ

Çalışmalarım süresince yardımları ve katkıları ile her zaman yol gösteren tez danışmanlarım Doç. Dr. Şima Uyar ve Turgut Uyar'a, hayatımın her döneminde olduğu gibi yüksek lisans eğitimim boyunca da bana destek olan aileme ve çalışmalarım boyunca desteğini benden esirgemeyen sevgili eşime teşekkürü bir borç bilirim.

Haziran 2013

Hilal Kevser CORA
(Bilgisayar Mühendisi)

İÇİNDEKİLER

Sayfa

ÖNSÖZ ..... **v**
İÇİNDEKİLER ..... **vii**
KISALTMALAR ..... **ix**
ÇİZELGE LİSTESİ ..... **xi**
ŞEKİL LİSTESİ ..... **xiii**
ÖZET ..... **xv**
SUMMARY ..... **xvii**
1. GİRİŞ **1**
2. ÜST SEZGİSELLER **3**
2.1 Giriş 3
2.2 Üst Sezgisel Yöntemlerinin Sınıflandırılması 4
2.2.1 Sezgisel seçen sezgisel metodolojileri 4
2.2.2 Sezgisel üreten sezgisel metodolojileri 5
2.3 Üst Sezgisel Algoritması 5
2.3.1 Sezgisel yönteminin seçilmesi 5
2.3.2 Seçilen yöntemin uygulanması 6
2.3.3 Uygulanan yöntemin değerlendirilmesi 6
2.3.4 Sonlanma kriteri 7
2.4 HyFlex Anaçatısı 7
3. ALANA ÖZGÜ DİLLER **11**
3.1 Giriş 11
3.2 Alana Özgü Dil Geliştirmek 14
3.2.1 Karar 14
3.2.2 Analiz 14
3.2.3 Tasarım 14
3.2.4 Gerçekleme 15
3.3 Alana Özgü Dil Geliştirme Araçları 15
4. ÜST SEZGİSELLER İÇİN ALANA ÖZGÜ DİL TASARLANMASI **23**
4.1 Analiz 23
4.1.1 Sezgisel seçim yönteminin seçilmesi 23
4.1.2 Kabul yönteminin seçilmesi 25
4.1.3 Başlatma 25
4.1.4 Sezgisellerin seçilmesi 26
4.1.5 Sezgisellerin uygulanması 26
4.1.6 Çözümlerin kabul edilmesi 26
4.1.7 Yapı tanımlanması 26
4.1.8 Tip Sistemi 28

5. ÜST SEZGİSELLER İÇİN ALANA ÖZGÜ DİL GERÇEKLENMESİ..... 31
5.1 Söz Dizimi 31
5.2 Kod üretme 34
5.3 Gerçekleme 35
5.3.1 Döngüler 35
5.3.2 Koşullu dallanmalar 37
5.3.3 Sezgisel seçim yönteminin tanımlanması 39
5.3.4 Kabul yöntemi seçimi 45
5.3.5 Başlatma 47
5.3.6 Sezgisellerin seçilmesi 49
5.3.7 Sezgisellerin uygulanması 49
5.3.8 Çözümlerin kabul edilmesi 54
5.3.9 Yapı tanımlanması 56
6. KULLANIM VE ÖRNEKLER 65
6.1 Kullanım 65
6.2 Örnekler 65
7. SONUÇ VE ÖNERİLER 69
KAYNAKLAR 71
EKLER 73
EK A.1 75
EK A.2 77
EK A.3 81
EK A.4 84
EK A.5 87
EK B 91
ÖZGEÇMİŞ 93

KISALTMALAR

CHeSC : Cross-domain Heuristic Search Challenge
CSS : Cascading Style Sheets
DSL : Domain Specific Language
EMF : Eclipse Modeling Framework
GPL : General Purpose Language
HyFlex : Hyper-heuristics Flexible Framework
LINQ : Language-Integrated Query
SDF : Syntax Definition Formalism
SQL : Structured Query Language

ÇİZELGE LİSTESİ

Sayfa

Çizelge 5.1: SDF'de yer alan bazı söz dizimi kalıpları. 34

ŞEKİL LİSTESİ

Sayfa

Şekil 2.1 : Bir üst sezgisel alt seviye sezgiseller ile haberleşirken, alt seviye sezgiseller problemle doğrudan haberleşirler..... 3
Şekil 2.2 : Kutulama problemine üst sezgisel yaklaşımı. 4
Şekil 2.3 : HyFlex problemler ve üst sezgiseller arasında yer alan bir köprüdür. 8
Şekil 3.1 : Spoofax ile kod renklendirme..... 19
Şekil 3.2 : Spoofax ile kod tamamlama. 20
Şekil 3.3 : Spoofax ile hata tespiti. 20
Şekil 3.4 : Spoofax ile kod katlama. 21
Şekil 5.1 : Tasarlanan dilin gramer kuralları. 32
Şekil 5.2 : Tasarlanan dilin gramer kuralları. 33
Şekil 6.1 : Basit rastgele seçim ve iyileşmeyen çözümlerin olasılıksal kabulünün yapıldığı, bir çözüm adayının olduğu örnek program. 66
Şekil 6.2 : Üç sezgisel seçim yönteminin olduğu, üretilen tüm çözümlerin kabul edildiği, 10 çözüm adayı olan örnek program..... 67
Şekil 6.3 : Sezgisel türünün kontrol edildiği örnek program. 67
Şekil 6.4 : Sezgisel seçim yönteminin tanımlandığı örnek program. 68
Şekil 6.5 : Kabul yönteminin tanımlandığı örnek program. 68
Şekil B.1 : Tasarlanan dilde algoritma yazmak ve çalışabilir kod üretmek için izlenecek yol haritası..... 91

ÜST-SEZGİSELLER İÇİN ALANA ÖZGÜ DİL TASARIMI

ÖZET

Üst sezgiseller alt seviye sezgisellerle işlem yapan metodolojilerdir. Alt seviye sezgiselleri seçmek veya üretmek için kullanılırlar. Üst sezgisellerle ilgili çalışma yapan kişilerin tasarladıkları algoritmaları gerçeklemeleri için yüksek düzey programlama dillerini bilmeleri gerekmektedir. Programlama bilgisi olmayan araştırmacılar bu açıdan zorluk çekmekte, çözmek istedikleri problemleri ve çözümlerini iyi ifade edememektedir.

Alana özgü diller belirli bir alan için özel olarak tasarlanmış dillerdir. Kullanıldıkları alanda mevcut olan kavramlar dilin bir parçasını oluşturmaktadır. Bu açıdan genel amaçlı dillerle kıyaslandıklarında alanla ilgili kavramlar daha kolay ve anlaşılır bir biçimde ifade edilebilmektedir. Alana özgü dilleri kullanarak program yazmak hiç programlama bilmeyen kişiler için daha kolay olmaktadır.

Bu çalışmada üst sezgiseller için alana özgü bir dil geliştirilmiştir. Alana özgü dil geliştirmek için Eclipse tabanlı Spoofax dil geliştirme aracı kullanılmıştır. Geliştirilen dilin söz dizimi ve kod üretme kuralları Spoofax üzerinde tanımlanmıştır. Spoofax geliştirilen dilde yazılmış ifadeleri tanımlanan kod üretme kurallarından yola çıkarak Java koduna dönüştürmektedir. Üst sezgiseller için alana özgü bir geliştirmeden önce bu alanda kapsamlı bir literatür araştırması yapıldı. Literatür araştırmasıyla birlikte alana özgü terimler, kavramlar ve yöntemler tespit edildi. Üst sezgisellerle ilgili yazılmış program kodları incelendi ve üst sezgisel algoritmaları Java dili kullanarak gerçeklendi. Böylece tasarlanan dilde ihtiyaç duyulan yapılar belirlendi. Bu tecrübelerden yola çıkarak dilin söz dizimi ve kod üretme kuralları tanımlandı. Geliştirilen dil ile belirli yöntemler kullanarak sezgisel seçmek, seçilen sezgiseli çözüm adaylarına uygulamak ve üretilen yeni çözümleri değerlendirmek mümkündür. Geliştirilen dil sayesinde, üst sezgisel alanında çalışan araştırmacılar genel amaçlı bir dil öğrenmek zorunda kalmadan algoritmalarını gerçekleyebilirler.

A DOMAIN SPECIFIC LANGUAGE FOR HYPER-HEURISTICS

SUMMARY

In this study, a domain specific language for developing hyper-heuristic programs is proposed.

A domain specific language (DSL) is a programming language which aims to make it easier to develop programs in a certain domain. A well designed DSL provides a more natural notation and more suitable data structures to express solutions to problems of the targeted domain as opposed to a general purpose language (GPL). Although using a GPL in combination with a library or package for the domain is a very common approach, it still assumes a considerable amount of programming knowledge, therefore making it harder to be productive for domain experts who might have limited or no programming skills.

The development of a DSL is a difficult task because it requires expertise both in the targeted domain and in programming language design and implementation. The phases of development are outlined as decision, analysis, design and implementation. First, it has to be decided if a new DSL is really needed instead of using existing languages and tools. The advantages of domain related notation and data structures have to be evaluated. In the analysis phase, domain knowledge is gathered from documents, domain experts and existing program codes developed for the domain. Next, the DSL is designed in light of the design patterns which emerged in the analysis phase. And finally, the DSL is implemented by developing a language processor such as an interpreter or a compiler or by embedding it in a GPL. There are many studies about DSL design and implementation.

Since developing a new DSL is a complex process, it is recommended to use language development tools or frameworks. Therefore, beside the research on developing DSLs, there are also many studies on developing frameworks to generate and implement DSLs.

Domain specific languages can be designed as internal or external. Internal DSLs are integrated into an existing general purpose language and use features of the original language. However, internal DSLs might be complex for non-programmers because they should know syntax and constraints of the host language. These people might feel more confident while using external DSLs. External DSLs also have IDE support like syntax highlighting and code completion which improves productivity.

Another decision is whether to design a textual or a graphical DSL. Graphical notation can seem simpler but it cannot cover all domain concepts. Domain concepts can be stated with textual DSLs more clearly.

Hyper-heuristics are high-level methodologies which were proposed as an alternative to heuristics tailored to a specific problem domain. There are two

types of hyper-heuristics in literature: heuristics to choose heuristics (selection hyper-heuristics) and heuristics to generate heuristics. In this study, we work with only the first type of hyper-heuristics. Therefore, in this work, the term hyper-heuristics refers to selection hyper-heuristics. A hyper-heuristic selects and applies a low-level heuristic without using any problem specific information. Low-level heuristics are heuristics tailored to a specific problem domain. Therefore, low-level heuristics operate on the problem search space, whereas a hyper-heuristic operates on the low-level heuristics search space.

Over the years, research on hyper-heuristics has shown an increase. Recently, HyFlex (Hyper-heuristics Flexible framework), a hyper-heuristics framework which provides easy implementation of hyper-heuristics and allows comparisons between them, has been implemented. HyFlex can be used with different hyper-heuristics and different problem domains. In the CHeSC (Cross-domain Heuristic Search Challenge) competitions, the HyFlex framework is used as the underlying system for hyper-heuristic development. CHeSC competitions aim to bring together researchers and practitioners from operational research and computer science fields to develop more general approaches to problem solving across many domains. Since researchers and practitioners come from different fields, programming skills may be a drawback for some. To write hyper-heuristic code working on HyFlex, one needs to have good Java programming skills. To alleviate this requirement and allow the researchers to focus on the hyper-heuristic development phase rather than the details of Java programming, we decided to develop a DSL for hyper-heuristic research. The proposed DSL aims to generate code which can be executed on HyFlex.

Hyper-heuristics are composed of two components: heuristic selection schemes and acceptance criteria. Heuristic selection schemes deal with deciding which low-level heuristic to choose and apply to the current candidate solution. Acceptance criteria are used to determine whether to accept or reject the new candidate solution created as a result of applying the selected low-level heuristic.

In this study, the Spoofax language development framework, which is integrated into Eclipse, is used. When using this framework, two aspects of the DSL have to be expressed: the grammar and the code generation rules. From these specifications, Spoofax generates the necessary components like the parser and the code generator. It also prepares an environment which supports assistive features like syntax highlighting for the target DSL.

The syntax of the language is specified using the SDF notation as required by the Spoofax platform. Some of the reserved keywords of our DSL are `best`, `first`, `random`, `select`, `solution`, `heuristic`, `apply` and `replace`, which map to concepts in the hyper-heuristics domain. Other than these, some standard programming language constructs like `if-else` and `loop` are also supported. The `loop` iteration construct can take two forms. Beside the usual counter controlled repetition which allows repeating for a certain number of times, loops can also be limited by the time they are allowed to spend; so the total time available to the program can be divided between loops. This feature is implemented as a convenience for programs developed for the CHeSC competition.

While some of these types of general purpose constructs are necessary when expressing hyper-heuristic algorithms, adding too many such constructs might damage the aim of being a DSL and the language might start resembling a GPL. Some higher level

features of our DSL are primarily meant for defining new hyper-heuristic operators instead of using standard operators which are already included in the platform.

Typically, a program in our DSL starts by setting the environment for later stages, such as configuring a selection heuristic method and an acceptance method. Then a set of initial solutions will be generated and the algorithm will proceed to a loop where the chosen methods will be applied to candidate solutions.

The code generation rules are written using the Stratego program transformation language which is integrated into Spoofax. The code generation process produces Java code by applying rewrite rules to terms. Code generation rules are matched with the labels of the production rules. In the code generation phase, our system generates Java code which can be directly executed on the HyFlex platform. The Spoofax framework uses string interpolation for code generation. In this approach, the code to generate is expressed as a template containing references to variables and these variables will be replaced by their values.

After editing the code in the editor component of the working environment, code generation is invoked from the menu. An Eclipse Java project is generated automatically when code generation is invoked.

Proposed DSL is a domain specific language for developing hyper-heuristic programs, which can be executed on HyFlex. Through examples, how hyper-heuristic programs can be developed using this DSL is illustrated. The examples show that writing a hyper-heuristic program in the proposed DSL is easier than writing the same in Java. Currently, the features provided in our DSL are designed to work with hyper-heuristic development on HyFlex, however, features can be added to make it a more general tool for hyper-heuristic research. The advantage of working with HyFlex is that HyFlex provides the problem domain implementations. Therefore, researchers can focus on hyper-heuristic development rather than implementing the problems to solve. It also allows researchers to benchmark their approaches on standard implementations of these problem domains.

1. GİRİŞ

Sezgiseller bir problemin arama uzayında çalışan alt düzey yöntemlerdir. Üst sezgiseller ise bu alt düzey yöntemler ile çalışan sezgisellerdir. Üst sezgiseller alt düzey sezgiselleri seçmek veya oluşturmak için kullanılırlar ve üzerinde çalışılan problemle ilgili hiçbir bilgiye sahip değildirler. Üst sezgisellerin amacı aynı yöntemi farklı problemlere uygulayarak çeşitli sonuçlar elde etmek ve bu sonuçları karşılaştırmaktır.

Günümüzde üst sezgisel algoritmalarını gerçeklemek amacıyla sunulan çeşitli anaçatılar bulunmaktadır. Ancak bu anaçatıları kullanabilmek için genellikle yüksek düzey bir programlama dilini bilmek gerekmektedir. Üst düzey bir programlama dili bilmeyen araştırmacılar tasarladıkları yöntemleri bu dillerle ifade edemedikleri için yaptıkları çalışmalarda zorluk çekmektedirler. Üst sezgiseller alanında kullanılmak üzere tasarlanan bir dil araştırmacıların sorunlarını çözecek, problemlere ilişkin geliştirdikleri çözümleri daha rahat ifade edebilmelerini sağlayacak ve üretkenliği artıracaktır.

Bu çalışmada üst sezgisel alanında çalışan araştırmacılara kolaylık sağlamak amacıyla üst sezgiseller için kullanılacak alana özgü bir dil sunulmuştur. Alana özgü diller belirli bir çalışma alanındaki modelleri, terimleri ve yöntemleri genel amaçlı dillere kıyasla daha anlaşılır şekilde ifade edebilmektedir.

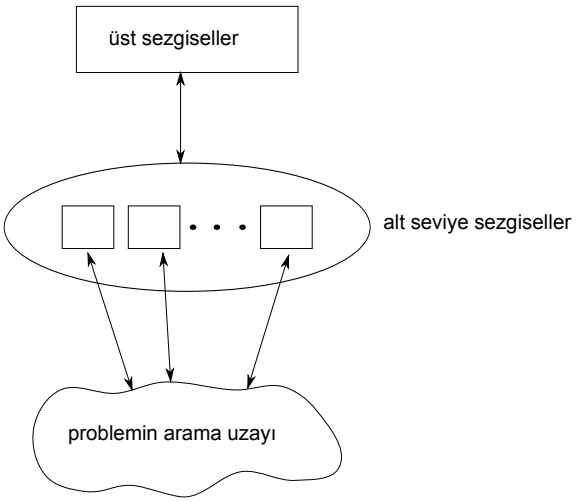
Alana özgü dil geliştirmek için takip edilmesi gereken adımlar bulunmaktadır. Bu adımlar karar, analiz, tasarım ve gerçekleme adımlarıdır. Bu çalışmada da bu adımlar izlenmiş ve detaylı olarak açıklanmıştır. Sunduğumuz dil üst sezgiseller alanında kullanılan kavramları üst düzey bir dil bilmeyi gerektirmeden gerçeklemeye imkan sağlamaktadır. Geliştirilen dilde üst sezgiseller alanında yer alan kavramlara karşılık gelen yapılar tanımlanacak, özetle sezgiselleri belirli yöntemlerle seçmek, uygulamak ve üretilen çözümleri birçok şekilde değerlendirmek mümkün olacaktır.

Tezin 2. bölümünde üst sezgiseller anlatılmış ve bu alanda literatür araştırması yapılmıştır. 3. bölümde alana özgü diller hakkında bilgi verilmiş, alana özgü dil geliştirme araçları anlatılmıştır. 4. bölümde üst sezgiseller için geliştirilmiş olan dilin tasarımına, 5. bölümde ise bu dilin gerçeklenmesine yer verilmiştir. Tasarlanan dilin kullanımı ve bu dil kullanılarak yazılmış örnek programlar 6. bölümde yer almaktadır. 7. bölümde ise sonuçlar ve önerilere yer verilmiştir.

2. ÜST SEZGİSELLER

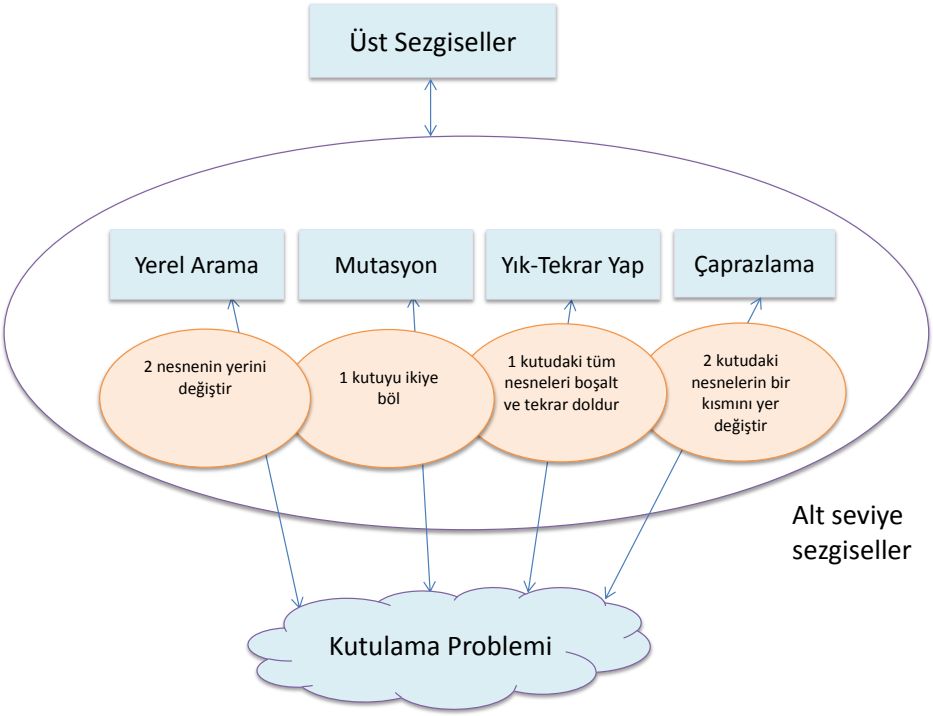
2.1 Giriş

Üst sezgiseller belirli bir problem alanında şekillenmiş sezgisellere bir alternatif olarak sunulmuş ileri seviye yöntemlerdir [1]. Alt seviye sezgiseller belirli bir probleme özgü olarak tanımlanmış sezgisellerdir. Bu yüzden alt seviye sezgiseller problemin arama uzayında çalışırken bir üst sezgisel alt seviye sezgisellerin arama uzayında çalışmaktadır (Şekil 2.1).



Şekil 2.1: Bir üst sezgisel alt seviye sezgiseller ile haberleşirken, alt seviye sezgiseller problemle doğrudan haberleşirler.

Şekil 2.2'de kutulama problemi için kullanılan alt seviye sezgiseller ve bu sezgiselleri seçen üst sezgiseller görülmektedir. Kutulama probleminde amaç N tane nesneyi V hacmindeki minimum sayıda kutuya sığdırmaktır. Bir problemde kullanılacak olan alt seviye sezgiseller türleri yerel arama, mutasyon, yık-yeniden yap ve çaprazlama olmak üzere dört genel başlıkta ifade edilebilir. Kutulama problemi için kullanılabilecek alt seviye sezgisellerden bazıları iki nesnenin yerini değiştirme, bir kutunun eşit sayıda nesne içerecek şekilde ikiye bölünmesi, bir kutudaki nesnelerin tamamının boşaltılıp yeni nesnelerin doldurulması ve iki kutudaki nesnelerin bir kısmının yer değiştirilmesi olarak sayılabilir. Üst sezgiseller ise burada sayılan alt seviye sezgiseller arasından seçim yaparak probleme uygular ve problemi çözmeye çalışır.



Şekil 2.2: Kutulama problemine üst sezgisel yaklaşımı.

2.2 Üst Sezgisel Yöntemlerinin Sınıflandırılması

Literatürde iki tür üst-sezgisel metodolojisi vardır: sezgisel seçmek için kullanılan sezgisel metodolojileri ve sezgisel üretmek için kullanılan sezgisel metodolojileri [2]. Bu çalışmada sadece sezgisel seçmek için kullanılan sezgisel çeşidi kullanılmıştır. Bu sebepten ötürü, bu çalışmada üst sezgisel terimi sezgisel seçen sezgiselleri ifade etmek için kullanılacaktır.

2.2.1 Sezgisel seçen sezgisel metodolojileri

Sezgisel seçme işlemi var olan sezgisellerin bir kombinasyonunu oluşturmaktır [2]. Sezgisel seçen sezgiseller tam bir çözüm oluşturmak amacıyla artımlı bir şekilde sezgisel seçer ve seçilen sezgisel yöntemi çözüme uygularlar [1]. Bir üst sezgisel problemle ilgili hiçbir bilgi sahibi olmadan arama uzayından bir sezgisel seçer ve seçtiği sezgiseli çözümlere uygular [3].

2.2.2 Sezgisel üreten sezgisel metodolojileri

Sezgisel üretme işlemi var olan sezgisellerin temel bileşenlerini kullanarak yeni sezgiseller üretmektir [2]. Literatürde bulunan sezgisel üretme metodolojilerinin çoğu, bilgisayar programlarını otomatik üretmeye imkan sağlayan genetik programlama yöntemini kullanmaktadır.

2.3 Üst Sezgisel Algoritması

Bir üst-sezgisel algoritması aşağıdaki adımlardan oluşur:

Adım 0

Uygulanacak alt seviye sezgisel, bir sezgisel seçme yöntemi kullanılarak seçilir.

Adım 1

Seçilen sezgisel çözüm adayına uygulanır, yeni bir çözüm adayı elde edilir ve sonuçlar kaydedilir. (çalışma süresi, çözümün iyileşmesi)

Adım 2

Yeni çözüm adayının kabul edilip edilmeyeceği kontrol edilir. Kabul edilirse yeni çözüm adayı ile çalışmaya devam edilir.

Adım 3

Sonlanma kriterine ulaşılıp ulaşılmadığı kontrol edilir. (çözümün yeterince iyileşip iyileşmediği) Sonlanma kriteri sağlanmadıysa 1. adıma geri dönülür.

2.3.1 Sezgisel yönteminin seçilmesi

Sezgisel yöntem seçilirken problemle ilgili hiçbir bilgi kullanılmadan bir sezgisel seçilir. Literatürde birçok sezgisel yöntemi bulunmaktadır, bunlardan bazıları [4]'te anlatılmaktadır.

- Basit seçim yöntemi (Simple Random) alt seviye sezgisellerden birini rastgele seçer ve çözüm adayına uygular.
- Rastgele permütasyon seçim yöntemi (Random Permutation) tüm alt seviye sezgisellerin rastgele bir permütasyonunu oluşturur ve bu sezgiselleri permütasyondaki sıraya göre çözüm adayına uygular. Bu yöntemde her alt seviye sezgisel yalnızca bir kez uygulanır.

- Açgözlü seçim yöntemi (Greedy) bütün alt seviye sezgiselleri çözüm adayına tek tek uygular ve en fazla iyileşmeyi sağlayan sezgiseli seçer.
- Rastgele azalan seçim yöntemi (Random Descent) basit seçim yöntemiyle aynıdır ancak seçilen sezgisel çözüm adayına hiçbir iyileşme olmayana kadar uygulanır.
- Rastgele permütasyonla azalan seçim yöntemi (Random Permutation Descent) rastgele permütasyon yönteminin aynısıdır ancak seçilen sezgisel çözüm adayına hiçbir iyileşme olmayana kadar uygulanır.
- Pekiştirmeli öğrenme (Reinforcement Learning) veya seçim fonksiyonu (Choice Function) yöntemlerinin ikisi de alt seviye sezgisellerin performansına dayanan bir puanlama mekanizması kullanırlar ve her iterasyonda en yüksek puana sahip olan sezgiseli seçerler.

Bunlar dışında gelişmiş birçok sezgisel seçim yöntemi de literatürde yer almaktadır. Tepe tırmanma yöntemi, genetik algoritmalar ve bulanık sistemler bu karmaşık yöntemler arasında sayılabilir [1].

2.3.2 Seçilen yöntemin uygulanması

Sezgisel yöntemi seçildikten sonra seçilen yöntem çözüm adayına uygulanır. Bunun sonucunda yeni bir çözüm adayı elde edilmektedir.

2.3.3 Uygulanan yöntemin değerlendirilmesi

Seçilen sezgisel yöntemi çözüm adayına uygulandıktan sonra yeni bir çözüm adayı elde edilir. Bu çözüm adayı eskisiyle karşılaştırıldığında iyileşmiş veya kötüleşmiş olabilir. Bunu değerlendirmek için kullanılan çeşitli yöntemler bulunmaktadır. Seçilen sezgisel yöntemi çözüm adayına uygulandıktan sonra yeni bir çözüm adayı elde edilir. Bu çözüm adayı eskisiyle karşılaştırıldığında iyileşmiş veya kötüleşmiş olabilir. Bunu değerlendirmek için kullanılan çeşitli yöntemler bulunmaktadır.

- Yeni çözüm adayları doğrudan kabul edilir.
- Sadece iyileşen çözüm adayları kabul edilir.
- İyileşen veya aynı kalan çözüm adayları kabul edilir.

- İyileşen çözüm adayları kabul edilir, iyileşmeyenler ise belirli bir olasılıkla kabul edilir.

Üretilen çözüm bu yöntemler kullanılarak değerlendirilir. Kabul kriterleri sağlanıyorsa yeni çözüm adayı kabul edilir, aksi durumda yeni çözüm kabul edilmez ve eski çözüm kullanılır. Bu yöntemler dışında kullanılan daha gelişmiş ve karmaşık değerlendirme yöntemleri de literatürde bulunmaktadır. Monte Carlo, Benzetimli Tavlama (Simulated Annealing) ve Büyük Tufan (Great Deluge) yöntemleri bu karmaşık yöntemler arasında sayılabilir [3].

2.3.4 Sonlanma kriteri

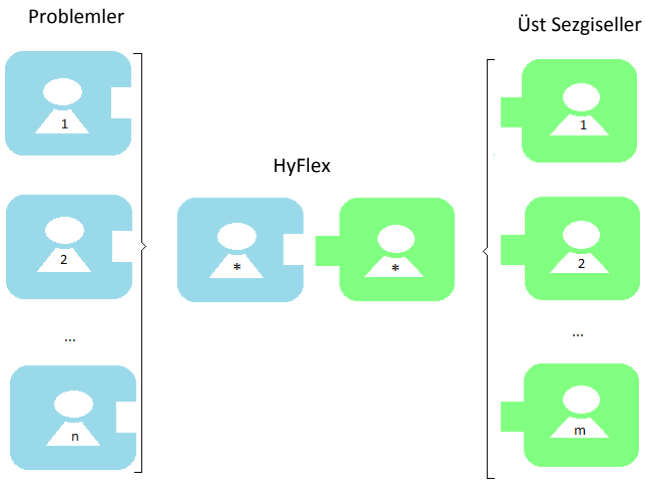
Üst sezgisel algoritmasında kullanılan iterasyonun bir sonlanma koşulu olmak zorundadır. Bu koşul daha önceden belirlenmiş bir sürenin dolması veya çözümün yeterince iyileşmiş olması olabilir.

2.4 HyFlex Anaçatısı

HyFlex (Hyper-heuristics Flexible framework), bir üst-sezgisel geliştirme anaçatısıdır [5]. HyFlex üst sezgisellerin kolaylıkla geliştirilmesine ve geliştirilen üst sezgiseller arasında karşılaştırma yapmaya imkan sağlayan bir anaçatıdır. HyFlex problemlerle üst sezgiseller arasında yer alan bir yazılım arayüzüdür [6]. HyFlex platformunda geliştirilen bir algoritma çeşitli problemlere kolaylıkla uygulanabilir.

HyFlex'te minimizasyon problemi olarak geliştirilmiş 6 problem yer almaktadır. Bunlar tek boyutlu kutulama, permütasyon akış tipi çizelgeleme (Permutation Flow Shop), personel çizelgeleme, mantıksal sağlanabilirlik (Boolean Satisfiability), gezgin satıcı ve araç rotalama problemleridir. Yeni problemler de HyFlex'e kolaylıkla eklenebilir. Her problem bir dizi alt düzey sezgisel içerir. Bunlar mutasyon, yık-tekrar yap, yerel arama ve çaprazlama sezgiselleridir. Alt düzey sezgiseller çözümleri değiştirmek için kullanılırlar.

HyFlex anaçatısı bir sezgisel arama algoritmasını genel amaçlı ve probleme özgü olmak üzere iki ana bölüme ayrıştırmaktadır (Şekil 2.3). Probleme özgü bölümde yer alan problemler ve bu problemlere özgü bilgiler HyFlex tarafından sağlanmaktadır. Genel amaçlı bölümde ise herhangi bir problem üzerinde çalışabilecek genel amaçlı üst sezgisel algoritmaları yer almaktadır.



Şekil 2.3: HyFlex problemler ve üst sezgiseller arasında yer alan bir köprüdür.

HyFlex'te yer alan bir problem domeninde aşağıdaki yetenekler bulunmaktadır:

- Üst sezgiseller tarafından yönetilebilen çözümler
- Çözümlerin ilk değerlerinin rastgele atanması
- Çözümleri değiştirecek bir dizi alt seviye sezgisel
- Kolayca yüklenebilen bir dizi problem örneği
- Mutasyon olasılığı ve arama derinliği

Her yıl düzenlenen CHeSC [1] (Cross-domain Heuristic Search Challenge) yarışmalarında, HyFlex anaçatısı üst sezgisel geliştirmek için kullanılmaktadır. CHeSC yarışmalarının amacı yöneylem araştırması ve bilgisayar bilimleri alanında yer alan araştırmacıları bir araya getirerek çeşitli alanlarda daha genel problem çözme yaklaşımları geliştirmektir. Araştırmacıların uzmanlık alanları farklı olduğundan hepsinin programlama bilgisinin aynı olduğu söylenemez. HyFlex üzerinde çalışan

[1] http://www.asap.cs.nott.ac.uk/external/chesc2011

üst sezgisel kodlarını yazabilmek için araştırmacıların iyi düzeyde Java bilgisine sahip olmaları gerekmektedir.

3. ALANA ÖZGÜ DİLLER

3.1 Giriş

Alana özgü diller belirli bir alanda kolayca uygulama geliştirmeye olanak sağlar. İyi tasarlanmış bir dil, genel amaçlı bir dil ile kıyaslandığında hedeflenen alandaki problemlerin çözümlerini daha doğal bir notasyon ve daha uygun veri yapılarıyla ifade etmeye olanak sağlar.

Hoare'e göre programlama dilleri programcıya sanatını icra ederken karşılaştığı birtakım zorluklarda (tasarım, dokümantasyon ve hata ayıklama gibi) yardımcı olan araçlardır [7]. Hoare'in tanımından yola çıkarak benzer tanım alana özgü diller için de söylenebilir. Günümüzde programlama sadece bilgisayar bilimleri veya mühendislik alanında değil, birçok alanda karşılaşılan problemlerin çözümü için ihtiyaç haline gelmiştir. İyi bir dil, bir programın çeşitli aşamalarında yerine getirilecek görevlerin net bir şekilde ifade edilmesinde yardımcı olur. İyi bir dil programcıya kendi kendini dokümante eden kodlar yazmak konusunda yardımcı olur. Kodun okunabilirliği, yazılabilirliğinden daha önemlidir. İyi bir dil hataların ayıklanmasında dili kullanan kişiye yardımcı olur. Dilin notasyonu yazım hatalarını en aza indirecek şekilde olmalı, mümkünse hatalar bir derleyici yardımıyla tespit edilebilmelidir.

Genel amaçlı dilleri bir kütüphane veya bir paket ile birlikte kullanmak genel bir yaklaşım olsa da bu yöntem göz ardı edilemez miktarda programlama tecrübesi gerektirir. Bu yüzden programlama konusunda az tecrübeli olan veya hiç tecrübesi olmayan uzmanlar için bu yöntem zor olmaktadır.

Alana özgü diller günümüzde çok yaygın kullanılmaktadır. SQL, LINQ, CSS alana özgü dillerin en bilinen örnekleri arasında sayılabilir.

SQL ilişkisel bir veri tabanında tutulan verileri yönetmek için tasarlanmış özel bir programlama dilidir. Veri tabanında yer alan verilere erişmek, yeni veriler oluşturmak ve mevcut veriler üzerinde değişiklik yapmak için kullanılmaktadır. Veri tabanı üzerinde çalışan SQL cümleleri rahat okunabildiği için yapılan işlem daha kolay

anlaşılabilmektedir. Aşağıdaki örnek SQL cümlesinde fiyatı 100 TL den fazla olan kitaplar sorgulanmakta ve başlığa göre sıralanmaktadır:

```
SELECT *
 FROM Kitap
 WHERE fiyat > 100.00
 ORDER BY baslik
```

LINQ, SQL dilinde yer alan yetenekleri .NET platformunda çalışan dillere kazandıran ve Microsoft tarafından geliştirilen özel bir dildir. LINQ dizilerde, sınıflarda, XML dokümanlarında ve ilişkisel veri tabanlarında yer alan verileri de işleyebilmektedir. LINQ dilinde yazılmış ifadelerin okunabilirliği kolaydır. Aşağıdaki örnek LINQ ifadesinde fiyatı 100 TL den fazla olan kitaplar sorgulanmakta ve başlığa göre sıralanmaktadır:

```
var pahaliKitaplar =
     Kitap
         .Where(ktp => ktp.fiyat > 100.00)
         .OrderBy(ktp => ktp.baslik)
         .Select(ktp => new {ktp.fiyat, ktp.baslik});
```

CSS Basamaklı Stil Şablonları olarak da bilinen, HTML tabanlı yazılmış web sayfalarını biçimlendirmek için tasarlanmış özel bir dildir. Bir web sayfasında yer alan resimlerin, metinlerin ve diğer nesnelerin biçimleri, renkleri, yazı tipleri ve diğer birçok özelliği CSS kullanılarak ayarlanabilmektedir. Aşağıdaki örnekte bir HTML sayfasının gövdesinin arka plan rengini kırmızı yapan CSS kodu görülmektedir:

```
<style>
body
{
    background-color:red;
}
</style>
```

Literatürde alana özgü diller konusunda yapılmış bir çok çalışma bulunmaktadır. Alana özgü diller ile genel amaçlı dillerin anlaşılabilirliğini kıyaslayan bir çalışma [8]'de yapılmıştır. Bu çalışmada öğrenciler üzerinde üç deney yapılmıştır. İlk deneyde özellik diyagramları domeni kullanılmıştır. Özellik diyagramları, uygulama alanlarının değişken kısımlarını fark etmeye ve bir sistemi oluşturan bileşenleri ifade etmeye yaramaktadır. İkinci deneyde çizge tanımlama domeni kullanılmıştır. Bu domende düğümlerden ve kenarlardan oluşan yönlü ve yönsüz çizgeler tanımlanmaktadır. Üçüncü deneyde ise grafik kullanıcı arabirimi domeni kullanılmıştır. Bu domende kullanıcıların görsel olarak etkileşimde bulunabilecekleri programlar yazılmaktadır. Hem genel amaçlı dillerle hem de alana özgü dillerle bu alanlar üzerinde programlar yazılmıştır. Test edilen dillerin söz diziminin ne kadar öğrenildiği, bu dillerde yazılmış programların ne derece anlaşıldığı test edilmiş ve öğrencilerin verilen bir program üzerinde bir takım değişiklikler yapması istenmiştir. Sonuçlara bakıldığında DSL kullanarak program geliştirmenin, yazılmış programları anlamanın ve öğrenmenin genel amaçlı dillere oranla daha kolay olduğu gözlemlenmiştir.

[9]'da veri toplama sistemi geliştirmek için alana özgü bir dil tasarlanmıştır. Daha önce hiç programlama tecrübesi olmayan uzmanlar bu dili kullanarak kendi veri toplama sistemlerini geliştirebilirler. Bu alanda yapılmış başka bir çalışma ise EasyTime adı verilen dildir [10]. Bu dil spor müsabakalarında zamanı ölçmek için kullanılmaktadır. Esnek bir zaman ölçüm sistemi sağlayan bu sistem ufak değişiklikler yapıldığında farklı türlerdeki spor müsabakalarına uyum sağlayabilmektedir. Böylece zamanı ölçmek için özel firmalarla anlaşma gereği de ortadan kalkmıştır.

Alana özgü dil geliştirmek karmaşık bir süreç olduğu için bir dil geliştirme aracından yararlanmak tavsiye edilmektedir [11]. Alana özgü dil geliştirme ile alakalı yapılan çalışmalara bakıldığında dil tasarlama ve gerçekleme araçları ve ortamları geliştiren bir çok çalışma bulunmaktadır. Örneğin, Delite derleyici aracı alana özgü paralel, gömülü dillerin geliştirilmesini basitleştirmeyi amaçlamaktadır [12]. Bu çalışmada, önerdiğimiz dili tasarlamak ve gerçeklemek için Spoofax dil geliştirme aracı kullanılmıştır.

3.2 Alana Özgü Dil Geliştirmek

Alana özgü dil geliştirmek zor bir iştir çünkü hem hedeflenen alanda hem de programlama dili tasarımı ve geliştirmesinde uzmanlık gerektirir. Dil geliştirmenin aşamaları karar, analiz, tasarım ve gerçekleme olarak özetlenmiştir [11]. öncelikle yeni bir dilin gerçekten gerekli olup olmadığına karar verilir. Mevcut diller ve araçlar yetersiz kalıyorsa yeni bir dil tasarlamak mantıklı olabilir. Alana özgü notasyonların ve veri yapılarının kullanılmasının avantajları düşünülür. Analiz safhasında dokümanlardan, uzmanlardan ve alandaki programların kaynak kodlarından faydalanarak alana özgü bilgi toplanır. Daha sonra, analiz aşamasında elde edilen bilgiler ışığında yeni dil tasarlanır. Son olarak bir dil geliştirme aracı yardımıyla yeni dil gerçeklenir.

3.2.1 Karar

Alana özgü dil geliştirme sürecinin ilk adımı karar adımıdır. Bu adımda yeni bir dil geliştirmek, var olan bir dili kullanmak veya genel amaçlı bir dili geliştirmek arasında seçim yapılır.

Günümüzde problem alanlarındaki gelişmeler çok fazla olduğu için genel amaçlı diller yetersiz kalmakta, alana özgü dillere olan ihtiyaç artmaktadır [13].

3.2.2 Analiz

Alana özgü dil geliştirmeye karar verdikten sonra analiz süreci başlar. Bu süreçte teknik dokümanlardan, alanın uzmanlarından, çeşitli analiz yöntemlerinden ve var olan program kodlarından yola çıkarak alan hakkında bilgi toplanır [11]. Analiz aşamasında dilin amacı belirlenmelidir. Alana özgü terimler, diyagramlar ve veri yapıları belirlenir.

3.2.3 Tasarım

Hoare, iyi bir dil tasarımının genel kriterlerini basitlik, güvenlik, hızlı kod üretme, etkin nesne kodu ve okunabilirlik olarak belirlemiştir [7].

Dil tasarlamada verilmesi gereken kararlardan biri dilin metin tabanlı veya grafik tabanlı olarak tasarlanmasıdır. Grafiksel notasyon daha basit görünse de alanla ilgili tüm kavramları ifade edemeyebilir [14]. Alana özgü kavramlar metin tabanlı dillerle daha net bir biçimde ifade edilebilir. Alana özgü bir dil kullanmak zorunda

kalan uzmanların öncelikle grafik tabanlı bir üzerinde geliştirme yapmak istedikleri gözlemlenmiştir. Ancak metin tabanlı bir dil kullanan uzmanlar başlarda biraz zorluk çekseler de dile hakim olduktan sonra kendilerini metin tabanlı dillerde daha rahat ifade edebilmektedirler.

3.2.4 Gerçekleme

Alana özgü dil tasarlandıktan sonra gerçekleme aşamasına geçilir. Bu aşamada çeşitli yöntemler kullanılabilir. Alana özgü diller genel amaçlı bir programlama diline gömülü olarak gerçeklenebilirler. Örneğin FPIC dili resim çizimi için SML diline gömülü olarak tasarlanmış bir dildir [15]. Gömülü diller başka bir dile entegre oldukları için o dilin tüm yeteneklerini kullanabilirler [14]. Bu özelliklerine rağmen gömülü bir dili kullanmak programlama bilgisine sahip olmayanlar için karmaşık olabilir. Çünkü bu dili kullanmak isteyen biri orijinal dilin söz dizimine ve kısıtlarına hakim olmak zorundadır.

Derleyici veya başka dile dönüştüren bir sistem kullanmak ise başka bir gerçekleme yöntemidir [14]. Bu yöntem kullanılarak gerçeklenen dilde programlama bilgisi olmayan kişiler algoritmalarını daha rahat ifade edebilirler. Bu diller, dil geliştirme ortamının sağladığı kod renklendirme, tamamlama gibi özellikleri de kullanabilmektedir. Bu da üretkenliği artırmaktadır.

Bu yöntemlerin dışında ön işlemci, yorumlayıcı gibi farklı yöntemler de bulunmaktadır. Ön işlemci yönteminde alana özgü dilde tanımlanmış olan yapılar var olan başka bir dile dönüştürülür. Örneğin kullanıcı ara yüzü oluşturmak için kullanılan AUI dili ön işlemci yöntemi kullanılarak tasarlanmıştır [16]. Yorumlayıcı yönteminde alana özgü dilde tanımlanmış olan yapılar standart bir getir-çöz-yürüt çevrimi kullanılarak yorumlanır. Bu yaklaşım dinamik bir karaktere sahip olan veya yorumlanan kodun çalışma süresinin önemli olmadığı diller için uygun olmaktadır. Örneğin dil işlemek için kullanılan ASTLOG dili yorumlayıcı yöntemi kullanılarak tasarlanmıştır [17].

3.3 Alana Özgü Dil Geliştirme Araçları

Alana özgü dil geliştirmek dil geliştirme tecrübesi gerektirdiği için yardımcı araçlar kullanmak kolaylık sağlayacaktır. Dil geliştirmek için tasarlanmış bir çok araç

bulunmaktadır. MPS, XText ve Spoofax bu araçlardan bazılarıdır. Dil geliştirme araçlarının bir karşılaştırması [18]'te verilmektedir.

MPS [19] metin ve tablo tabanlı dil geliştirmeyi desteklemektedir. 2014 yılında grafik tabanlı notasyonu da destekleyeceği duyurulmuştur. MPS Java dili üzerinde çalıştırılmaktadır. Kod üretme işlemi Java dilinde tanımlanabilen dönüşüm kuralları kullanılarak yapılmaktadır. Açık kaynak kodlu bir araçtır. MPS ile tanımlanan dillerin kolaylıkla genişletilebilir olması MPS'nin güçlü yönü olarak sayılabilir. MPS Eclipse tabanlı bir dil değildir ancak 2013 yılı içerisinde Eclipse desteğinin sağlanması planlanmaktadır.

XText [20] kullanılarak metin tabanlı diller tasarlanabilir. XText Eclipse Modeling Framework (EMF) üzerinde Java tabanlı olarak çalışmaktadır. Kod üretme kuralları Xtend [21] kullanılarak tanımlanmaktadır. Açık kaynak kodlu bir araçtır. Ölçeklenebilir ve genişletilebilir mimariye sahip olması XText'in güçlü yönlerini oluşturmaktadır. Editörleri geliştirebilmek için çok fazla Java kodunun gerekli olması ise XText'in zayıf yönüdür.

Spoofax [22] metin tabanlı dil tasarlamaya imkan sağlar. Eclipse üzerinde Java kullanılarak çalıştırılmaktadır. Dilin söz dizimi Syntax Definition Formalism (SDF) kullanılarak tasarlanmaktadır. Örnek bir söz dizimi kuralı aşağıda görülmektedir:

```
"module" ID Definition*          -> Start {cons("Module")}
"entity" ID "{" Property* "}"    -> Definition {cons("Entity")}
ID ":" Type                      -> Property {cons("Property")}
ID                               -> Type {cons("Type")}
```

Yukarıda tanımlanmış olan söz dizimine uygun olarak Spoofax'te yazılmış örnek bir dil aşağıda görülmektedir:

```
module example
entity User {
  name     : String
  password : String
  homepage : URL
}
```

Kod üretme kuralları Stratego dili kullanılarak tanımlanmaktadır. Burada tanımlanan örnek dil için tasarlanan kod üretme kuralları aşağıda görülmektedir:

```
to-java:
    Module(x, d*) ->
    $[ package [x];
       [d'*]
     ]
    with
      d'* := <to-java> d*

to-java:
    Entity(x, p*) ->
    $[ class [x] {
           [p'*]
       }
     ]
    with
      p'* := <to-java> p*

to-java:
    Property(x, Type(t)) -> $[
      private [t] [x];
      public [t] get_[x] {
          return [x];
      }

      public void set_[x] ([t] [x]) {
          this.[x] = [x];
      }
  ]
```

```
  to-java:
    Type(t) -> t
```

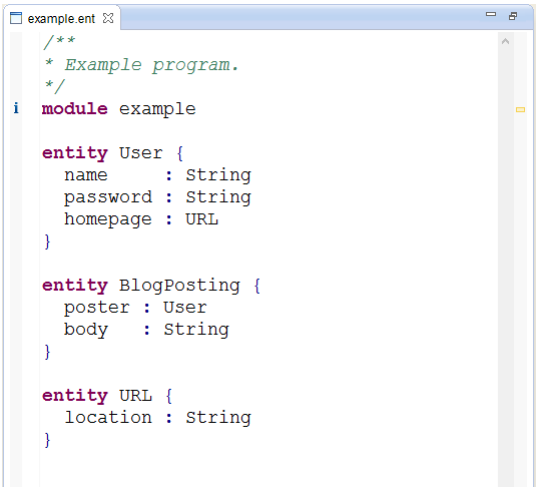
Bu dilde yazılmış olan programın Java karşılığı aşağıda görülmektedir:

```
package example;
class User {
    private String name;
    public String get_name {
        return name;
    }
    public void set_name (String name) {
        this.name = name;
    }
    private String password;
    public String get_password {
        return password;
    }
    public void set_password (String password) {
        this.password = password;
    }
    private URL homepage;
    public URL get_homepage {
        return homepage;
    }
    public void set_homepage (URL homepage) {
        this.homepage = homepage;
    }
}
```

Spoofax açık kaynak kodlu bir araçtır. Metinsel dil tanımlamaya imkan sağlaması ve dilin modüler olması Spoofax'in güçlü yönlerini oluşturmaktadır. Grafik editörünün olmaması ve EMF ile entegre olmaması ise zayıf yönleri arasında sayılabilir.

Bu çalışmada Spoofax dil geliştirme aracı [1] kullanılmıştır. Spoofax dilin gramerini ve kod üretim kurallarını ifade etmeyi sağlar, çözümleyici ve kod dönüştürücü gibi gerekli bileşenleri oluşturur. Spoofax editörünün sahip olduğu bazı özellikler aşağıdadır:

- Kod renklendirme ile geliştirilen dilde tanımlanmış özel sözcükler renklendirilerek kodun daha rahat okunması sağlanmaktadır. İstenirse kullanılan renkler özel olarak değiştirilebilir. Spoofax'in kod renklendirme özelliği Şekil 3.1'de görülmektedir.

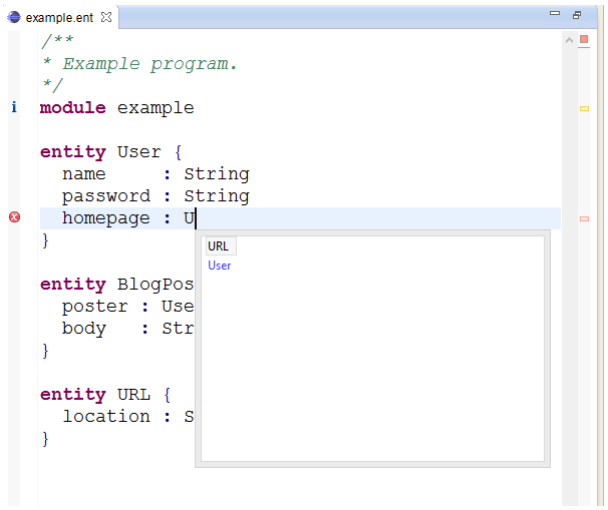


Şekil 3.1: Spoofax ile kod renklendirme.

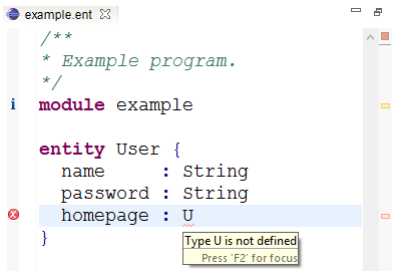
- Kod tamamlama özelliği Spoofax'in desteklediği bir özelliktir. Ctrl ve boşluk tuşuna basılarak kod tamamlama özelliği tetiklenebilir. Kod tamamlama özelliği Şekil 3.2'de görülmektedir.

- Hataların tespit edilmesi ve hata olduğunda uyarı verilmesi de Spoofax'in gelişmiş özellikleri arasındadır. Spoofax'ta tanımlanan söz dizimine uygun olmayan kodlar hatalı olarak nitelendirilir. Aynı zamanda dile özel kısıtlar ve bu kısıtlara uyulmaması durumunda verilecek hata mesajları da tanımlanabilir. Bu durumda hatalı kodun altı kırmızı çizgi ile çizilmekte ve hatalı kodun üzerine gelindiğinde yazılan mesaj gösterilmektedir. Hata tespit özelliği Şekil 3.3'te görülmektedir.

[1] http://strategoxt.org/Spoofax



Şekil 3.2: Spoofax ile kod tamamlama.



Şekil 3.3: Spoofax ile hata tespiti.

- Kod katlama özelliği kullanılarak kodun okunurluğu artırılabilir. Katlanan kodun üzerine gelindiğinde kodun tamamı gösterilmektedir. Kod katlama özelliği Şekil 3.4'te görülmektedir.



Şekil 3.4: Spoofax ile kod katlama.

4. ÜST SEZGİSELLER İÇİN ALANA ÖZGÜ DİL TASARLANMASI

4.1 Analiz

Analiz aşamasında HyFlex'te yazılmış örnek kodlar incelenmiş ve geliştirilecek dilde nelere ihtiyaç duyulacağı tespit edilmiştir. Bu ihtiyaçlar aşağıda listelenmiştir:

- Sezgisel seçim yönteminin seçilmesi
- Kabul yönteminin seçilmesi
- Başlatma
- Sezgisellerin seçilmesi
- Sezgisellerin uygulanması
- Çözümlerin kabul edilmesi
- Özel yapıların tanımlanması

4.1.1 Sezgisel seçim yönteminin seçilmesi

Üst-sezgisel algoritmaları sezgisel seçimi için farklı mekanizmalara ihtiyaç duymaktadır. Bazı mekanizmalar (basit rastgele seçim gibi) seçimlerini geçmişte yapılmış olan seçimlerden bağımsız olarak yaparken, bazıları ise (rastgele permütasyon) geçmişte seçilmiş sezgisellerin bilgisine ihtiyaç duyarlar. Ayrıca belirlenmiş bir tipteki sezgiseller arasından bir seçim yapmak da istenebilir (mutasyon gibi). Bunlar gibi farklı durumları desteklemek amacıyla öncelikle bir seçicinin oluşturulması gerekir. İhtiyaç duyulan sezgisel tipleri mutasyon, çaprazlama, bölgesel arama veya burada sayılanların bir kombinasyonu olabilir. Dilde bir sezgisel seçmek için tanımlanmış yöntemler bulunmalıdır. Ayrıca özelleşmiş sezgisel seçim yöntemlerinin tasarlanan dilde tanımlanabilmesi de ihtiyaçlar arasındadır. Geçerli sezgisel tipleri mutasyon, çaprazlama, bölgesel arama veya burada sayılanların bir kombinasyonu olabilir.

Tüm sezgisel tipleri arasından basit rastgele seçim yapan bir seçici aşağıdaki gibi tanımlanabilir:

```
hs = selection simple_random
```

Tüm sezgisel tipleri arasından rastgele permütasyon yöntemini kullanarak seçim yapan bir seçici aşağıdaki gibi tanımlanabilir:

```
hs = selection random_permutation
```

Tüm sezgisel tipleri arasından açgözlü seçim yöntemini kullanarak seçim yapan bir seçici aşağıdaki gibi tanımlanabilir:

```
hs = selection greedy
```

Tüm sezgisel tipleri arasından pekiştirmeli öğrenme yöntemini kullanarak seçim yapan bir seçici aşağıdaki gibi tanımlanabilir:

```
hs = selection reinforcement_learning
    defaultScore:1 min:1 max:5 inc:1 dec:0
```

Sadece mutasyon türünden sezgisel tipleri arasından basit rastgele seçim yapan bir seçici aşağıdaki gibi tanımlanabilir:

```
hs = mutation selection simple_random
```

Sadece mutasyon ve yerel arama türünden sezgisel tipleri arasından basit rastgele seçim yapan bir seçici aşağıdaki gibi tanımlanabilir:

```
hs = mutation+local_search selection simple_random
```

4.1.2 Kabul yönteminin seçilmesi

Üretilen çözümlerin hangi kurallara göre kabul edileceğinin belirlenmesi gerekir. Dilde bazı temel kabul yöntemlerinin tanımlanması gerekir. Aşağıdaki örnekte tüm çözümleri kabul eden bir kabul edici tanımlanmıştır:

```
ac = acceptance accept_all
```

Sadece iyileşen çözümleri kabul eden bir kabul edici aşağıdaki şekilde tanımlanabilir:

```
ac = acceptance only_improving
```

Sadece iyileşen ve aynı kalan çözümleri kabul eden bir kabul edici aşağıdaki şekilde tanımlanabilir:

```
ac = acceptance improving_or_equal
```

İyileşen çözümleri kabul edip iyileşmeyenleri belirli bir olasılıkla kabul eden bir kabul edici aşağıdaki şekilde tanımlanabilir:

```
ac = acceptance nonimproving_with_probability 0.5
```

4.1.3 Başlatma

Problemde kullanılacak çözüm kümesinin oluşturulması ve gerekli konfigürasyonların yapılması başlatma aşamasında yapılmalıdır. Bu aşamada çözümler yaratılmalı ve ilk değerleri atanmalıdır. Ayrıca çözümlerin amaç fonksiyonlarının ilk değerlerinin atanması da bu aşamada yapılmalıdır.

Aşağıdaki ifade 10 tane çözüm üretecek ve rastgele ilk değer atama işlemini gerçekleştirecektir:

```
initialize solution 10
```

Eğer çözüm sayısı verilmemişse sadece bir çözüm kullanılır. Aşağıdaki örnekte sadece bir çözüm kullanılmıştır:

```
initialize solution
```

4.1.4 Sezgisellerin seçilmesi

Sezgisel seçim yöntemleri tanımlandıktan sonra `next` komutu kullanılarak seçiciden sıradaki sezgisel için istekte bulunulur. Aşağıdaki örnekte daha önceden tanımlanmış `hs` seçicisinden yeni bir sezgisel istenmektedir:

```
h = next hs
```

4.1.5 Sezgisellerin uygulanması

Seçilen sezgisellerin bir çözüme uygulanması gerekir. Seçilen sezgisel bir çözüme uygulandıktan sonra yeni bir çözüm elde edilir.

Bir sezgisel bir çözüme `apply` ifadesiyle uygulanır. Örneğin, aşağıdaki kod (`h`) sezgiselini (`s`) çözümüne uygular ve üretilen çözümü yeni bir değişkene atar (`n`):

```
n = apply h s
```

4.1.6 Çözümlerin kabul edilmesi

Üretilen bir çözüm, kabul edicinin kontrollerinden sonra kabul edilebilir veya reddedilebilir. Bu işlem eski ve yeni çözümleri parametre olarak alır ve yeni çözümü kabul edip etmemeye karar verir. Eğer yeni çözüm kabul edilirse bu çözüme hangi işlemin uygulanacağı da belirtilir (eski çözümü yenisiyle değiştirmek gibi). Aşağıda verilen örnekte eğer kabul koşulları yeterliyse kabul edici (`ac`) eski çözümü (`s`) yeni çözüm (`n`) ile değiştirecektir:

```
check ac n replace s
```

4.1.7 Yapı tanımlanması

Dilde tanımlanan temel sezgisel seçim yöntemleri ve yeni çözümlerinin kabul yöntemleri kullanıcıların oluşturmak istedikleri üst sezgisel algoritmaları için yetersiz kalabilir. Böyle bir durumda kullanıcılar kendi seçim veya kabul yöntemlerini tanımlamak isteyebilir. Buna izin veren yapıların dilde tanımlanmasına ihtiyaç duyulmaktadır.

Dili kullanan kişiler sezgisel seçmek için kendi yöntemlerini tanımlamak isteyebilirler. Bu durumda yapılardan yararlanabilirler. Bir yapı ifadelerden ve metotlardan oluşur.

Aşağıda verilen örnekte sezgisel seçimi için kullanılmak üzere (`myHeuristic`) isimli bir yapı tanımlanmıştır. `permute` komutuyla var olan sezgisellerin bir permütasyonu oluşturulmuş ve bir diziye atanmıştır. `next` metodu permütasyon dizisinden bir sezgiseli seçmek ve döndürmek için kullanılmıştır.

```
define myHeuristic
{
    h[] = permute heuristic
    current = -1

    next() : Heuristic
    {
        current = current + 1
        return h[current]
    }
}
```

Önceden tanımlı sezgisel yöntemleri kullanılırken de bu yapı ile tanımlanmış sezgisel yöntemleri kullanılırken de aynı metot kullanılır. Aşağıdaki örnekte görüldüğü gibi `selection` anahtar sözcüğünden sonra tanımlanmış yapının ismi yazılır ve bu şekilde çağrım yapılır:

```
hs = mutation selection myHeuristic
h = next hs
```

Özel kabul yöntemleri tanımlanmak istenirse yine bu yapılar kullanılabilir. Aşağıda verilen örnekte kabul yöntemi olarak kullanılmak üzere `myAcceptance` isimli bir yapı tanımlanmıştır. Bu yapı içerisinde tanımlanan `check` metodu eski çözümü ve yeni çözümü parametre olarak almaktadır ve geriye `Boolean` değer dönmektedir. Çözümleri `better` komutunu kullanarak karşılaştırmakta ve yeni çözüm eskisinden daha iyiyse geriye `true` dönmektedir.

```
define myAcceptance
{
```

```
    check(exSolution, newSolution) : Boolean
    {
      return better newSolution exSolution
    }
}
```

Önceden tanımlı kabul yöntemleri kullanılırken de bu yapı ile tanımlanmış kabul yöntemleri kullanılırken de aynı metot kullanılır. Aşağıdaki örnekte görüldüğü gibi `acceptance` anahtar sözcüğünden sonra tanımlanmış yapının ismi yazılır ve bu şekilde çağrım yapılır:

```
accepter = acceptance myAcceptance
check accepter n replace s
```

4.1.8 Tip Sistemi

Tip sistemi bir fonksiyonun veya alt programın girdilerini ve çıktılarını tanımlamak için kullanılmaktadır. Birçok programlama dilinde fonksiyonların geri dönüş tiplerini ve aldıkları parametrelerin tiplerini de içermektedir. Tasarladığımız dilde bu tipler üst sezgisel algoritmalarının gerektirdiği şekilde oluşturulmaktadır. Örneğin sezgisel seçen bir fonksiyonun geri dönüş değeri bir sezgisel nesnesi olmaktadır. Tanımladığımız yöntemlerin aldıkları parametrelerin tiplerini ve fonksiyonların geri dönüş tiplerini kullanıcının bilmesine gerek yoktur. Ancak kullanıcı tarafından tanımlanan özel yapılarda kullanılan parametrelerin tipleri ve metotların geri dönüş tipi belirtilmelidir. Tasarlanan dilde kullanılabilir geçerli tipler `Heuristic`, `Integer`, `String`, `Double` ve `Boolean` olarak tanımlanmıştır. Aşağıdaki örnekte bir yapının aldığı parametrenin tip tanımlaması görülmektedir:

```
define myHeuristic(defaultValue:Double)
{
}
```

Tanımlanan yapıda parametre varsa bu parametreler yapının adından sonra parantezler içerisinde yazılmalıdır. Parametrenin adı, ":" işareti ve parametrenin tipi yazılmalıdır.

Metotlarda tip tanımlaması aşağıdaki gibi yapılmaktadır:

```
next() : Heuristic
{
}
```

Metodun adını izleyen parantezlerden sonra ":" işareti konulmalı, ardından geri dönüş tipi belirtilmelidir. Bu örnekte geri dönüş tipi `Heuristic` nesnesidir.

5. ÜST SEZGİSELLER İÇİN ALANA ÖZGÜ DİL GERÇEKLENMESİ

5.1 Söz Dizimi

Dil tasarlanırken öncelikle sözdiziminin tanımlanması gerekir. Spoofax platformunda sözdizimi "Syntax Definition Formalism" (SDF) notasyonu kullanılarak tanımlanır [23]. Dilin sözdiziminin bir özeti Şekil 5.1 ve 5.2'de görülmektedir.

Spoofax anaçatısında, tasarlanan dilin söz dizimi kuralları ve kod üretim kuralları tanımlanır. Spoofax bu kurallardan yola çıkarak ayrıştırıcı ve kod oluşturucu gibi gerekli bileşenleri oluşturur. SDF'de tanımlanmış iki üretim kuralı aşağıdaki örnekte görülebilir:

```
ID "=" Expression
        -> Stm        {cons("Assignment")}

HeuristicType? "selection"
    HeuristicSelectionType Params*
        -> Expression {cons("SelectionType")}
```

Bu örnekte, `Stm` isimli terminal olmayan eleman, terminal olmayan `ID` elemanına, onu izleyen eşittir işaretine ve terminal olmayan `Expression` elemanına dönüştürülür. Terminal olmayan `Expression` elemanı isteğe bağlı olarak terminal olmayan `HeuristicType` elemanı, `selection` anahtar kelimesi, terminal olmayan `HeuristicSelectionType` isimli eleman ve istenen sayıda terminal olmayan `Params` elemanlarına dönüştürülür. SDF'de yer alan bazı sözdizimi kalıpları Çizelge 5.1'de listelenmiştir [24].

Üst-sezgiseller alanındaki kavramları ifade eden bazı özel terimler dilde anahtar sözcük olarak tanımlanmıştır. Bu anahtar sözcüklerden bazıları `best`, `first`, `random`, `select`, `solution`, `heuristic`, `apply` ve `replace`'tir. Anahtar sözcük olarak tanımlanmış sözcükler değişken adı olarak kullanılamazlar.

```
Stm* StructDcl*
  -> Start          {cons("TypesAndStatements")}
ID "=" Expression
  -> Stm            {cons("Assignment")}
Array "=" ArrayExpr
  -> Stm            {cons("ArrayAssignment")}
"check" ID ID "replace" ID ("[" INT "]")?
  -> Stm            {cons("Acceptance")}
"initialize" "solution" INT?
  -> Stm            {cons("Initialization")}
"define" ID "{" Stm* Methods* "}"
  -> StructDcl    {cons("StructDeclaration")}
"define" ID "(" StructParams ")" "{" Stm* Methods* "}"
  -> StructDcl    {cons("StructDclWithParams")}
ID"(" MethodParams? ") : " VariableType? "{" Stm* "}"
  -> Methods        {cons("Methods")}
HeuristicType? "selection" HeuristicSelectionType Params*
  -> Expression    {cons("SelectionType")}
"acceptance" AcceptanceSelectionType AcceptanceParams*
  -> Expression    {cons("AcceptanceType")}
"next" ID
  -> Expression    {cons("NextHeuristicSelection")}
"apply" Arguments
  -> Expression    {cons("ApplyHeuristic")}
AdjExpr HSType
  -> Expression    {cons("HSSelection")}
"permute" "heuristic"
  -> ArrayExpr     {cons("PermuteHeuristic")}
AdjExpr "solution" INT "!"
  -> ArrayExpr     {cons("ArraySelection")}
{Heuristic "+"}+
  -> HeuristicType{cons("MultipleHeuristics")}
"update" ID
  -> Stm            {cons("UpdateHeuristicScore")}
"best"
  -> AdjExpr        {cons("Best")}
"first"
  -> AdjExpr        {cons("First")}
"random"
  -> AdjExpr        {cons("Random")}
{Arg ","}+
  -> StructParams {cons("StructParams")}
ID ":" VariableType
  -> Arg            {cons("Arg")}
```

Şekil 5.1: Tasarlanan dilin gramer kuralları.

```
"improved"
  -> Expr       {cons("Improved")}
ID "in" ID
  -> Expr         {cons("CheckType")}
ID "[]"
  -> Array        {cons("Array")}
"heuristic"
  -> HSType       {cons("HeuristicType")}
"solution"
  -> HSType       {cons("SolutionType")}
"#" HSType
  -> Size         {cons("Size")}
Size
  -> Expression   {cons("SizeExpression")}
"if" Expr "{" Stm* "}" ("else" "{" Stm* "}")
  -> Stm          {cons("IfElse")}
"if" Expr "{" Stm* "}"
  -> Stm          {cons("If")}
"loop" LoopCondition "{" Stm* "}"
  -> Stm          {cons("Loop")}
"#" HSType
  -> Size         {cons("Size")}
Size
  -> Expression   {cons("SizeExpression")}
NUMBER"%"
  -> LoopCondition{cons("TimeLoopCond")}
ID "in" "[" ParamType ":" ParamType "]"
  -> LoopCondition{cons("CntControlLoopCond")}
ID ID
  -> Arguments    {cons("ApplyArg")}
ID
  -> Arguments    {cons("ApplyArg")}
ID Arrays
  -> Arguments    {cons("ApplyArrayArg")}
ArrayWithIndex ID
  -> Arguments    {cons("ApplyArrayArg")}
Array
  -> Arrays       {cons("Arrays")}
ArrayWithIndex
  -> Arrays       {cons("Arrays")}
"return" ReturnStm
  -> Stm          {cons("ReturnStm")}
ArrayWithIndex
  -> ReturnStm    {cons("ReturnArrayWithIndex")}
```

Şekil 5.2: Tasarlanan dilin gramer kuralları.

Çizelge 5.1: SDF'de yer alan bazı söz dizimi kalıpları.

Kalıp	Açıklama	Örnekler
A*	Sıfır veya daha fazla A sembolü.	Stm*, [a-zA-Z]*
A+	Bir veya daha fazla A sembolü.	TypeDec+, [a-zA-Z]+
A?	İsteğe bağlı A sembolü.	Expr?, [fFdD]?
{A0 A1}*	A1 sembolü ile ayrılmış sıfır veya daha fazla A0 sembolü.	{Exp ","}*, {FormalParam ","}*
{A0 A1}+	A1 sembolü ile ayrılmış bir veya daha fazla A0 sembolü.	{Id "."}+, {InterfaceType ","}+
A0 | A1	A0 veya A1 sembolü.	{Expr ","}* | LocalVarDec

5.2 Kod üretme

Spoofax anaçatısında, tasarlanan dilin kod üretme kuralları Stratego/XT dili kullanılarak yazılır [25]. Her kod üretme kuralı `to-java` ifadesiyle başlar. Kod üretme kuralı iki ana kısımdan oluşur. Sözdiziminde tanımlanmış olan etiketler kullanılarak kod üretme kuralları tanımlanır. Kod üretme kuralının çalışabilmesi için söz dizimine uygun etkiketleri içermesi gerekir.

Kod üretme işlemi katar değiştirme (string interpolation) yöntemi kullanılarak yapılır. "$" işaretinden sonra gelen köşeli parantezlerin arasında katar değiştirme işlemi yapılır. Dolar işareti olmadan kullanılan köşeli parantezler arasına değişkenler konulabilir.

Örnek bir kod üretme kuralı aşağıda görülmektedir:

```
to-java:
ApplyHeuristic(heuristicName) -> $[

curr_obj_function["["]0["]"] =
    heuristicApplier.apply(problem, [heuristicName],
                           0, solutionmemorysize);

delta = curr_obj_function["["]solutionmemorysize["]"] -
        curr_obj_function["["]0["]"];
```

```
]
```

Bu örnekte söz diziminde yer alan `ApplyHeuristic` kuralı için kod üretme kuralı görülmektedir. Söz dizimindeki kuralda bir tane terminal sembol yer aldığı için bu sembol üretim kuralında `heuristicName` isimli parametre olarak yer almaktadır. "$" işareti ve köşeli parantezler arasında yer alan bölüm üretilen Java koduna karşılık gelmektedir. Köşeli parantezler arasına yazılmış `heuristicName` parametresinin çalışma anındaki değeri bu parantezlerin yerine yazılmaktadır. Köşeli parantezler Stratego/XT dilinde katar değiştirme işleminin başladığı ve bittiği noktaları belirlemek için kullanılmaktadır. Ancak bazı durumlarda üretilen Java kodunun köşeli parantez işaretlerini içermesi de istenebilir. Bu durumda bu örnekte görüldüğü gibi köşeli parantez içine istenen köşeli parantez yazılarak Java kodunda yer alması sağlanır.

Örneğin `apply h` komutu için yukarıdaki kod üretme kuralı çalışacak ve aşağıdaki Java kodu üretilecektir:

```
curr_obj_function[0] =
    heuristicApplier.apply(problem, h,
                           0, solutionmemorysize);

delta = curr_obj_function[solutionmemorysize] -
        curr_obj_function[0];
```

5.3 Gerçekleme

5.3.1 Döngüler

Üst sezgisel algoritmaları çalıştırılırken iteratif olarak tekrarlanması gereken ifadeler olabilir. Geliştirilen dilde bu ihtiyacı karşılamak amacıyla döngüler tanımlanmıştır.

Döngü tanımlamak için kullanılan söz dizimi kuralı aşağıda görülmektedir:

```
"loop" LoopCondition "{" Stm* "}"
    -> Stm  {cons("Loop")}
NUMBER"%"
    -> LoopCondition {cons("TimeLoopCond")}
```

```
ID "in" "[" ParamType ":" ParamType "]"
    -> LoopCondition {cons("CounterLoopCond")}
```

Döngüler `loop` anahtar sözcüğü kullanılarak tanımlanır. Bu sözcükten sonra döngünün çalışma koşulunu oluşturan bir `LoopCondition` elemanı ve süslü parantezler içerisinde çalıştırılacak komutlar yazılır. Döngünün koşulu olan `LoopCondition` iki şekilde tanımlanabilir. Birincisinde döngü programın çalışması için verilen sürenin belli bir yüzdesi boyunca çalışır. Bunun için bir sayı ve "%" işareti kullanılmaktadır. İkinci tür ise döngünün bir sayaç yardımıyla tekrarlanmasıdır. Bu durumda sayacın alt ve üst sınırı parametre olarak verilmelidir.

Süreli çalışan döngünün kullanıldığı durumda aşağıdaki kod üretme kuralı çalışmaktadır:

```
to-java:
Loop(TimeLoopCond(Time), Statements*) -> $[
while(!hasTimeExpired())
{
[Statements'*]
}
]
with
Statements'* := <to-java> Statements*
```

Süreli çalışan döngüler Java'da `while` döngülerine karşılık gelmektedir. Üretilen kodda bir döngü oluşturulmakta ve döngünün içinde çalışacak ifadeler üretilmektedir. Döngünün çalışma koşulunu kontrol etmek için ise HyFlex'in `hasTimeExpired()` metodu kullanılmaktadır. `hasTimeExpired()` metodu programın çalışması için geçen sürenin aşılıp aşılmadığını kontrol etmektedir.

Alt ve üst sınırları verilen bir sayaç kullanılarak çalışan döngü için aşağıdaki kod üretme kuralı çalışmaktadır:

```
to-java:
Loop(CounterLoopCond(indexName, min, max),
```

```
    Statements*) -> $[
for(int [indexName] = [min];
    [indexName] < [max]; [indexName]++)
{
[Statements'*]
}
]
with
Statements'* := <to-java> Statements*
```

Bir sayaç kullanılarak çalışan döngüler Java'da `for` döngülerine karşılık gelmektedir. Üretilen kodda bir `for` döngüsü oluşturulmakta, döngünün kullandığı sayacın başlangıç ve bitiş değerleri atanmaktadır. Döngünün içinde çalışacak olan ifadeler üretilmekte ve süslü parantezler arasına yazılmaktadır.

5.3.2 Koşullu dallanmalar

Programın çalışma esnasında kontrol edilmesi gereken bazı durumlar olabilir. Bu durumlar karşısında programın farklı dallara ayrılması ve bazı blokları koşullu olarak çalıştırılması istenebilir. Bu ihtiyacı karşılamak amacıyla koşullu dallanma ifadeleri tanımlanmıştır.

Koşullu dallanmalar için `if` ve `else` sözcükleri kullanılmaktadır. Söz dizimi kuralları aşağıda görülmektedir:

```
"if" Expr "{" Stm* "}"
    -> Stm  {cons("If")}
"if" Expr "{" Stm* "}" ("else" "{" Stm* "}")
    -> Stm  {cons("IfElse")}
```

Söz dizimi `if` anahtar sözcüğü, bir koşul ve koşulun doğru olması durumunda çalıştırılacak ifadelerden oluşmaktadır. Koşulun doğru olmadığı durumda `else` anahtar sözcüğü ile birlikte bu durumda çalıştırılacak ifadeler tanımlanabilir. Koşullar için kullanılan söz dizimi kuralları aşağıda görülmektedir:

```
"improved"
```

```
    -> Expr   {cons("Improved")}
ID "in" ID
    -> Expr {cons("CheckType")}
```

Gerçeklenen dilde iki tür koşul tanımlanabilir. İlk koşulda üretilen bir çözüm adayı ile eski çözüm adayı arasında iyileşme olup olmadığı kontrol edilir. Bunun için `improved` sözcüğü kullanılmaktadır. İkinci tür koşul ise seçilen bir sezgiselin belirli bir sezgisel tipine ait olup olmadığını kontrol etmektedir. Bunun için kullanılan söz dizimi kuralında seçilen sezgiselin adı, `in` sözcüğü ve aranan sezgisel tipi yer almaktadır.

Koşullu dallanmalar için kullanılan kod üretme kuralı aşağıda görülmektedir:

```
to-java:
IfElse(Condition, TrueBranch, ElseBranch) -> $[
if ([Condition']) {
[TrueBranch']
}
else {
[ElseBranch']
}
]
with
    Condition'  := <to-java> Condition;
    TrueBranch' := <to-java> TrueBranch;
    ElseBranch' := <to-java> ElseBranch
```

Bu kural çalıştırılacak koşulu, koşulun doğru olması durumunda çalışacak ifadeleri ve koşulun yanlış olması durumunda çalışacak ifadeleri parametre olarak almaktadır.

`else` durumu olmayan koşullu dallanmalar için kullanılan kod üretme kuralı aşağıda görülmektedir:

```
to-java:
If(Condition, TrueBranch) -> $[
```

```
if ([Condition']) {
[TrueBranch']
}
]
with
    Condition'  := <to-java> Condition;
    TrueBranch' := <to-java> TrueBranch
```

Bu kural çalıştırılacak koşulu ve koşulun doğru olması durumunda çalışacak ifadeleri parametre olarak almaktadır.

Üretilen bir çözüm adayı ile eski çözüm adayı arasında iyileşme olup olmadığını kontrol eden koşul için çalışan kod üretme kuralı aşağıda görülmektedir:

```
to-java:
Improved() -> $[
delta > 0
]
```

Seçilen bir sezgiselin belirli bir sezgisel tipine ait olup olmadığını kontrol eden koşul için çalışan kod üretme kuralı aşağıda görülmektedir:

```
to-java:
CheckType(Heuristic, heuristicType) -> $[
externalMethods.contains(
    externalMethods.getHeuristicList("[heuristicType]", problem),
[Heuristic].getHeuristicIndex())
]
```

5.3.3 Sezgisel seçim yönteminin tanımlanması

Bir sezgisel seçmek için tanımlanmış geçerli yöntemler basit rastgele seçim, rastgele permütasyon seçim, pekiştirmeli öğrenme (reinforcement learning), açgözlü (greedy) seçim yöntemleridir. Bu yöntemler dışında ihtiyaç duyulan başka seçim yöntemlerinin tanımlanması ve kullanılması ilerleyen bölümlerde anlatılmaktadır. Bir seçici

oluşturmak için kullanılan söz dizimi kuralı sezgisel tipi, `selection` anahtar sözcüğü ve seçim yöntemidir. Geçerli sezgisel tipleri mutasyon, çaprazlama, yerel arama veya burada sayılanların bir kombinasyonu olabilir.

Bir seçici tanımlandıktan sonra `next` işlemi seçiciden bir sonraki sezgiseli istemek için kullanılır. Örneğin aşağıdaki kod mutasyon tipinden rastgele bir sezgisel seçen bir seçicinin (`hs` değişkenine atanmış) nasıl oluşturulduğunu ve sezgisellerin (`h` değişkenine atanmış) bu seçiciden nasıl isteneceklerini göstermektedir:

```
hs = mutation selection simple_random
h = next hs
```

Birden fazla sezgisel tipi arasından seçim yapmak istenildiğinde istenen tipler bir kombinasyon halinde belirtilmelidir. Örneğin rastgele bir mutasyon veya bölgesel arama tipinden bir sezgisel seçmek için aşağıdaki komut kullanılabilir:

```
hs = mutation+local_search selection simple_random
```

Sezgisel seçimi için kullanılan söz dizimi kuralı aşağıda görülmektedir. Bu kural isteğe bağlı bir sezgisel tipi, `selection` anahtar sözcüğü, sezgisel seçim yöntemi ve istenen sayıda parametreden oluşmaktadır.

```
HeuristicType? "selection" HeuristicSelectionType Params*
  -> Expression   {cons("SelectionType")}
```

Sezgisel tipi `HeuristicType` bir veya birden fazla sezgisel tipinden oluşabilir. Birden fazla sezgisel tipinin seçilmesi durumunda kullanılacak sezgisel türleri "+" işaretiyle ayrılmalıdır. Bunun için tanımlanmış üretim kuralı aşağıda görülebilir:

```
{Heuristic "+"}+
    -> HeuristicType {cons("MultipleHeuristics")}
```

Terminal olmayan `Heuristic` elemanı ise aşağıda görülen sezgisel terminal elemanlarına dönüştürülür. Sezgisel türü belirtilirken `mutation` veya `mutation_heuristic` yazmak arasında bir fark yoktur. Kullanıcılara kolaylık sağlaması açısından söz dizimi iki yazımı da destekleyecek şekilde gerçeklenmiştir.

```
"mutation"                -> Heuristic
"mutation_heuristic"      -> Heuristic
"local_search"            -> Heuristic
"local_search_heuristic" -> Heuristic
"crossover"               -> Heuristic
"crossover_heuristic"     -> Heuristic
```

Terminal olmayan `HeuristicSelectionType` elemanı aşağıda görülen sezgisel seçim yöntemlerine dönüştürülür.

```
"simple_random"           -> HeuristicSelectionType
"random_permutation"      -> HeuristicSelectionType
"greedy"                  -> HeuristicSelectionType
"reinforcement_learning"  -> HeuristicSelectionType
ID                        -> HeuristicSelectionType
```

Burada anlatılan sezgisel seçim yöntemleri basit rastgele seçim, rastgele permütasyon seçimi, aç gözlü seçim ve pekiştirmeli öğrenmedir. `ID` elemanı ise kullanıcının kendisinin tanımladığı yöntemleri kullanmak istemesi durumunda kullanılır. Kullanıcı bu yöntemler dışında başka bir yöntem kullanmak istediğinde gerekli tanımlamaları yapar ve burada yöntemin adını yazarak kullanabilir.

Sezgisel seçim yöntemlerinin parametrelerini üretmek için kullanılan `Params` elemanı aşağıdaki terminallere dönüştürülmektedir:

```
"defaultScore:" NUMBER    ->Params
"min:"  NUMBER            ->Params
"max:"  NUMBER            ->Params
"inc:"  NUMBER            ->Params
"dec:"  NUMBER            ->Params
```

Sezgisel seçmek için tanımlanmış yöntemlerden sadece pekiştirmeli öğrenme yöntemi parametre aldığı için bu yöntem kullanıldığında bu parametreler anlam kazanmaktadır. İstenen sayıda parametre kullanılabilir.

Pekiştirmeli öğrenme yöntemi her sezgisel için bir başarım puanı tutar. Sezgisellerin varsayılan başarım puanları, alabilecekleri minimum ve maksimum puanlar belirlenir. Bir sezgisel bir çözüme uygulandığında çözümde iyileşme görülüyorsa sezgiselin puanı artırılır, aksi durumda azaltılır. Artırma ve azaltma için kullanılacak puanlar da yine bu parametrelerle belirlenmektedir.

Sezgisel seçim yöntemi tanımlanırken sezgisel tipi isteğe bağlı olarak belirtilir. İsteğe bağlı olarak belirtilen elemanlar için iki farklı kod üretme kuralı yer almaktadır. Bu elemanlar kullanılmadığında kod üretme kuralındaki parametrelerinin yerine `None()` ifadesi, kullanıldığında ise `Some(...)` ifadesi yazılmaktadır.

Kod üretme kurallarının hangi durumda çalışacaklarını belirlemek için koşullar kullanılabilir. Bu koşullar `where` sözcüğü kullanılarak belirtilir.

Aşağıdaki örnekte tüm sezgisel tipleri arasından basit rastgele seçim yapan bir seçiciyi tanımlayan kod üretilmektedir:

```
to-java:
Assignment(VarName,
    SelectionType(None(), selectionMethod, params*)) -> $[
SimpleRandomSelection [VarName] =
    new SimpleRandomSelection(hList, rng);
]
where
<eq>(selectionMethod, "simple_random");
<set-global> (VarName, "simple_random")
```

`Assignment` kuralı bir değişken adı ve bir ifadeden oluşmaktadır. Buradaki ifade sezgisel seçim yönteminin tanımlanmasıdır. Sezgisel seçim yönteminin belirlenmesi için kullanılan söz dizimi kuralı `SelectionType` üç parametre almaktadır. Bunlar sezgisel tipi, seçim yöntemi ve yöntemin parametreleridir. Bu kod üretim kuralı sezgisel tipinin belirtilmediği durumlarda çalışmaktadır. Bu yüzden `SelectionType`'ın ilk parametresi `None()` olmuştur. Buradaki üretim kuralına bir çalışma kısıtı konulmuştur. Bu kural `selectionMethod` parametresi sadece `simple_random`'a eşit olduğunda çalışmaktadır. `where` koşulunda bir ifadenin daha yer aldığı görülmektedir. `<set-global>` ifadesi seçim yönteminin adını global

bir değişkende tutmak için kullanılmaktadır. Hangi değişkenin hangi seçim yöntemini kullandığının tutulması için bu yönteme başvurulmuştur. Bu bilgi pekiştirmeli öğrenme gibi seçim yöntemleri kullanıldığında lazım olmaktadır.

Pekiştirmeli öğrenme için örnek bir kod üretme kuralı aşağıda görülmektedir:

```
to-java:
Assignment(VarName,
    SelectionType(None(), selectionMethod, params*))-> $[
ReinforcementLearningSelection [VarName] =
    new ReinforcementLearningSelection(hList,
    "[params']", rng);
]
where
<eq>(selectionMethod, "reinforcement_learning");
params' := <separate-by(|", ")>params*;
<set-global> (VarName, "RL")
```

Bu kural `selectionMethod` parametresinin değeri sadece pekiştirmeli öğrenmeyi belirten `reinforcement_learning`'e eşit olduğunda çalışmaktadır. Pekiştirmeli öğrenme yönteminin alması gereken bazı parametreler vardır. Bu parametreler `SelectionType` söz dizimi kuralının `params*` parametresinde tutulmaktadır. "*" işareti istenen sayıda parametrenin verilmesine imkan sağlar. Kod üretim kuralına gelen parametreler bir katar dizisinde tutulur. Bu katar dizisinin tek bir parametreye dönüştürülmesi gerekmektedir. Bunun için Stratego/XT dilinde tanımlanmış olan `<separate-by(|", ")>` kuralı kullanılabilir. `<separate-by(|", ")>` kuralı kendisine gelen katar dizisindeki ifadeleri "," ile ayırarak tek bir katar oluşturur. Bu katar `params'` değişkenine atanmıştır ve üretim kuralının içinde kullanılmaktadır.

Basit rastgele seçim ve pekiştirmeli öğrenmenin dışında açgözlü seçim ve rastgele permütasyon seçim yöntemleri de bulunmaktadır. Bu yöntemlerin kod üretme kuralları da benzer şekildedir. Bu yöntemler dışında bir seçim yöntemi kullanılmak istendiğinde kullanıcı tarafından önceden tanımlanmış olması gereklidir. Kullanıcı önceden tanımladığı yöntemi kullanmak istediğinde aşağıdaki kod üretme kuralı çalışmaktadır:

```
to-java:
Assignment(VarName,
    SelectionType(None(), selectionMethod, params*))-> $[
[selectionMethod] [VarName] =
    new [selectionMethod](problem);
]
where
<not("simple_random")> selectionMethod;
<not("random_permutation")> selectionMethod;
<not("reinforcement_learning")> selectionMethod;
<not("greedy")> selectionMethod;
<set-global> (VarName, "other")
```

Bu kural `selectionMethod` parametresi önceden tanımlanmış seçim yöntemlerinden hiçbirine eşit olmadığı durumda çalışmaktadır.

Şimdiye kadar anlatılan kod üretme kurallarında sezgisel tipi belirtilmemişti. Bir veya birden fazla sezgisel tipi arasından seçim yapmak için kullanılan kod üretme kuralı aşağıda görülmektedir:

```
to-java:
Assignment(VarName,
    SelectionType(
    Some(MultipleHeuristics(heuristics*)),
    selectionMethod, params*))-> $[
SimpleRandomSelection [VarName] =
    new SimpleRandomSelection(
    externalMethods.getHeuristicList("[heuristics']",
    problem),
    rng);
]
where
heuristics' := <separate-by(|", ")>heuristics*;
<eq>(selectionMethod, "simple_random");
```

```
<set-global> (VarName, "simple_random")
```

İsteğe bağlı olarak belirtilen sezgisel tipi kullanıldığında bu parametre `Some(...)` ifadesi içinde yer almaktadır. Sezgisel tipleri söz dizimi kuralında `MultipleHeuristics` etiketiyle tanımlanmıştır. `MultipleHeuristics` kuralında bir veya birden fazla sezgisel tipi "+" işareti ile ayrılabilir. Burada `heuristics*` parametresi bu sezgisel tiplerini bir katar dizisi olarak tutmaktadır. Sezgisel tipleri kod üretme kuralına gönderilmeden önce "," ile ayrılarak tek bir katara dönüştürülmesi gerekir.

5.3.4 Kabul yöntemi seçimi

Üretilen çözümlerin kabul edilmesi için bazı yöntemler tanımlanmıştır. Bu yöntemler şunlardır: Tüm çözümlerin kabul edilmesi, sadece iyileşen çözümlerin kabul edilmesi, iyileşen ve aynı kalan çözümlerin kabul edilmesi, iyileşen çözümlerin kabul edilip iyileşmeyen çözümlerin belli bir olasılıkla kabul edilmesi.

Aşağıdaki örnekte tüm çözümleri kabul eden bir kabul edici tanımlanmıştır:

```
ac = acceptance accept_all
```

Kabul yöntemi seçimi için kullanılan söz dizimi kuralı aşağıda görülmektedir. Bu kural `acceptance` anahtar sözcüğü, kabul yönteminin adı ve istenen sayıda parametreden oluşmaktadır.

```
"acceptance" AcceptanceSelectionType AcceptanceParams*
  -> Expression   {cons("AcceptanceType")}
```

Terminal olmayan `AcceptanceSelectionType` elemanı aşağıdaki kabul yöntemleri terminallerinden birine dönüştürülür:

```
"accept_all"                      -> AcceptanceSelectionType
"improving_or_equal"              -> AcceptanceSelectionType
"only_improving"                  -> AcceptanceSelectionType
"nonimproving_with_probability"-> AcceptanceSelectionType
```

Tanımlanmış kabul yöntemleri arasından parametre alan tek kabul yöntemi olasılıklı kabul yöntemi olduğu için terminal olmayan `AcceptanceParams` elemanı sadece bu yöntem tarafından kullanıldığında anlam kazanmaktadır. Bu yüzden bu parametrenin üretim kuralı sadece olasılık değerini almak için FLOAT türünden değer alacak şekilde oluşturulmaktadır. Bu kural aşağıda görülebilir:

```
FLOAT -> AcceptanceParams
```

Aşağıdaki örnekte üretilen tüm çözümlerin kabul edildiği kod üretilmektedir:

```
to-java:
Assignment(VarName,
    AcceptanceType(acceptanceType, params))-> $[
AcceptAll [VarName] = new AcceptAll();
]
where
<eq>(acceptanceType, "accept_all")
```

Bu kural sadece `acceptanceType` parametresi `accept_all`'a eşit olduğunda çalışmaktadır. Tüm çözümlerin kabul edildiği bu yöntemin parametreye ihtiyacı yoktur. Bu yüzden `params` parametresi kullanılmamıştır.

İyileşen çözümlerin kabul edilip iyileşmeyen çözümlerin belli bir olasılıkla kabul edildiği yöntemin kod üretme kuralı aşağıda görülebilir:

```
to-java:
Assignment(VarName,
    AcceptanceType(acceptanceType, probability))-> $[
NonImprovingWithProbabilityAcceptance [VarName] =
    new NonImprovingWithProbabilityAcceptance(
        rng, [probability]);
]
where
<eq>(acceptanceType, "nonimproving_with_probability")
```

Bu kural sadece `acceptanceType` parametresinin değeri iyileşen çözümlerin kabul edilip iyileşmeyen çözümlerin belli bir olasılıkla kabul edildiğini belirten `nonimproving_with_probability`'ye eşit olduğunda çalışmaktadır. Bu yöntemde olasılık değeri parametrik olarak alınmaktadır. Bu değer `probability` değişkeninde tutulmaktadır. `probability` değişkeni üretilen Java koduna parametrik olarak verilmektedir.

5.3.5 Başlatma

Başlatma aşamasında çözüm adayları ve amaç fonksiyonlarının değerlerinin tutulduğu bir dizi yaratılır, ilk değerleri atanır. Problemde yer alan çözüm adayı sayısı isteğe bağlı olarak verilebilir. Çözüm adayı sayısının verilmediği durumda sadece bir çözüm adayı oluşturulur. Örneğin aşağıdaki ifade 10 tane çözüm adayı üretecek ve rastgele ilk değer atama işlemini gerçekleştirecektir:

```
initialize solution 10
```

Başlatma aşamasında çözümlerin ilk değerlerinin atanması için belirlenmiş olan söz dizimi kuralı aşağıda görülmektedir:

```
"initialize" "solution" INT?
  -> Stm          {cons("Initialization")}
```

Bu kural `initialize solution` anahtar sözcüklerinden ve isteğe bağlı olarak belirtilen çözüm sayısından oluşmaktadır.

Çözüm adayı sayısının verilmediği durumda çalışan kod üretme kuralı aşağıda görülmektedir:

```
to-java:
Initialization(None()) -> $[
int solutionmemorysize = 1;
problem.setMemorySize(solutionmemorysize+1);
double["[]"] curr_obj_function =
    new double["["]solutionmemorysize+1["]"];
```

```
for (int x = 0; x < solutionmemorysize+1; x++) {
problem.initialiseSolution(x);
curr_obj_function["[x]"] =
      problem.getFunctionValue(x);
}
ApplyHeuristic heuristicApplier = new ApplyHeuristic();
]
```

`Initialization` söz dizimi kuralı bir tane terminal sembol içermektedir ve bu sembol isteğe bağlı olarak verilebilir. Bu kuralda bu sembol verilmediği için parametre `None()` olmaktadır. Buna göre sadece bir tane çözüm adayı ve bir eleman içerecek bir amaç fonksiyonu dizisi tanımlanır ve ilk değer atamaları rastgele yapılır. Üretilen Java kodunda seçilen sezgiselleri uygulayan `ApplyHeuristic` nesnesi yer almaktadır. Bu nesnenin bir örneği yaratılır ve çalışmaya hazır hale getirilir.

Çözüm adayı sayısının verildiği durumda çalışan kod üretme kuralı aşağıda görülmektedir:

```
to-java:
Initialization(Some(MemorySize)) -> $[
int solutionmemorysize = [MemorySize];
problem.setMemorySize(solutionmemorysize+1);
double["[]"] curr_obj_function =
    new double["["]solutionmemorysize+1["]"];
for (int x = 0; x < solutionmemorysize+1; x++) {
problem.initialiseSolution(x);
curr_obj_function["[x]"] =
    problem.getFunctionValue(x);
}
ApplyHeuristic heuristicApplier = new ApplyHeuristic();
]
```

Bu kuralda çözüm adayı sayısı verildiği için bu parametre `Some(MemorySize)` ifadesi içindeki `MemorySize` parametresinde tutulmaktadır. Bu sayı üretilen Java

koduna parametre olarak verilmektedir. Buna göre verilen sayı kadar çözüm adayı yaratılmakta ve ilk değerleri atanmaktadır.

5.3.6 Sezgisellerin seçilmesi

Sezgisel seçim yöntemleri tanımlandıktan sonra bu yöntemler kullanılarak sezgiseller seçilir.

Sezgisel seçimi için kullanılan söz dizimi kuralı aşağıda görülmektedir:

```
"next" ID
  -> Expression   {cons("NextHeuristicSelection")}
```

Bu kural `next` anahtar sözcüğünden ve tanımlanan seçim yöntemine verilen isimden oluşur.

Sezgisellerin seçimi için kullanılan kod üretme kuralı aşağıda görülmektedir:

```
to-java:
Assignment(heuristicName,
    NextHeuristicSelection(heuristicToSelect))-> $[
Heuristic [heuristicName] = [heuristicToSelect].next();
]
```

`NextHeuristicSelection` söz dizimi kuralı bir tane terminal sembol içermektedir. Bu sembol sezgisel seçme yönteminin adını ifade etmektedir ve kod üretme kuralında `heuristicToSelect` parametresinde tutulmaktadır. `heuristicToSelect` parametresinde tutulan değer Java'da bir sezgisel seçen nesneye denk gelmektedir. Tüm sezgisel seçim sınıfları `next` metoduna sahiptir. Burada parametre olarak gelen sezgisel seçim yöntemine ait `next` metodu çağrılmakta ve geriye bir `Heuristic` nesnesi dönmektedir. Geriye dönen nesne `heuristicName` ismindeki bir `Heuristic` nesnesine atanmaktadır.

5.3.7 Sezgisellerin uygulanması

Bir sezgisel bir çözüme `apply` ifadesiyle uygulanır. Sezgisel bir çözüme uygulanabildiği gibi bir çözüm kümesine de uygulanabilir. Örneğin, aşağıdaki kod `h` sezgiselini `s` çözümüne uygular ve üretilen çözümü yeni bir değişkene atar (`n`):

```
n = apply h s
```

Sezgisellerin çözümlere uygulanması için kullanılan söz dizimi kuralı `apply` anahtar sözcüğü, sezgisel ve çözüm bilgilerini içeren terminal olmayan `Arguments` elemanından oluşmaktadır. Bu kural aşağıda görülebilir:

```
ID "=" Expression
  -> Stm          {cons("Assignment")}

"apply" Arguments
  -> Expression   {cons("ApplyHeuristic")}
```

`Arguments` elemanı aşağıda gösterilen elemanlara dönüştürülür. İlk eleman seçilen sezgiseli, ikinci eleman ise seçilen çözümleri belirtmektedir.

```
ID ID                 -> Arguments{cons("ApplyArg")}
ID                    -> Arguments{cons("ApplyArg")}
ID Arrays             -> Arguments{cons("ApplyArrayArg")}
ArrayWithIndex ID     -> Arguments{cons("ApplyArrayArg")}

Array                 -> Arrays {cons("Arrays")}
ArrayWithIndex        -> Arrays {cons("Arrays")}

ID "[]"               -> Array  {cons("Array")}
ID "[" ParamType "]"  -> ArrayWithIndex
                              {cons("ArrayWithIndex")}
```

Seçilen sezgiseli ifade etmek için sezgiselin adı veya sezgisel dizisindeki yeri belirtilebilir. Seçilen çözümleri ifade etmek için ise çözümün adı veya çözüm dizisi belirtilebilir. Tek çözümün kullanıldığı durumlarda sadece seçilen sezgiselin adı verilerek bu çözüme uygulanabilir.

Bir sezgisel bir çözüme veya çözüm kümesine uygulanabilir. Bir sezgisel bir çözüme uygulandığında çalışan kod üretme kuralı aşağıda görülmtektedir:

```
to-java:
Assignment(newSolution,
    ApplyHeuristic(
    ApplyArg(heuristicName, existingSolution))) -> $[
int [newSolution] = solutionmemorysize;
curr_obj_function["["][newSolution]["]"] =
    heuristicApplier.apply(
    problem, [heuristicName],
[existingSolution], [newSolution]);
delta =
    curr_obj_function["["][existingSolution]["]"] -
    curr_obj_function["["][newSolution]["]";
[heuristicType].updateDelta([heuristicName], delta);
]
where
heuristicType := <get-global> heuristicName;
heuristicTypeName := <get-global> heuristicType;
<eq>(heuristicTypeName, "RL")
```

Bu üretim kuralı seçilen sezgiselin tipi sadece pekiştirmeli öğrenme olduğunda çalışmaktadır. Ancak üretim kuralına gelen parametreler arasında sezgiselin tipi yoktur, sadece adı bulunmaktadır. Sezgisel seçme yöntemlerinin tanımlanması esnasında yöntemin adı ile yöntemin türü arasında bir ilişki kurularak global değişkenlerde tutulmaktaydı. Burada da `<get-global>` kuralına sezgiselin adı verilerek seçim yöntemi bulunmaktadır. Eğer seçim yöntemi pekiştirmeli öğrenmeyi temsil eden "RL" ifadesine eşitse bu kural çalışmaktadır.

`ApplyHeuristic` söz dizimi kuralı terminal olmayan `Arguments` sembolünü içermektedir. Bu sembol ise iki tane terminale dönüştürülebilen `ApplyArg` sembolüne dönüştürülmektedir. Bu terminal simgelerden `heuristicName` seçilen sezgiseli, `existingSolution` ise sezgiselin uygulanacağı çözüm adayını belirtmektedir. Üretilen kod bloğunda seçilen sezgisel çözüm adayına uygulanır ve yeni bir amaç fonksiyon değeri elde edilir. Bu değer amaç fonksiyon değerlerinin tutulduğu diziye atanır ve çözüm adayındaki iyileşme `delta` isimli bir değişkende

tutulur. Pekiştirmeli öğrenme yönteminde sezgisel uygulandıktan sonra sezgiselin delta değeri güncellenmektedir.

Eğer seçilen sezgiselin tipi pekiştirmeli öğrenme değilse sezgiselin delta değeri güncellenmemektedir. Bunun için çalışan kod üretme kuralı aşağıda görülmektedir:

```
to-java:
Assignment(newSolution,
    ApplyHeuristic(
    ApplyArg(heuristicName, existingSolution))) -> $[
int [newSolution] = solutionmemorysize;
curr_obj_function["["][newSolution]["]"] =
    heuristicApplier.apply(
    problem, [heuristicName],
    [existingSolution], [newSolution]);
delta =
    curr_obj_function["["][existingSolution]["]"] -
    curr_obj_function["["][newSolution]["]"];
]
where
heuristicType := <get-global> heuristicName;
heuristicTypeName := <get-global> heuristicType;
<not("RL")> heuristicTypeName
```

Sadece bir tane çözüm adayının olduğu programlarda seçilen sezgiselin hangi çözüm adayına uygulanacağını söylemeye gerek yoktur. Bu durum için çalışan kod üretme kuralı aşağıda görülmektedir:

```
to-java:
Assignment(newSolution,
    ApplyHeuristic(ApplyArg(heuristicName))) -> $[
int [newSolution] = solutionmemorysize;
curr_obj_function["["]0["]"] =
    heuristicApplier.apply(
```

```
    problem, [heuristicName],
    0, solutionmemorysize);
delta =
    curr_obj_function["["]solutionmemorysize["]"] -
    curr_obj_function["["]0["]"];
]
```

Bu durumda `ApplyArg` sembolü sadece uygulanacak olan sezgiseli parametre almaktadır. Üretilen kod bloğunda seçilen sezgisel programdaki tek çözüm adayına uygulanır ve yeni bir amaç fonksiyon değeri elde edilir. Bu değer amaç fonksiyon değerlerinin tutulduğu diziye atanır. Bu kural sezgisel seçim yönteminin pekiştirmeli öğrenme olmadığı durumda çalışmaktadır. Pekiştirmeli öğrenme yöntemi kullanılıyor olsaydı yukarıda bahsettiğimiz gibi delta değerinin güncellenmesi gerekecekti.

Seçilen sezgiselin bir çözüm kümesine uygulandığı durumlar da vardır. Bu duruma sadece çaprazlama türünden bir sezgisel seçilmişse ihtiyaç duyulur. Aşağıdaki kod üretme kuralı bu ihtiyacı karşılamaktadır:

```
to-java:
Assignment(newSolution,
    ApplyHeuristic(
        ApplyArrayArg(heuristicName,
            Arrays(Array(array)))))  -> $[
int [newSolution] = solutionmemorysize;
curr_obj_function["["][newSolution]["]"] =
    heuristicApplier.apply(problem,[heuristicName],
    [array].get([0]), [array].get([1]), [newSolution]);
delta =
    curr_obj_function["["][array].get([0])["]"] -
    curr_obj_function["["][newSolution]["]"];
]
```

Burada `ApplyHeuristic` söz dizimi kuralındaki `Arguments` sembolü `ApplyArrayArg` sembolüne dönüştürülmektedir. `ApplyArrayArg` sembolü

ise seçilen sezgiselin adını tutan bir terminale ve çözüm adaylarının dizisini tutan bir terminale dönüştürülmektedir. Sezgiselin adı `heuristicName` parametresinde tutulurken, çözüm adaylarının dizisi ise `array` parametresinde tutulmaktadır. Üretilen Java kodunda seçilen sezgisel çözüm adayı dizisinin 0 ve 1 numaralı elemanlarına uygulanmaktadır.

5.3.8 Çözümlerin kabul edilmesi

Üretilen bir çözüm, kabul edicinin kontrollerinden sonra kabul edilebilir veya reddedilebilir. Bu işlem eski ve yeni çözümleri parametre olarak alır ve yeni çözümü kabul edip etmemeye karar verir. Eğer yeni çözüm kabul edilirse bu çözüme hangi işlemin uygulanacağı da belirtilir (eski çözümü yenisiyle değiştirmek gibi). Aşağıda verilen örnekte eğer kabul koşulları yeterliyse kabul edici (`ac`) eski çözümü (`s`) yeni çözüm (`n`) ile değiştirecektir:

```
check ac n replace s
```

Çözümlerin kabul edilmesi için kullanılan söz dizimi kuralı aşağıda görülmektedir:

```
"check" ID ID "replace" ID ("[" INT "]")?
  -> Stm          {cons("Acceptance")}
```

`Acceptance` söz dizimi kuralında sırasıyla `check` anahtar sözcüğü, kabul yönteminin adı, yeni çözüm adayının adı, `replace` anahtar sözcüğü, eski çözüm adayının adı ve isteğe bağlı olarak köşeli parantezler içerisinde bir indis yer alır. Bir çözüm dizisi kullanıldığında elemanın dizideki yerini belirtmek amacıyla bu indis kullanılmaktadır.

Sezgisel bir çözüme uygulandıktan sonra yeni bir çözüm elde edilmektedir. Üretilen yeni çözümün kabul edilip edilmeyeceğini kontrol etmek için çalışan kod üretme kuralı aşağıda görülmektedir:

```
to-java:
Acceptance(accepter, newSolutionInd,
    exSolutionInd, None()) -> $[
```

```
if([accepter].checkIfAcceptance(
    curr_obj_function["["][exSolutionInd]["]"],
    curr_obj_function["["][newSolutionInd]["]"])) {
problem.copySolution([newSolutionInd],
    [exSolutionInd]);
curr_obj_function["["][exSolutionInd]["]"] =
    curr_obj_function["["][newSolutionInd]["]"];
}
]
```

`Acceptance` kod üretme kuralında kabul yönteminin adı `accepter` parametresinde tutulmaktadır. Eski ve yeni çözüm adaylarının indisleri de sırasıyla `exSolutionInd` ve `newSolutionInd` parametrelerinde tutulmaktadır. Bir çözüm dizisiyle çalışıldığı durumlarda `Acceptance` kod üretme kuralı bir parametre daha almaktadır. Bu kuralda dördüncü parametre olmadığı için onun yerine `None()` ifadesi yazılmıştır. `accepter` parametresinde tutulan değer Java'da bir nesneye karşılık gelmektedir. Üretilen Java kodunda bu nesnenin `checkIfAcceptance` metodu çağrılarak çözümlerin kabul edilip edilmeyeceği sorgulanmaktadır. Eğer çözümler kabul edilirse eski çözüm adayının yerini almaktadır.

Bir çözüm dizisiyle çalışıldığı durumlarda çözümlerin kabul edilmesi için çalışan kod üretme kuralı aşağıda görülmektedir:

```
to-java:
Acceptance(accepter, newInd,
    exArray, Some(Index)) -> $[
if([accepter].checkIfAcceptance(
    curr_obj_function["["][exArray].get([Index])["]"],
    curr_obj_function["["][newInd]["]"])) {
problem.copySolution([newInd], [exArray].get([Index]));
curr_obj_function["["][exArray].get([Index])["]"] =
    curr_obj_function["["][newInd]["]"];
}
]
```

Çaprazlama türünden sezgiseller seçildiğinde bir çözüm dizisiyle çalışılmaktadır. Bu kod üretme kuralı da sadece çaprazlama türünden sezgisller seçildiğinde kullanılmaktadır. `Acceptance` kod üretme kuralında kabul yönteminin adı yine `accepter` parametresinde tutulmaktadır. Eski çözüm adaylarının yer aldığı dizinin adı `exArray` parametresinde, kontrol edilecek çözüm adayının bu dizideki indisi `Index` parametresinde, yeni çözüm adayının indisi ise `newInd` parametresinde tutulmaktadır. Yeni çözüm adayının indisi isteğe bağlı bir parametredir. Bir önceki kuralda bu parametre kullanılmadığı için yerinde `None()` ifadesi yer alırken, burada kullanıldığı için `Some(Index)` olarak belirtilmiştir. Üretilen Java kodunda `accepter` parametresinde tutulan nesnenin `checkIfAcceptance` metodu çağrılmaktadır. Çözüm dizisindeki `Index` indisinde tutulan eleman ile yeni çözüm adayının indisi bu metoda parametre olarak gönderilmektedir. Yeni çözümün kabul edilmesi durumunda çözüm dizisindeki elemanın yerini yeni çözüm adayı almaktadır.

5.3.9 Yapı tanımlanması

Dili kullanan kişiler sezgisel seçmek için kendi yöntemlerini tanımlamak isteyebilirler. Bu durumda yapılardan yararlanabilirler. Yapılar içerisinde birden fazla ifade ve metot yer alabilir. Aşağıda verilen örnekte sezgisel seçimi için kullanılmak üzere (`myHeuristic`) isimli bir yapı tanımlanmıştır. `permute` komutuyla var olan sezgisellerin bir permütasyonu oluşturulmuş ve bir diziye atanmıştır. `next` komutu permütasyon dizisinden bir sezgiseli seçmek ve döndürmek için kullanılmıştır.

```
define myHeuristic
{
    h[] = permute heuristic
    current = -1

    next() : Heuristic
    {
        current = current + 1
        return h[current]
    }
```

```
}
```

Önceden tanımlı sezgisel yöntemleri kullanılırken de bu yapı ile tanımlanmış sezgisel yöntemleri kullanılırken de aynı metot kullanılır. Aşağıdaki örnekte görüldüğü gibi `selection` anahtar sözcüğünden sonra tanımlanmış yapının ismi yazılır ve bu şekilde çağrım yapılır:

```
hs = mutation selection myHeuristic
h = next hs
```

Kabul yöntemi tanımlamak için de yapılardan yararlanılır. Aşağıda verilen örnekte bir kabul yöntemi tanımlanmıştır. `myAcceptance` ismindeki kabul yönteminin `check` metodu iki çözümü parametre olarak almakta ve bunları karşılaştırmaktadır. Yeni çözümün kabul edilip edilmeyeceği bilgisini dönmektedir.

```
define myAcceptance
{
    check(exSolution, newSolution) : Boolean
    {
        return better newSolution exSolution
    }
}
```

Yapı tanımlamak için kullanılan söz dizimi kuralı `define` anahtar sözcüğü, yapının adı, varsa parametreler ve süslü parantezler içerisinde yer alan ifadeler ve metotlardan oluşmaktadır. İstenen sayıda ifade ve metot burada tanımlanabilir. Söz dizimi kuralları aşağıda görülmektedir:

```
"define" ID "{" Stm* Methods* "}"
  -> StructDcl   {cons("StructDeclaration")}
"define" ID "(" StructParams ")" "{" Stm* Methods* "}"
  -> StructDcl   {cons("StructDclWithParams")}
```

Parametre almayan bir yapı tanımlamak için `StructDeclaration` söz dizimi kuralı kullanılmaktadır. Bu kural ile yapının adı, yapı içerisinde yer alan ifadeler ve

metotlar tanımlanabilir. Yapının içinde tanımlanan ifadeler dilin izin verdiği tüm `Stm` elemanları olabilir. `Methods` elemanı için tanımlanan söz dizimi kuralı ise aşağıda görülmektedir:

```
ID"(" MethodParams? ") : " VariableType? "{" Stm* "}"
  -> Methods      {cons("Methods")}
```

Termianl olmayan `Methods` elemanı metodun adını ifade eden bir terminale, parantez içerisinde yer alan isteğe bağlı `MethodParams` elemanına, onu izleyen ":" işaretine, isteğe bağlı olarak belirtilen `VariableType` elemanına ve süslü parantezler içerisinde yer alan ifadelere dönüştürülür.

`MethodParams` elemanı metotların aldığı parametreleri belirlemektedir. Parametreler virgüllerle ayrılmış katarlardan oluşmaktadır:

```
{ID ","}+
  -> MethodParams {cons("MethodParams")}
```

`VariableType` elemanı metotların geri dönüş değerlerini belirtmektedir ve aşağıdaki elemanlara dönüştürülebilir:

```
"Heuristic"   -> VariableType
"Integer"     -> VariableType
"String"      -> VariableType
"Double"      -> VariableType
"Boolean"     -> VariableType
```

Parametre alan bir yapı tanımlamak için `StructDclWithParams` söz dizimi kuralı kullanılmaktadır. Bu kural ile yapının adı, aldığı parametreler, yapı içerisinde yer alan ifadeler ve metotlar tanımlanabilir. Yapının aldığı parametreler için kullanılan söz dizimi kuralı aşağıda görülmektedir:

```
{Arg ","}+
  -> StructParams {cons("StructParams")}
```

`StructDclWithParams` kuralıyla tanımlanan yapılarda en az bir parametre olmalı ve tüm parametreler virgüllerle ayrılmalıdır. Parametrelerin yapısı ise aşağıda görülebilir:

```
ID ":" VariableType
  -> Arg          {cons("Arg")}
```

Bu kurala göre parametreler bir isimden ve bir tipten oluşmaktadır.

Tanımlanan yapılar Java tarafındaki sınıflara karşılık gelmektedir. Her yapı için ayrı bir sınıf tanımlanmaktadır. Parametre almayan bir yapı tanımlandığında çalışan kod üretme kuralı aşağıda görülmektedir:

```
to-java:
StructDeclaration(className, stm*, methods*)-> $[
public class [className]{
[stm'*]
ProblemDomain problem;
int solutionmemorysize;
ApplyHeuristic heuristicApplier;
double delta;
double["["]["]"] curr_obj_function;
public [className](ProblemDomain problem)
{
  this.problem = problem;
  heuristicApplier = new ApplyHeuristic();
  delta = 0;
}
public void setMemorySize(int size)
{
  solutionmemorysize = size;
  curr_obj_function = new double["["]size+1["]"];
}
[methods'*]
```

```
}
]
with
stm'* := <to-java>stm*;
methods'* := <to-java>methods*
```

Bu kural yapının adını, ifadeleri ve metotları parametre olarak almaktadır. Üretilen kodda yapının adıyla aynı ada sahip bir sınıf tanımlanmaktadır. Parametre olarak gelen ifadelere ve metotlara `<to-java>` komutu uygulanarak karşılık gelen Java kodları oluşturulmakta ve tanımlanan sınıf içine yerleştirilmektedir. HyFlex tarafından ihtiyaç duyulan bazı temel tanımlamalar, sınıfın kurucu metodu ve problemin çözüm adayı sayısını belirlemek için kullanılan `setMemorySize` metodu da sınıf içerisinde tanımlanmaktadır.

Parametre alan bir yapı tanımlandığında çalışan kod üretme kuralı aşağıda görülmektedir:

```
to-java:
StructDclWithParams(className,
    StructParams(params*), stm*, methods*) -> $[
public class [className] {
[params'*];
[stm'*]
ProblemDomain problem;
int solutionmemorysize;
ApplyHeuristic heuristicApplier;
double delta;
double["["]["]"] curr_obj_function;
public [className]([paramsNames*])
{
[constructorBody]
this.problem = problem;
  heuristicApplier = new ApplyHeuristic();
  delta = 0;
```

```
}
public void setMemorySize(int size)
{
  solutionmemorysize = size;
  curr_obj_function = new double["["]size+1["]"];
}
[methods'*]
}
]
with
stm'* := <to-java>stm*;
methods'* := <to-java>methods*;
constructorParams'* := <to-java>params*;
params'* := <separate-by(|"; ")>constructorParams'*;
paramsNames* := <separate-by(|", ")>constructorParams'*;
constructorBody :=
   <map(concatConstructorBody)> constructorParams'*

to-java:
Arg(VarName, VarType)-> $[
[VarType] [VarName]
]
```

Bu kural yapının adını, parametreleri, ifadeleri ve metotları parametre olarak almaktadır. Yapıya parametre olarak verilen ifadelere `<to-java>` komutu uygulanarak aşağıdaki kod üretme kuralı çalışmaktadır. Parametreler `ID ":" VariableType` yapısında oldukları için önce parametrenin adı sonra da tipi gelmektedir. Bu kural çalıştırılarak önce veri tipinin daha sonra da parametrenin yer aldığı kodlar üretilmektedir ve bu kodlar `constructorParams'*` ara değişkeninde tutulmaktadır. Elde edilen kodlara `<separate-by(|"; ")>` komutu uygulanarak tüm parametrelerin noktalı virgül ile ayrılması sağlanmaktadır. Böylece parametrelerin Java dilindeki tanımları yapılmaktadır. `constructorParams'*` ara değişkeninde tutulan kodlara `<separate-by(|", ")>` komutu uygulanarak tüm parametrelerin

virgül ile ayrılması sağlanmakta ve bu kod kurucu metodun parametrelerini oluşturmaktadır. Kurucu metodun parametreleri `paramsNames*` ara değişkeninde tutulmaktadır. Daha sonra `constructorParams'*` ara değişkeninde tutulan tüm kodlara `concatConstructorBody` komutu uygulanmakta ve kurucu metodun içerisinde yer alacak olan parametre değer atamalarına ait kod üretilmektedir. `<map>` komutu bir komutu bir dizinin tüm elemanlarına uygulamak için kullanılmaktadır.

`concatConstructorBody` komutu yapıcı metodun gövdesini oluşturmak amacıyla tanımlanmış bir komuttur. Tanımı aşağıda görülmektedir:

```
concatConstructorBody =
  list := <string-tokenize(|[' '])> <id>;
  element := <last> list;
  <conc-strings>("this.", element, "=", element,";
")
```

Bu komuta parametre olarak bir katar gelmekte ve bu katarda parametrenin tipi, bir boşluk ve parametrenin adı tutulmaktadır. Gelen parametre `<id>` değişkeninde tutulmaktadır. Stratego/XT dilinde tanımlı `<string-tokenize(|[' '])>` komutu kullanılarak katar boşluk karakterinin olduğu noktadan iki parçaya bölünmekte ve `list` isimli bir diziye atanmaktadır. Bu dizinin ilk elemanı parametrenin tipini, ikinci elemanı ise parametrenin adını tutmaktadır. Stratego/XT dilinin `<last>` komutu kullanılarak dizinin son elemanı yani parametrenin adı alınarak `element` değişkeninde tutulmaktadır. Daha sonra `<conc-strings>` komutuyla katar birleştirme işlemi yapılarak parametrelere değerleri atanmaktadır.

Yapılar içerisinde tanımlanan metotlar için çalışan kod üretme kuralı aşağıda görülmektedir:

```
to-java:
Methods(MethodName, None(), Some(VariableType), Stm*) ->
$[
[VariableType] [MethodName]()
{
  [Stm'*]
```

```
}
]
with
Stm'* := <to-java>Stm*
```

Bu kural metodun adını, geri dönüş tipini ve metot içerisinde çalıştırılacak olan ifadeleri parametre olarak almaktadır.

6. KULLANIM VE ÖRNEKLER

6.1 Kullanım

Bu bölümde verilmiş olan kodlar HyFlex platformunda yer alan örneklerden yararlanılarak hazırlanmıştır. Bu örnekler geliştirilen sistem üzerinde test edilmiş ve üretilen kodun HyFlex'te yer alan örneklerin koduna eşdeğer olup olmadığı kontrol edilmiştir. Tasarlanan dil aynı zamanda HyFlex'te yer almayan başka üst sezgisel algoritmalar gerçeklenerek de test edilmiştir.

Tasarlanan dil modern geliştirme ortamlarından biri olan Eclipse'e entegre olan Spoofax dil geliştirme platformu kullanılarak gerçeklenmiştir. Bu yüzden çalışma ortamı Eclipse ve Spoofax tarafından sağlanan kod renklendirme, kod katlama gibi avantajlardan faydalanabilmektedir. Çalışma ortamının editöründe kod yazıldıktan sonra menüden kod üretimi çalıştırılır. Kod üretme işlemi ile birlikte Java kodu üretilir ve otomatik olarak bir Eclipse Java projesi oluşturularak üretilen kodlar bu projenin altına eklenir.

6.2 Örnekler

Şekil 6.1 verilen zaman diliminde sadece bir çözümle çalışan basit bir üst sezgisel algoritmayı göstermektedir. Sezgisel seçim yöntemleri problemin desteklediği tüm türler arasından rastgele seçilecektir. Üretilen çözüm orijinal çözümden daha iyiyse kabul edilir, daha kötüyse de %50 olasılıkla kabul edilecektir. Her iterasyonda rastgele bir sezgisel seçilip yeni bir çözüm elde etmek amacıyla var olan çözüme uygulanır. Üretilen yeni çözümün kabul şartlarını sağlayıp sağlamadığı kontrol edilir ve eğer kabul edilirse orijinal çözümle değiştirilir. Bu iterasyon programa ayrılmış tüm süre boyunca çalışır.

İkinci örnek program Şekil 6.2'de görülmektedir ve 10 çözüm adayı ile çalışmaktadır. Algoritmanın farklı adımlarında üç farklı sezgisel seçim yöntemi kullanılmıştır. Üretilen tüm çözümler iyileşme olup olmamasına bakılmadan kabul edilmektedir. Her iterasyonda mutasyon türünden sezgisellerin arasından rastgele bir seçim yapılır ve

```
hs = selection simple_random
ac = acceptance nonimproving_with_probability 0.5
initialize solution
loop 100% {
    h1 = next hs
    s = first solution
    n1 = apply h1 s
    check ac n1 replace s
}
```

Şekil 6.1: Basit rastgele seçim ve iyileşmeyen çözümlerin olasılıksal kabulünün yapıldığı, bir çözüm adayının olduğu örnek program.

seçilen sezgisel eldeki çözüm kümesinde yer alan en iyi çözüm adayına uygulanır. Üretilen çözüm adayı kabul edilir ve var olan en iyi çözümün yerini alır. Daha sonra yerel arama türünden sezgiseller arasından rastgele bir seçim yapılır ve en iyi çözüme uygulanır. Üretilen yeni çözüm de en iyi çözümle değiştirilir. Daha sonra 2 farklı çözüm adayı rastgele seçilir ve rastgele seçilen çaprazlama sezgisel yöntemi bu çözümlere uygulanır. Çaprazlama sezgisel yöntemi iki çözüme ihtiyaç duyduğu için burada iki çözüm seçilmiştir. Üretilen yeni çözüm de kabul edilir ve çözüm adaylarının ilkiyle değiştirilir. Her iterasyonun sonunda en iyi çözüm adayı seçilir.

Şekil 6.3 seçilen sezgiselin türüne göre farklı dallara ayrılmanın bir örneğini göstermektedir. İterasyon başlamadan önce yerel arama türünden sezgiseller arasından bir seçim yapılır. Daha sonra iterasyonun her adımında mutasyon ve yerel arama türünden sezgiseller arasından rastgele bir seçim yapılır ve çözüm adayına uygulanır. Seçilen sezgiselin türü kontrol edilir. Seçilen sezgisel mutasyon türünden bir sezgisel ise önceden seçilmiş olan yerel arama sezgiseli çözüm adayına uygulanır.

Şekil 6.4'te bir sezgisel yöntemi tanımlamak ve kullanmak için bir yapının nasıl tanımlandığını gösteren bir örnek yer almaktadır. Bunun için
`myHeuristic` isimli bir yapı tanımlanır. Problemin desteklediği tüm sezgisellerin bir permütasyonu oluşturulur. Permütasyon dizisinin sıradaki elemanını seçmek için `next` metodu tanımlanmıştır. Tanımlanan bu yapı bir sezgisel seçme yöntemi olarak kullanılır. Bu örnekte üretilen tüm çözümler kabul edilmektedir.

Şekil 6.5'te bir kabul yöntemi tanımlamak ve kullanmak için kullanılacak bir yapının nasıl tanımlandığını gösteren bir örnek yer almaktadır. Bunun için `myAcceptance` isimli bir yapı tanımlanır. Tanımlanan `check` metodu iki çözümü parametre olarak

```
hs1 = mutation selection simple_random
hs2 = local_search selection simple_random
hs3 = crossover selection simple_random
ac = acceptance accept_all
initialize solution 10
loop 100% {
    h1 = next hs1
    s = best solution
    n1 = apply h1 s
    check ac n1 replace s
    h2 = next hs2
    n2 = apply h2 s
    check ac n2 replace s
    s3[] = random solution 2!
    h3 = next hs3
    n3 = apply h3 s3[]
    check ac n3 replace s3[0]
    currentBest = best solution
}
```

Şekil 6.2: Üç sezgisel seçim yönteminin olduğu, üretilen tüm çözümlerin kabul edildiği, 10 çözüm adayı olan örnek program.

```
hs1 = mutation+local_search selection simple_random
hs2 = local_search_heuristic selection simple_random
ac = acceptance accept_all
h2 = next hs2
initialize solution
loop 100% {
    h1 = next hs1
    s = first solution
    n = apply h1 s
    if h1 in mutation {
        n = apply h2 n
    }
    check ac n replace s
}
```

Şekil 6.3: Sezgisel türünün kontrol edildiği örnek program.

```
hs1 = selection myHeuristic
ac = acceptance accept_all
initialize solution 10
loop 100%]{
    h1  = next hs1
    s = first solution
    n = apply h1 s
    check ac n replace s
}

define myHeuristic
{
    h[] = permute heuristic
    current = -1

    next() : Heuristic
    {
        current = current + 1
        return h[current]
    }
}
```

Şekil 6.4: Sezgisel seçim yönteminin tanımlandığı örnek program.

```
p1 = selection simple_random
accepter = acceptance myAcceptance
initialize solution 10
loop 100%{
    h = next p1
    s = first solution
    n = apply h s
    check accepter n replace s
}

define myAcceptance
{
    check(exSol, newSol) : Boolean
    {
        return better newSol exSol
    }
}
```

Şekil 6.5: Kabul yönteminin tanımlandığı örnek program.

almakta, bu çözümleri karşılaştırarak yeni çözümün kabul edilip edilmeyeceği bilgisini dönmektedir.

7. SONUÇ VE ÖNERİLER

Çalışmada HyFlex platformu üzerinde çalışacak üst sezgisel algoritmalarını yazmaya imkan sunan alana özgü bir dil tasarlanmıştır. Bu dili geliştirmeden önce alana özgü diller ve üst sezgiseller hakkında kapsamlı bir araştırma yapılmıştır. Üst sezgisel algoritmaları önce Java dilinde yazılmış ve kullanıcıların karşılaşabileceği zorluklar öngörülmeye çalışılmıştır.

Bu dilde sağlanan özellikler HyFlex platformunda bir üst sezgisel programı geliştirmek için tasarlanmıştır. HyFlex ile çalışmanın avantajı HyFlex'in içinde problemlerin yer almasıdır. Bu açıdan bakıldığında araştırmacılar problemleri bir programlama dilinde yazmak yerine problemleri çözecek üst sezgisel algoritmalarını geliştirmeye odaklanabilirler. Tasarlanan dil ileride üst sezgiseller alanında daha genel bir kullanıma imkan sağlayacak şekilde genişletilebilir.

Dilde sezgisel seçmek ve çözümleri değerlendirmek için özel yapılar tanımlanabilmektedir. Bu yapılar tanımlandıkları program içinde kullanılabilir durumdadır. Tanımlanan yapıların paketlenerek başka programlar tarafından da kullanılmasına imkan sağlamak ilerleyen zamanlarda dile eklenebilecek özellikler arasında sayılabilir.

Geliştirilen dilin kullanımı çeşitli örneklerle açıklanmıştır. Geliştirilen dile ait kaynak kodlarına, bu dilde yazılmış çeşitli programlara ve bu programların karşılığı olan Java kodlarına `https://bitbucket.org/hcora/hh-dsl` adresinden erişilebilir.

KAYNAKLAR

[1] **Burke, E.K.**, **Hyde, M.**, **Kendall, G.**, **Ochoa, G.**, **Özcan, E. ve Rong, Q.** (2009). A Survey of Hyper-heuristics, **Teknik Rapor**.

[2] **Burke, E.K.**, **Hyde, M.**, **Kendall, G.**, **Ochoa, G.**, **Özcan, E. ve Woodward, J.R.**, (2010). A Classification of Hyper-heuristic Approaches, **M. Gendreau ve J.Y. Potvin**, (düzenleyenler), Handbook of Metaheuristics, cilt146 of *International Series in Operations Research and Management Science*, Springer, s.449–468.

[3] **Chakhlevitch, K. ve Cowling, P.**, (2008). Hyperheuristics: Recent Developments, cilt136 of *Studies in Computational Intelligence*, Springer Berlin Heidelberg, s.3–29.

[4] **Cowling, P.**, **Kendall, G. ve Soubeiga, E.** (August 2000). A hyper-heuristic approach to scheduling a sales summit, *Practice and Theory of Automated Timetabling III : Third International Conference, PATAT 2000*, ciltLNCS 2079.

[5] **Ochoa, G.**, **Hyde, M.**, **Curtois, T.**, **Vazquez-Rodriguez, J.**, **Walker, J.**, **Gendreau, M.**, **Kendall, G.**, **McCollum, B.**, **Parkes, A.**, **Petrovic, S. ve Burke, E.** (2012). HyFlex: A Benchmark Framework for Cross-domain Heuristic Search, ***7245***.

[6] **Graham, H.**, **Gabriela, K.**, **Sanja, O. ve Antonio, P.J.**, MISTA 2009 HyFlex: A Flexible Framework for the Design and Analysis of Hyper-heuristics.

[7] **Hoare, C.A.R.** (1973). Hints on programming language design., **Teknik Rapor**, Stanford, CA, USA.

[8] **Kosar, T.**, **Mernik, M. ve Carver, J.C.** (2012). Program comprehension of domain-specific and general-purpose languages: comparison using a family of experiments, *Empirical Softw. Engg.*, ***17***(3), 276–304.

[9] **Kos, T.**, **Kosar, T. ve Mernik, M.** (2012). Development of data acquisition systems by using a domain-specific modeling language, *Comput. Ind.*, ***63***(3), 181–192.

[10] **Fister, Jr., I.**, **Fister, I.**, **Mernik, M. ve Brest, J.** (2011). Design and implementation of domain-specific language easytime, *Comput. Lang. Syst. Struct.*, ***37***(4), 151–167.

[11] **Mernik, M.**, **Heering, J. ve Sloane, A.M.** (2005). When and how to develop domain-specific languages, *ACM Comput. Surv.*, ***37***(4), 316–344.

[12] **Olukotun, K.** (2012). High performance embedded domain specific languages, *SIGPLAN Not.*, ***47***(9), 139–140.

[13] **Karsai, G.**, **Krahn, H.**, **Pinkernell, C.**, **Rumpe, B.**, **Schneider, M. ve Völkel, S.** (2009). Design Guidelines for Domain Specific Languages, ***M. Rossi, J. Sprinkle, J. Gray ve J.P. Tolvanen****, (düzenleyenler), Proceedings of the 9th OOPSLA Workshop on Domain-Specific Modeling (DSM'09)*, s.7–13.

[14] **Voelter, M.**, **Benz, S.**, **Dietrich, C.**, **Engelmann, B.**, **Helander, M.**, **Kats, L.C.L.**, **Visser, E. ve Wachsmuth, G.** (2013). *DSL Engineering - Designing, Implementing and Using Domain-Specific Languages*, dslbook.org.

[15] **Kamin, S.N. ve Hyatt, D.** (1997). A special-purpose language for picture-drawing, *Proceedings of the Conference on Domain-Specific Languages on Conference on Domain-Specific Languages (DSL), 1997*, DSL'97, USENIX Association, Berkeley, CA, USA, s.23–23.

[16] **Schneider, K.A. ve Cordy, J.R.**, (2002), AUI: A Programming Language for Developing Plastic Interactive Software.

[17] **Crew, R.F.** (1997). ASTLOG: A Language for Examining Abstract Syntax Trees, *In Proceedings of the USENIX Conference on Domain-Specific Languages*, s.229–242.

[18] LWC2011 Comparison Matrix, `http://www.languageworkbenches.net/index.php?title=LWC2011_Comparison_Matrix`, alındığı tarih: 10.02.2013.

[19] `http://www.jetbrains.com/mps`, alındığı tarih: 20.04.2013.

[20] `http://www.eclipse.org/Xtext`, alındığı tarih: 20.04.2013.

[21] `http://www.eclipse.org/xtend`, alındığı tarih: 20.04.2013.

[22] **Kats, L.C.L. ve Visser, E.** (2010). The Spoofax Language Workbench, *Companion to the Conference on Systems, Programming, Languages, and Applications: Software for Humanity (SPLASH 2010)*, ACM.

[23] **Heering, J.**, **Hendriks, P.R.H.**, **Klint, P. ve Rekers, J.**, (1992), The Syntax Definition Formalism SDF - Reference Manual.

[24] **Kalleberg, K.T. ve Visser, E.**, Syntax Definition in SDF, `http://hydra.nixos.org/build/2583471/download/1/manual/chunk-chapter/tutorial-sdf.html`, alındığı tarih: 05.01.2013.

[25] **Kalleberg, K.T. ve Visser, E.** (2007). Spoofax: An Interactive Development Environment for Program Transformation with Stratego/XT, *Proceedings of the Seventh Workshop on Language Descriptions, Tools and Applications (LDTA 2007)*, Electronic Notes in Theoretical Computer Science, Elsevier, Braga, Portugal.

EKLER

EK A.1 : Şekil 6.1 İçin Üretilen Java Kodu
EK A.2 : Şekil 6.2 İçin Üretilen Java Kodu
EK A.4 : Şekil 6.3 İçin Üretilen Java Kodu
EK A.5 : Şekil 6.4 İçin Üretilen Java Kodu
EK A.6 : Şekil 6.5 İçin Üretilen Java Kodu
EK B : Yol Haritası

EK A.1

```
import java.util.*;
import AbstractClasses.HyperHeuristic;
import AbstractClasses.ProblemDomain;
import Heuristic.*;
import HeuristicSelection.*;
import Acceptance.*;
import ExternalMethods.*;
import ApplyHeuristic.*;

public class example extends HyperHeuristic {
  public example(long seed) {
    super(seed);
}

  int[] hList ;

  public void solve(ProblemDomain problem) {
    hList = new int[problem.getNumberOfHeuristics()];

    int[]local_search_heuristics =
        problem.getHeuristicsOfType(
          ProblemDomain.HeuristicType.LOCAL_SEARCH);

    int[] mutation_heuristics =
        problem.getHeuristicsOfType(
          ProblemDomain.HeuristicType.MUTATION);

    int[] crossover_heuristics =
        problem.getHeuristicsOfType(
          ProblemDomain.HeuristicType.CROSSOVER);

    for(int i=0;i<local_search_heuristics.length;i++)
    {
      hList[i] = local_search_heuristics[i];
    }
    int hListIndex = local_search_heuristics.length;

    for(int i=0;i<mutation_heuristics.length;i++)
    {
      hList[hListIndex+i] = mutation_heuristics[i];
    }
    hListIndex = mutation_heuristics.length;
```

```
    for(int i=0;i<crossover_heuristics.length;i++)
    {
      hList[hListIndex+i] = crossover_heuristics[i];
    }

    double delta = 0;
    int number_of_heuristics = problem.getNumberOfHeuristics();

    SimpleRandomSelection hs =
        new SimpleRandomSelection(hList, rng);

    NonImprovingWithProbabilityAcceptance ac =
        new NonImprovingWithProbabilityAcceptance(rng, 0.5);

    int solutionmemorysize = 1;
    problem.setMemorySize(solutionmemorysize+1);

    double[] curr_obj_function =
      new double[solutionmemorysize+1];

    for (int x = 0; x < solutionmemorysize+1; x++) {
      problem.initialiseSolution(x);
      curr_obj_function[x] = problem.getFunctionValue(x);
    }

    ApplyHeuristic heuristicApplier = new ApplyHeuristic();

    while(!hasTimeExpired())
    {
      Heuristic h1 = hs.next();
      int s = 0;
      int n1 = solutionmemorysize;
      curr_obj_function[n1] =
        heuristicApplier.apply(problem, h1, s, n1);
      delta = curr_obj_function[s] - curr_obj_function[n1];

      if(ac.check(curr_obj_function[s], curr_obj_function[n1]))
      {
        problem.copySolution(n1, s);
        curr_obj_function[s] = curr_obj_function[n1];
      }
    }
  }

  public String toString() {
    return "example";
  }
}
```

EK A.2

```
import java.util.*;
import AbstractClasses.HyperHeuristic;
import AbstractClasses.ProblemDomain;
import Heuristic.*;
import HeuristicSelection.*;
import Acceptance.*;
import ExternalMethods.*;
import ApplyHeuristic.*;

public class example extends HyperHeuristic {
  public example(long seed) {
    super(seed);
}

  int[] hList ;

  public void solve(ProblemDomain problem) {
    hList = new int[problem.getNumberOfHeuristics()];

    int[]local_search_heuristics =
        problem.getHeuristicsOfType(
          ProblemDomain.HeuristicType.LOCAL_SEARCH);

    int[] mutation_heuristics =
        problem.getHeuristicsOfType(
          ProblemDomain.HeuristicType.MUTATION);

    int[] crossover_heuristics =
        problem.getHeuristicsOfType(
          ProblemDomain.HeuristicType.CROSSOVER);

    for(int i=0;i<local_search_heuristics.length;i++)
    {
      hList[i] = local_search_heuristics[i];
    }
    int hListIndex = local_search_heuristics.length;

    for(int i=0;i<mutation_heuristics.length;i++)
    {
      hList[hListIndex+i] = mutation_heuristics[i];
    }
    hListIndex = mutation_heuristics.length;

    for(int i=0;i<crossover_heuristics.length;i++)
    {
      hList[hListIndex+i] = crossover_heuristics[i];
    }
```

```
double delta = 0;

int number_of_heuristics = problem.getNumberOfHeuristics();

SimpleRandomSelection hs1 =
  new SimpleRandomSelection(
    externalMethods.
      getHeuristicList("mutation", problem), rng);

SimpleRandomSelection hs2 =
  new SimpleRandomSelection(
    externalMethods.
      getHeuristicList("local_search", problem), rng);

SimpleRandomSelection hs3 =
  new SimpleRandomSelection(
    externalMethods.
      getHeuristicList("crossover", problem), rng);

AcceptAll ac = new AcceptAll();

int solutionmemorysize = 10;

problem.setMemorySize(solutionmemorysize+1);

double[] curr_obj_function =
  new double[solutionmemorysize+1];

for (int x = 0; x < solutionmemorysize+1; x++) {
  problem.initialiseSolution(x);
  curr_obj_function[x] = problem.getFunctionValue(x);
}

ApplyHeuristic heuristicApplier = new ApplyHeuristic();

while(!hasTimeExpired())
{
  Heuristic h1 = hs1.next();
  int s = 0;
  Double sValue = Double.POSITIVE_INFINITY;
  for (int x = 0; x < solutionmemorysize; x++) {
    if (curr_obj_function[x] < sValue) {
      sValue = curr_obj_function[x];
      s = x;
    }
  }
  int n1 = solutionmemorysize;
```

```
curr_obj_function[n1] =
  heuristicApplier.apply(problem, h1, s, n1);

delta = curr_obj_function[s] - curr_obj_function[n1];

if(ac.check(
            curr_obj_function[s],
            curr_obj_function[n1]))
{
  problem.copySolution(n1, s);
  curr_obj_function[s] = curr_obj_function[n1];
}

Heuristic h2 = hs2.next();

int n2 = solutionmemorysize;

curr_obj_function[n2] =
  heuristicApplier.apply(problem, h2, s, n2);

delta = curr_obj_function[s] - curr_obj_function[n2];

if(ac.check(
            curr_obj_function[s],
            curr_obj_function[n2])
            )
{
  problem.copySolution(n2, s);
  curr_obj_function[s] = curr_obj_function[n2];
}

ArrayList<Integer> s3 = new ArrayList<Integer>();

int tmp = rng.nextInt(solutionmemorysize);

s3.add(tmp);

int numOfExp = 2;
numOfExp--;

while(numOfExp > 0)
{
  tmp = rng.nextInt(solutionmemorysize);

  while (true) {
    if (s3.contains(tmp)) {
      tmp = rng.nextInt(solutionmemorysize);
    } else {break;}}
```

```
        s3.add(tmp);
        numOfExp--;
      }

      Heuristic h3 = hs3.next();

      int n3 = solutionmemorysize;

      curr_obj_function[n3] =
        heuristicApplier.apply(
          problem, h3, s3.get(0), s3.get(1), n3);

      delta = curr_obj_function[s3.get(0)] -
        curr_obj_function[n3];

      if(ac.check(
                  curr_obj_function[s3.get(0)],
                  curr_obj_function[n3]))
      {
        problem.copySolution(n3, s3.get(0));
        curr_obj_function[s3.get(0)] = curr_obj_function[n3];
      }

      int currentBest = 0;
      Double currentBestValue = Double.POSITIVE_INFINITY;
      for (int x = 0; x < solutionmemorysize; x++) {
        if (curr_obj_function[x] < currentBestValue) {
          currentBestValue = curr_obj_function[x];
          currentBest = x;
        }
      }
    }
  }

  public String toString() {
    return "example";
  }
}
```

EK A.3

```
import java.util.*;
import AbstractClasses.HyperHeuristic;
import AbstractClasses.ProblemDomain;
import Heuristic.*;
import HeuristicSelection.*;
import Acceptance.*;
import ExternalMethods.*;
import ApplyHeuristic.*;

public class example extends HyperHeuristic {
  public example(long seed) {
    super(seed);
}

  int[] hList ;

  public void solve(ProblemDomain problem) {
    hList = new int[problem.getNumberOfHeuristics()];

    int[]local_search_heuristics =
        problem.getHeuristicsOfType(
          ProblemDomain.HeuristicType.LOCAL_SEARCH);

    int[] mutation_heuristics =
        problem.getHeuristicsOfType(
          ProblemDomain.HeuristicType.MUTATION);

    int[] crossover_heuristics =
        problem.getHeuristicsOfType(
          ProblemDomain.HeuristicType.CROSSOVER);

    for(int i=0;i<local_search_heuristics.length;i++)
    {
      hList[i] = local_search_heuristics[i];
    }
    int hListIndex = local_search_heuristics.length;

    for(int i=0;i<mutation_heuristics.length;i++)
    {
      hList[hListIndex+i] = mutation_heuristics[i];
    }
    hListIndex = mutation_heuristics.length;

    for(int i=0;i<crossover_heuristics.length;i++)
    {
      hList[hListIndex+i] = crossover_heuristics[i];
    }
```

```
double delta = 0;

int number_of_heuristics = problem.getNumberOfHeuristics();

SimpleRandomSelection hs1 =
  new SimpleRandomSelection(
    externalMethods.
      getHeuristicList("mutation, local_search", problem),
        rng);

SimpleRandomSelection hs2 =
  new SimpleRandomSelection(
    externalMethods.
       getHeuristicList("local_search_heuristic", problem),
        rng);

AcceptAll ac = new AcceptAll();

Heuristic h2 = hs2.next();

int solutionmemorysize = 1;

problem.setMemorySize(solutionmemorysize+1);

double[] curr_obj_function =
  new double[solutionmemorysize+1];

for (int x = 0; x < solutionmemorysize+1; x++) {
  problem.initialiseSolution(x);
  curr_obj_function[x] = problem.getFunctionValue(x);
}

ApplyHeuristic heuristicApplier = new ApplyHeuristic();

while(!hasTimeExpired())
{
  Heuristic h1 = hs1.next();
  int s = 0;
  int n = solutionmemorysize;

  curr_obj_function[n] =
    heuristicApplier.apply(problem, h1, s, n);

  delta = curr_obj_function[s] - curr_obj_function[n];

  if (externalMethods.contains(
      externalMethods.
        getHeuristicList("mutation", problem),
```

```
              h1.getHeuristicIndex())
        ){
          int n = solutionmemorysize;
          curr_obj_function[n] =
            heuristicApplier.apply(problem, h2, n, n);
          delta = curr_obj_function[n] -
            curr_obj_function[n];
      }

      if(ac.check(
        curr_obj_function[s],
        curr_obj_function[n]))
      {
        problem.copySolution(n, s);
        curr_obj_function[s] = curr_obj_function[n];
      }
    }
  }

  public String toString() {
    return "example";
  }
}
```

EK A.4

```
import java.util.*;
import AbstractClasses.HyperHeuristic;
import AbstractClasses.ProblemDomain;
import Heuristic.*;
import HeuristicSelection.*;
import Acceptance.*;
import ExternalMethods.*;
import ApplyHeuristic.*;

public class example extends HyperHeuristic {
  public example(long seed) {
    super(seed);
}

  int[] hList ;

  public void solve(ProblemDomain problem) {
    hList = new int[problem.getNumberOfHeuristics()];

    int[]local_search_heuristics =
        problem.getHeuristicsOfType(
          ProblemDomain.HeuristicType.LOCAL_SEARCH);

    int[] mutation_heuristics =
        problem.getHeuristicsOfType(
          ProblemDomain.HeuristicType.MUTATION);

    int[] crossover_heuristics =
        problem.getHeuristicsOfType(
          ProblemDomain.HeuristicType.CROSSOVER);

    for(int i=0;i<local_search_heuristics.length;i++)
    {
      hList[i] = local_search_heuristics[i];
    }
    int hListIndex = local_search_heuristics.length;

    for(int i=0;i<mutation_heuristics.length;i++)
    {
      hList[hListIndex+i] = mutation_heuristics[i];
    }
    hListIndex = mutation_heuristics.length;

    for(int i=0;i<crossover_heuristics.length;i++)
    {
      hList[hListIndex+i] = crossover_heuristics[i];
    }
```

```
  double delta = 0;

  int number_of_heuristics = problem.getNumberOfHeuristics();

  myHeuristic hs1 = new myHeuristic(problem);

  AcceptAll ac = new AcceptAll();

  int solutionmemorysize = 10;

  problem.setMemorySize(solutionmemorysize+1);

  double[] curr_obj_function =
    new double[solutionmemorysize+1];

  for (int x = 0; x < solutionmemorysize+1; x++) {
    problem.initialiseSolution(x);
    curr_obj_function[x] = problem.getFunctionValue(x);
  }

  ApplyHeuristic heuristicApplier = new ApplyHeuristic();

  while(!hasTimeExpired())
  {
    Heuristic h1 = hs1.next();

    int s = 0;

    int n = solutionmemorysize;

    curr_obj_function[n] =
      heuristicApplier.apply(problem, h1, s, n);

    delta = curr_obj_function[s] - curr_obj_function[n];

    if(ac.check(
      curr_obj_function[s],
      curr_obj_function[n]))
    {
      problem.copySolution(n, s);
      curr_obj_function[s] = curr_obj_function[n];
    }
  }
}


public class myHeuristic{
  ArrayList<Integer> h = new ArrayList<Integer>();
```

```
    {
      for (int i = 0; i < hList.length; i++) {
        h.add(hList[i]);
      }

      Collections.shuffle(h);
    }

    int current = -1;

    ProblemDomain problem;
    int solutionmemorysize;
    ApplyHeuristic heuristicApplier;
    double delta;
    double[] curr_obj_function;

    public myHeuristic(ProblemDomain problem)
    {
      this.problem = problem;
      heuristicApplier = new ApplyHeuristic();
      delta = 0;
    }

    public void setMemorySize(int size)
    {
      solutionmemorysize = size;
      curr_obj_function = new double[size+1];
    }

    Heuristic next()
    {
      current = current + 1;
      return new Heuristic(h.get(current), "");
    }
  }

  public String toString() {
    return "example";
  }
}
```

EK A.5

```
import java.util.*;
import AbstractClasses.HyperHeuristic;
import AbstractClasses.ProblemDomain;
import Heuristic.*;
import HeuristicSelection.*;
import Acceptance.*;
import ExternalMethods.*;
import ApplyHeuristic.*;

public class example extends HyperHeuristic {
  public example(long seed) {
    super(seed);
}

  int[] hList ;

  public void solve(ProblemDomain problem) {
    hList = new int[problem.getNumberOfHeuristics()];

    int[]local_search_heuristics =
        problem.getHeuristicsOfType(
          ProblemDomain.HeuristicType.LOCAL_SEARCH);

    int[] mutation_heuristics =
        problem.getHeuristicsOfType(
          ProblemDomain.HeuristicType.MUTATION);

    int[] crossover_heuristics =
        problem.getHeuristicsOfType(
          ProblemDomain.HeuristicType.CROSSOVER);

    for(int i=0;i<local_search_heuristics.length;i++)
    {
      hList[i] = local_search_heuristics[i];
    }
    int hListIndex = local_search_heuristics.length;

    for(int i=0;i<mutation_heuristics.length;i++)
    {
      hList[hListIndex+i] = mutation_heuristics[i];
    }
    hListIndex = mutation_heuristics.length;

    for(int i=0;i<crossover_heuristics.length;i++)
    {
      hList[hListIndex+i] = crossover_heuristics[i];
    }
```

```
  double delta = 0;

  int number_of_heuristics = problem.getNumberOfHeuristics();

  SimpleRandomSelection p1 =
    new SimpleRandomSelection(hList, rng);

  myAcceptance accepter = new myAcceptance(problem);

  int solutionmemorysize = 10;

  problem.setMemorySize(solutionmemorysize+1);

  double[] curr_obj_function =
    new double[solutionmemorysize+1];

  for (int x = 0; x < solutionmemorysize+1; x++) {
    problem.initialiseSolution(x);
    curr_obj_function[x] = problem.getFunctionValue(x);
  }

  ApplyHeuristic heuristicApplier = new ApplyHeuristic();

  while(!hasTimeExpired())
  {
    Heuristic h = p1.next();

    int s = 0;

    int n = solutionmemorysize;

    curr_obj_function[n] =
      heuristicApplier.apply(problem, h, s, n);

    delta = curr_obj_function[s] - curr_obj_function[n];

    if(accepter.check(
      curr_obj_function[s],
      curr_obj_function[n]))
    {
      problem.copySolution(n, s);
      curr_obj_function[s] = curr_obj_function[n];
    }
  }
}


public class myAcceptance{
```

```
    ProblemDomain problem;
    int solutionmemorysize;
    ApplyHeuristic heuristicApplier;
    double delta;
    double[] curr_obj_function;

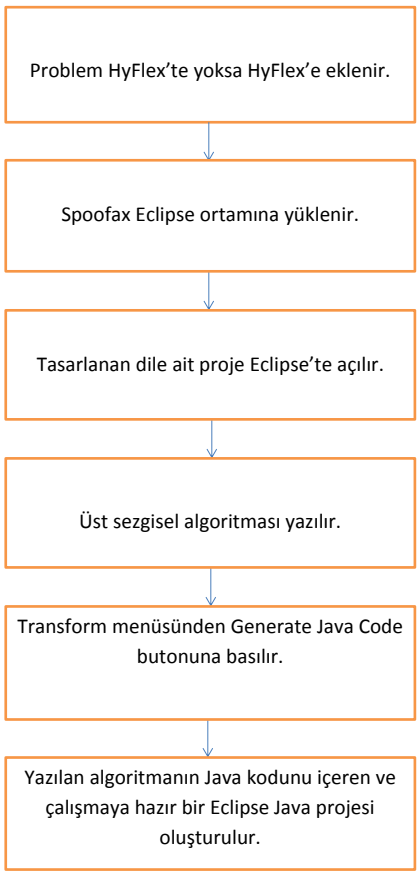
    public myAcceptance(ProblemDomain problem)
    {
      this.problem = problem;
      heuristicApplier = new ApplyHeuristic();
      delta = 0;
    }

    public void setMemorySize(int size)
    {
      solutionmemorysize = size;
      curr_obj_function = new double[size+1];
    }

    Boolean check(double exSol, double newSol)
    {
      delta = -(exSol - newSol);
      return newSol > exSol;
    }
  }

  public String toString() {
    return "example";
  }
}
```

EK B



Şekil B.1: Tasarlanan dilde algoritma yazmak ve çalışabilir kod üretmek için izlenecek yol haritası.

ÖZGEÇMİŞ

Ad Soyad: Hilal Kevser Cora

Doğum Yeri ve Tarihi: İstanbul, 12.02.1988

Adres: İstanbul Ulaşım A.Ş. Ferhatpaşa Metro Tesisleri, Esenler, İstanbul

E-Posta: hilalkevsercora@outlook.com

Lisans: İstanbul Teknik Üniversitesi Bilgisayar Mühendisliği

Mesleki Deneyim: 2010 - Halen, Yazılım Geliştirme Mühendisi, İstanbul Ulaşım A.Ş.

Yayın ve Patent Listesi:

- Subaşı, H. K., 2010: Automatic XPath Expression Generation with Grammatical Evolution, *1st International Symposium on Computing in Science & Engineering (ISCSE)*, June, 3-5, 2010, Kuşadası, Aydın, Turkey

TEZDEN TÜRETİLEN YAYINLAR/SUNUMLAR

- **Cora H.K.**, Etaner-Uyar Ş. and Uyar H.T., 2013: HH-DSL: A Domain Specific Language for Selection Hyper-heuristics, *EvoSoft Workshop, GECCO 2013: Genetic and Evolutionary Computation Conference*, July 6-10, 2013 Amsterdam, Netherlands.