Informatics

An Interactive Visualization Approach to Tackle Design Constraints in a Rule-Based Recommendation System

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Visual Computing

eingereicht von

Bernhard Pointner, BSc
Matrikelnummer 01527081

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Univ.Prof. Mag. Dr. Silvia Miksch
Mitwirkung: Dipl.-Ing. Dr. Johanna Schmidt

Wien, 22. August 2022

Bernhard Pointner	Silvia Miksch

Informatics

An Interactive Visualization Approach to Tackle Design Constraints in a Rule-Based Recommendation System

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Visual Computing

by

Bernhard Pointner, BSc
Registration Number 01527081

to the Faculty of Informatics

at the TU Wien

Advisor: Univ.Prof. Mag. Dr. Silvia Miksch
Assistance: Dipl.-Ing. Dr. Johanna Schmidt

Vienna, 22nd August, 2022

Bernhard Pointner	Silvia Miksch

Technische Universität Wien
A-1040 Wien ▪ Karlsplatz 13 ▪ Tel. +43-1-58801-0 ▪ www.tuwien.at

Erklärung zur Verfassung der Arbeit

Bernhard Pointner, BSc

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 22. August 2022

Bernhard Pointner

Acknowledgements

First of all, I want to express my thanks to my advisors Johanna Schmidt and Silvia Miksch. They supported me in shaping the underlying ideas of the thesis and contributed with their expertise in information visualization.

Furthermore, I am grateful for all the experiences and support during my studies in difficult times. It resulted in awesome friendships and memories of my time at university. Mainly, I want to thank my colleagues Elitza Vasileva and Dominik Scholz for their motivating attitude and collaboration during our journey that led to remarkable successes.

Finally, I want to thank my parents who made all this possible for me and other close people in my life. I am convinced that they know how grateful I am for their versatile support, not only at the finishing straight of this thesis.

This work was enabled by the Competence Centre VRVis. VRVis is funded by BMK, BMDW, Styria, SFG, Tyrol and Vienna Business Agency in the scope of COMET (879730) which is managed by FFG.

Kurzfassung

Bisher sind bestehende regelbasierte Frameworks, die Richtlinien für das Visualisierungsdesign kodieren, entweder zu komplex, zu mühsam zu pflegen und zu erweitern oder führen zu vernachlässigbaren Ergebnissen. Aus diesem Grund wurde Draco von Dominik Moritz et al. entwickelt - ein automatisiertes Visualisierungs-Empfehlungssystem, das Designwissen als auf logischen Ausdrücken basierte Regeln zur Einschränkung des möglichen Eregebnissets in Answer Set Programming (ASP) formalisiert. Es hat zum Ziel die Lücke zwischen Visualisierungsrichtlinien und deren Anwendung in Visualisierungswerkzeugen zu schließen. Unter Berücksichtigung aller Vorteile und Möglichkeiten von Draco verlieren jedoch selbst Visualisierungsexpert:innen mit einer zunehmenden Anzahl an Regeln und beinhaltetem Designwissen den Überblick und haben Mühe, die vom System getroffenen automatisierten Empfehlungsentscheidungen nachzuvollziehen.

Diese Arbeit schlägt einen interaktiven Visualisierungsansatz für Dracos Regelwerk vor. Es soll Visualisierungsexpert:innen in die Lage versetzen, identifizierte Aufgaben bezüglich der Wissensbasis zu lösen und sie beim Verständnis des Systems unterstützen. Um die notwendigen Daten für diesen Visualisierungsansatz zu gewinnen, erweitern wir die bestehende Datenextraktionsstrategie von Draco um eine Datenverarbeitungsarchitektur, die in der Lage ist, Informationen aus der Wissensbasis zu extrahieren. Eine überarbeitete Version der ASP-Grammatik liefert die Grundlage für diese Datenverarbeitungsstrategie. Die daraus resultierenden Merkmale der Regeln werden dann unter Verwendung einer Hypergraphstruktur innerhalb der radial angeordneten Regeln visualisiert. Die hierarchischen Kategorien der Regeln werden durch Bögen angezeigt, die die Regeln umgeben. Darüber hinaus schlagen wir eine getrennte, aber interaktiv zusammenhängende Ansicht von Dracos Visualisierungsempfehlungen und unserer Visualisierung für die Regeln vor. Diese geteilte Ansicht soll es Expert:innen ermöglichen, Verstöße gegen die Designregeln auf Basis von Hervorhebungen betroffener Regeln interaktiv zu untersuchen.

Der implementierte Prototyp verifiziert die Machbarkeit der Datenextraktionsstrategie und des vorgeschlagenen Visualisierungsansatzes. Eine Evaluierung des Prototyps, die qualitative und quantitative Methoden kombiniert, zeigt offene Schwierigkeiten und irreführende Darstellungen auf. Die Evaluierungsergebnisse bestätigen jedoch auch die Effektivität und den Wert des Prototyps. Dieser ermöglicht Einsicht und ein besseres Verständnis von Dracos Empfehlungsprozess und des zugrundeliegende Regelwerks zu erhalten.

Abstract

So far, existing rule-based frameworks that encode visualization design guidelines are either too complex, laborious to maintain and extend, or produce negligible results. Therefore, Draco has been developed by Dominik Moritz et al. - an automated visualization recommendation system formalizing design knowledge as logical constraints in Answer Set Programming (ASP). It is supposed to close the gap between visualization guidelines and their application in visualization tools. However, taking all advantages and capabilities of Draco into account, with an increasing set of constraints and incorporated design knowledge, even visualization experts lose overview and struggle to retrace the automated recommendation decisions made by the system.

This thesis proposes an interactive visualization approach to Draco's constraints. It is supposed to enable visualization experts to accomplish identified tasks regarding the knowledge-base and support them in understanding the system. To acquire the necessary data for this visualization approach, we extend the existing data extraction strategy of Draco with a data processing architecture capable of extracting features of interest from the knowledge-base. A revised version of the ASP grammar provides the basis for this data processing strategy. The resulting incorporated and shared features of the constraints are then visualized using a hypergraph structure inside the radial-arranged constraints of the elaborated visualization. The hierarchical categories of the constraints are indicated by arcs surrounding the constraints. In addition, we suggest a split but connected view of Draco's visualization recommendations and our visualization. This faceted view is supposed to enable visualization experts to interactively explore the design rules' violations based on highlighting respective constraints or recommendations.

The implemented prototype verifies the feasibility of the data extraction strategy and the proposed visualization approach. An evaluation of the prototype combining qualitative and quantitative methods reveals open difficulties and misleading representations. However, the evaluation results also confirm the prototype's effectiveness and value in acquiring insights into Draco's recommendation process and design constraints.

Contents

CHAPTER 1

Introduction

1.1 Motivation and Problem Statement

Effective information visualization is becoming crucial with today's increasing number of data and its complexity. For a long time, only researchers and data analysts, so-called experts, have been associated with analytical tasks and the process of visualizing complex data. However, with increasing digitization, not only experts process data, but also children from an early age are confronted with data visualizations in their daily life in school, at home, on TV, on social media, and so on. It is insufficient only to be able to read and understand data visualizations but also to be capable of creating visualizations by yourself (user) to get insights and to derive value from a data domain of interest. Such insights involve detecting trends, spotting anomalies and correlations, clusters, patterns, distributions, or compositions [VHS^{+}17].

Whereas earlier data visualizations have to be drawn manually, current popular tools, like Excel and Tableau, assist the user in creating data visualizations by proposing popular types of visual encodings, such as line charts, bar charts, or scatter plots. However, the story of creating effective visualizations does not end with knowing which chart type to select. Visual encodings, such as visual channels, mark types, binning and aggregation operations, faceted view, and others, must be carefully selected and used to create a visualization for given data. This knowledge of how to properly use and apply these encodings can be learned from books, scientific papers, courses in schools, and similar. However, studies show that visualization designers still not always follow these guidelines due to various reasons [BRBF14a, BRBF14b].

Therefore, it is unsurprising that visualization recommendation, especially automated visualization design, have enjoyed an increasing interest in recent years. Saket et al. argue that it is time to go beyond manually curated and applied visualization design guidelines [SML^{+}18].

According to Moritz et al. [MWN+19], the existing design principles and guidelines are generally incomplete and continually evolving. Consequently, as mentioned by Moritz et al., the outcome of regularly published empirical study results has to be incorporated into these systems to ensure the observance of theories concerning effective design by time. However, due to Moritz et al. [MWN+19], we still lack a formal framework for representing design knowledge that the research community can easily update to fit those requirements. Although Machine-Learning (ML)-based approaches enjoy increasing popularity in the field of automated visualization design, the process of applying the new findings and theories is slowed down, which is in the ML-based systems' nature. These systems can only learn from previous visualizations in which the study results are not yet taken into account and are, therefore, unsuitable for testing new theories and study results concerning visualization design.

Consequently, Moritz et al. proposed a system called Draco [MWN+19] to address this problem. It is based on a novel formal model that represents visualizations as a set of logical facts. Furthermore, it constitutes design guidelines and principles as an extendable list of ∼230 hard and soft constraints over these facts – the *knowledge-base*. Whereas hard constraints prune all visual encodings, which would result in non-expressive or ill-formed visualization specifications, the soft constraints determine the final ranking of remaining design specifications based on their violations. The whole knowledge-base, including logical facts, the constraints, and the recommendation query definition, is expressed in Answer Set Programming (Answer Set Programming (ASP)), a declarative constraint-based logic programming language.

Draco proposes numerous suggestions for improving the system, and possible usage scenarios, such as design space enumeration, visualization model comparison, or design debugging [MWN+19]. Of course, not all of these suggestions can be addressed at once. However, we believe that the aspect of understanding the system's knowledge-base and recommendation process should be prioritized. Making the system more accessible and understandable by the research community and visualization designers would create the basis for further developments, such as using Draco for testing and comparing new theories and empirical study results.

Therefore, we analyzed Draco regarding this aspect and encountered the following weaknesses of the system:

- Visualization designers have to be familiar with logic programming in general, but especially with ASP in order to be able to work with the knowledge-base. The current implementation of Draco not only requires the user to know how to formulate logical expressions syntactically correct but also to know which parameters are already encoded into the system and how the expressions relate to each other. Since the knowledge-base is only textually encoded and no advanced filter or reorganization mechanisms are available, it is very time-consuming to investigate the system's ingredients and purpose.
- Although the underlying constraint solver provides information on which constraints

have been violated concerning a recommended visualization, it is perceptually hard to trace the recommendation process and obtain a "big picture". On the one hand, recommended visualizations can not be directly compared, and on the other hand, the influence of specific soft constraints concerning these recommendations is hard to derive.

In this thesis, we want to address these limitations by investigating the knowledge-base, extracting features of interest, and elaborating a proper visualization of the knowledge-base. Additionally, we connect the recommendations with the elaborated visualization to highlight the recommendations' violations and show constraints' influence on the recommendation results. However, we concentrate on only visualizing the hard and soft constraints as it is most difficult to get an overview here due to the number of rules.

1.2 Research Questions

This research aims to investigate the knowledge-base behind Draco and find proper tooling and visualization to support developers, researchers, and designers in understanding the system and its recommendation process.

We assume that adding meta-data to Draco's constraints and visualizing them in a proper and interactive way would help visualization experts gain the desired insights into the underlying knowledge-base (RQ1.1+RQ1.2). Furthermore, we suspect that visually connecting the recommendation results with the created visualization of the constraints and highlighting respective violations enables these experts to adjust the constraints and their weights accordingly to concrete preferences (RQ2.1+RQ2.2).

To verify these claims, we aim to answer the following research questions:

RQ1.1: Which predicates, variables, and other syntax features are shared by different design constraints?

RQ1.2: How does the distribution of costs look like within the constraints?

RQ2.1: Which constraints are violated by specifically recommended visualizations and to which degree?

RQ2.2: Which recommended visualizations are violated by specific constraints and to which degree?

1.3 Methodology

1. **Research and Literature Review**
   - Analysis of the state of the start of visualization recommendation systems with focus on rule-based ones.
   - Investigation and comparison of different grammars for ASP.
   - Examination of different visualizations for logic programming, like ASP, graph[1]-based data, and hierarchical-based data.
2. **Grammar and Design Investigation**
   - Elaboration of a suitable grammar to parse the knowledge-base and extract features of interest.
   - Analysis of extracted features to find an appropriate visual mapping for the design constraints.
   - Iterative design process to visualize the constraints, their interrelations, and to improve user experience.
3. **Prototype Development**
   - Selection of a suitable web-based framework as the underlying basis for the tool.
   - Integrating Draco's JavaScript module into the framework.
   - Testing different ASP grammars and parser generators to select the most suitable combination to parse Draco's knowledge-base and to create the Abstract Syntax Tree (AST).
   - Selection of most suitable visualization library to visualize constraints and their relations.
4. **Evaluation**
   - Expert-based analysis of the completeness of the elaborated grammar, its abilities, and limitations.
   - Qualitative and quantitative evaluation of the created prototype by performing a task-based performance analysis with target users and determining the visualization's value through a heuristic-based survey and qualitative questionnaire.

[1] *graph* is used as a synonym for *network*

CHAPTER 2

Related Work

The following chapter gives an overview of related work in different areas relevant to tackling and visualizing design constraints in the context of visualization recommendation systems. First, we give a brief overview of different existing recommendation systems and analyze respectively categorize them regarding their design of the recommendation engine. Then we will look at the various available types and versions of the grammar for ASP as the extraction of features of design constraints is an essential task to be capable of visualizing the constraints and their feature-based overlaps. Next, we examine existing web-based syntax parser projects and compare them regarding their performance and grammar support as we have to select a suitable ASP parser for our prototype. Then, we recap tools that proposed different visualizations in the context of knowledge representation formalism and logic programming. Finally, we conclude this chapter with different existing visual approaches and techniques to show relationships and overlaps between entities of a set and their hierarchical affiliation to categories.

2.1 Recommendation Systems

Although several integrations and applications of data visualization recommender systems have been proposed in the last years, they are barely known by data analysts, researchers, and non-experts [KFD19]. Most of these approaches are scientific-related and are not incorporated into any public available and known data analysis tool. On the contrary, the most popular available systems are the Show Me feature [MHS07] of Tableau [MHS07], Explore in Google Sheets [VWS+18], Watson Analytics[1], and Spotfire[2].

As mentioned by Kubernátová et al. [KFD19], Kaur and Ownibi distinguish 4 types of recommender systems [KO17]: Data Characteristics-Oriented, Task-Oriented, Domain

[1]`https://www.ibm.com/analytics` (accessed 2022/08/22)
[2]`https://www.tibco.com/products/tibco-spotfire` (accessed 2022/08/22)

Knowledge-Oriented, and User Preference-Oriented. This categorization is extended by Vartak et al. [VHS+17] with the field of 'Visual Ease of Understanding.' In addition to this categorization, we further distinguish between approaches recommending WHAT data to show [SS05, ESC16, VRM+15] in contrast to HOW to show it (like [WMA+16b, WQM+17, LQTL18, HBL+19, MWN+19, DD19]). Furthermore, these systems can be divided into approaches only proposing the visualization type and actual visual encodings. Lastly, the underlying recommendation strategies differ by rule-based, ML-based, and hybrid-based approaches.

The majority of the most relevant visualization recommender systems are data-driven. They recommend visualizations based on the characteristics of the data. Their main goal is to get insights of the dataset and to recognize interesting values, trends, patterns, distributions, compositions, and differences. Such tools are BHARAT [Gna81], APT [Mac86], Explore in Google Sheets [VWS+18], Watson Analytics, Google Sheets [VWS+18], Show Me [MHS07], VizDeck [KHPA12], Seedb [VRM+15], Voyager [WMA+16b], Voyager 2 [WQM+17], VizML [HBL+19], Draco [MWN+19], Data2Vis [DD19] and DeepEye [LQTL18].

In contrast to data-driven systems, task-oriented-based approaches focus on the user's intent respectively goal. Such approaches vary widely in their design and application area. For instance, the tool BOZ [Cas91] analyses tasks encoded as logical facts and produces graphics with perceptional instructions to reach the goal. IMPROVISE [ZF01] performs a natural language query on the user's intents to extract and automatically propose visual tasks. Other task-oriented systems are HARVEST [GW09] or the more recent one called DataSlicer [ACC+17]. However, these systems are mostly scientific-related and play a minor role in user-centered data visualization recommender systems.

According to recent approaches, it can be derived that systems suggesting actual visual encodings to get insights into a dataset are preferred over systems only proposing the type of data visualization to use. Although Kubernátová et al. [KFD19] propose a formal ML-friendly model only recommending chart types, they also suggest enhancing their system to provide information on how to create these charts.

Whereas earlier recommendation strategies mainly encoded decisions on visualization guidelines and principles as a set of rules and constraints, recent approaches increasingly use ML-based, respectively, hybrid-based recommendations. These systems learn relationships between design guidelines and principles and weight their influence on respective visual encodings. Such ML-based systems are, for example, Data2Vis [DD19] and VizML [HBL+19]. However, the predominant part of systems still rely on hand-crafted rules, such as APT [Mac86], SAGE [RKMG94], BOZ [Cas91], VizDeck [KHPA12], DIVE [HOH18], CompassQL [WMA+16a], Voyager [WMA+16b], Voyager 2 [WQM+17], and the most famous ones Show Me [MHS07] and Explore in Google Sheets [VWS+18]. Recent hybrid-based systems combine the advantages of rule-based and ML-based approaches by applying a mixed strategy, like DeepEye [LQTL18] and Draco [MWN+19].

Draco combines hand-crafted visualization facts, hard and soft constraints with learned weights from a Support Vector Machine (RankSVM) model trained on labeled visualization pairs. The underlying module processes an input consisting of a dataset definition, its partial specification, and a user task to a query definition describing a data schema and query constraints (see Figure 2.1). This query definition is combined with the predefined search space definition consisting of aggregate rules, well-formedness constraints, expressiveness constraints, and the preference model. Draco then calls the ASP solver Clingo [GKK+08a] to solve the resulting program and subsequently, to obtain ranked answer sets. These answer sets (also called models) are finally translated to Vega-Lite (VL) specifications [SMWH17]. A provided web-based Application Programming Interface (API) of Draco facilitates easy access to the module and decouples Draco from a tool using the system.

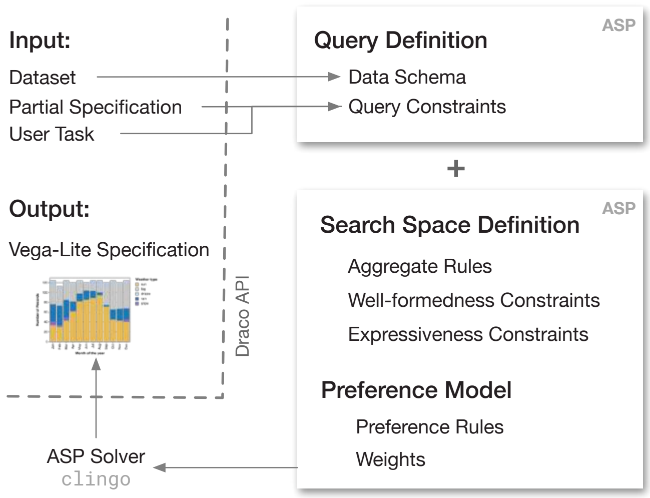
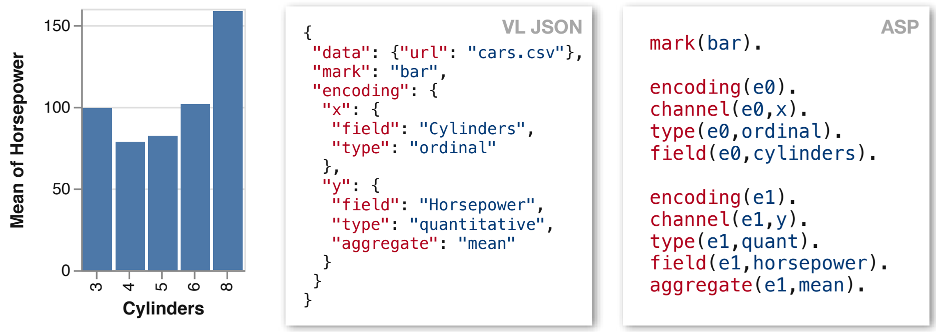


Figure 2.1: Draco's recommendation engine that compiles a user query consisting of a dataset, partial specification, and user task into a set of rules and combines them with the existing knowledge-base. The resulting ASP program is then processed and solved by Clingo to obtain an optimal answer set which is translated to a VL-based specification [SMWH17] - reprinted from Moritz [MWN+19].

An exemplary model of Draco's output is illustrated in Figure 2.2. It shows the underlying formal specification of a bar chart once as a VL-based specification in JavaScript Object Notation (JSON) format and the respective ASP-based definition. The encoding identifiers `e0` and `e1` group field, data type, and data transformation definitions for each encoding. These encodings reflect the two axis `x` (mean of horsepower) and `y` (cylinders).



```
{
 "data": {"url": "cars.csv"},
 "mark": "bar",
 "encoding": {
  "x": {
   "field": "Cylinders",
   "type": "ordinal"
  },
  "y": {
   "field": "Horsepower",
   "type": "quantitative",
   "aggregate": "mean"
  }
 }
}
```

```
mark(bar).

encoding(e0).
channel(e0,x).
type(e0,ordinal).
field(e0,cylinders).

encoding(e1).
channel(e1,y).
type(e1,quant).
field(e1,horsepower).
aggregate(e1,mean).
```

Figure 2.2: An example specification of a bar chart by VL and in the ASP notation resulting from Draco's recommendation pipeline. It defines a mark type, two encodings and their field, data type, and data transformation definition - reprinted from Moritz [MWN+19].

2.2 Grammars for Answer Set Programming

ASP is based on the concept of stable models as a result of difficult search problems. Dimopoulos and Köhler [DNK97] applied this concept for logic programming in their planning method for the first time. Since then, several adaptions and refinements have been made to this paradigm [SN98, MT99]. Lifschitz [Lif99] introduced the terminology `answer set` instead of `stable model` to highlight that those stable models are answers to a search problem.

The search problem is expressed in a declarative logic programming notation following the Prolog-style [CM03]. These queries are either processed by imperative parsers, like Clingo, or parser generators that are applying a Context-Free Grammar (CFG). Parser generators using grammars have the advantage of being quickly and easily adaptable as well as easy to comprehend, whereas imperative parsers can be implemented more efficiently. Therefore, in scientific papers related to ASP, the input language format is often specified by CFGs using production rules, such as Backus-Naur Form (BNF), Extended Backus-Naur Form (EBNF), Augmented Backus-Naur Form (ABNF), and similar.

Analogous to other knowledge representation and reasoning research areas, the standardization process of the input language of ASP led to more advanced and efficient solvers. This process was mainly driven by Gebser et al. [GKK+08a], Brewka et al. [BET11], and Calimeri et al. [CIR11], who proposed the first standardized versions of the grammar for ASP in different implementations of the BNF notation. This first standardized version, called ASP-Core, was revised in ASP-Core 2 [CFG+12] and has been regularly updated through ongoing ASP solving competition events since 2012.

The different types and versions of the ASP grammar mainly differ by the syntax of additional solving features of different ASP solvers, e.g., the optimization function of Clingo [GKK+08a] or the meta-syntax giving additional information to the solver [GKK+08b]. Basic answer set solvers use the syntax of ASP-Core 2 [CFG+12] to solve the question of whether a set of atoms is a valid answer set. More advanced solvers, like Gringo and Clingo, however, determine not only if it is a valid set, but whether it is an optimal one or not [GKK+08b]. They make use of so-called weak constraints with assigned weights defined as constants. The optimization functions then reason these special constraints.

Gringo respectively Clingo implemented an imperative approach to parse ASP syntax[3]. In their API documentation of the AST module they mentioned the underlying grammar. It is defined very language specific and is difficult to comprehend. However, it resembles an EBNF notation. The first and more generalized EBNF version of this input language specification was proposed by Gebser et al. [GKK+08a]. Besides, another very similar version can be found in the API documentation of the TweetyProject[4]. Although this grammar supports most of the features defined by Gringo and Clingo [GKK+08a], optimization functions and meta-syntax statements, like `#show`, `#hide`, `#include`, `#external`, `#program`, `#script`, and `#const` are not supported.

Depending on the BNF type of the different grammar versions proposed in the mentioned sources, the definition of the lexical values differs. These values are either written in a right-recursive style with empty words and alternative terminal symbols, as necessary for the basic BNF notation, or in Flex[5] syntax similar to Regular Expression (REGEX). The proposed grammar in ASP-Core 2 [CFG+12] is defined in the Flex notation. The usage of the different lexical value specifications depends on the system executing the grammar.

2.3 Parser Generators for Answer Set Programming

Since we want to investigate the knowledge-base of Draco in a web-based scenario, we analyzed a set of easily accessible and applicable existing ASP grammar parsers and parser generators regarding the accepted BNF types, the AST creation support, performance, and relevance (see Table 2.1). The relevance criterion is derived from the weekly downloads at the JavaScript node package manager platform NPM according to the publishing date of this thesis. The assessment scale is divided into low relevance (0-100 downloads p.w.), medium relevance (100-1000 downloads p.w.), and high relevance (>1000 downloads p.w.).

[3]`https://potassco.org/clingo/python-api/5.4/ast/#clingo.ast.AggregateFunction` (accessed 2022/08/22)

[4]`http://tweetyproject.org/api/1.17/net/sf/tweety/lp/asp/parser/ASPCore2Parser.html` (accessed 2022/08/22)

[5]`https://github.com/westes/flex` (accessed 2022/08/22)

[6]Relevance according recent downloads on npm (assessment scale from low, medium, high) (accessed 2022/08/22)

[7]`https://www.npmjs.com/package/bnf` (accessed 2022/08/22)

[8]`https://www.npmjs.com/package/bnf-parser` (accessed 2022/08/22)

[9]`https://www.npmjs.com/package/ebnf` (accessed 2022/08/22)

Generator/Parser	BNF Types	AST	Performance	Relevance[6]
(1) BNF[7]	BNF, ABNF, MBNF	N/A	N/A	low
(2) BNF-Parser[8]	Mix of BNF and EBNF	promising	N/A	low
(3) EBNF[9]	BNF, EBNF	supported	BNF slow EBNF fast	low
(4) EBNF-Parser[10]	modified BNF/EBNF	supported	N/A	medium
(5) ANTLR4[11]	modified EBNF	supported	fast	high

Table 2.1: List of relevant web-based context free parser generators and grammar parsers.

There are many other web-based parser generators like Jison[12], PEG.js[13], Nearley[14], or APG[15]. However, they all have in common that their grammar notations resemble the standards BNF, EBNF, ABNF or any other extension of BNF, but are not strictly standardized. Consequently, writing a grammar in their notation creates a strong dependency on their parser generator's implementation.

2.4 Visualizations in Context of Logic Programming

The visualizations in the context of logic programming are very diverse and address different goals and steps. These steps reach from interpreting the logic programs using a proper visualization of the program to the visualization of the solvers' results. For better comprehension, we categorized these visualizations into three classes: visualizations showing the program respectively the query encoded as logical expressions, visualizations showing the solving process of the logic program, and visualizations showing the answer sets. For this categorization, we analyzed related work to logic programming in ASP, Prolog, and Constraint Logic Programming (CLP):

Visualizations showing program dependencies

The first category is the most relevant regarding the context of this thesis. Therefore, we look closely at the related work in this field. As a starting point, dependency graphs are often used to show the relations between the units of a logic program – also often known as features or arguments represented as nodes, such as predicates, variables, constants, and similar. These graphs can be visualized as free node-link layout graphs, where the node positions are only dependent on their linked neighbor nodes or in tree respectively flow structures showing the deductions of the predicates.

[10]`https://www.npmjs.com/package/ebnf-parser` (accessed 2022/08/22)
[11]`https://www.npmjs.com/package/antlr4` (accessed 2022/08/22)
[12]`https://www.npmjs.com/package/jison` (accessed 2022/08/22)
[13]`https://pegjs.org` (accessed 2022/08/22)
[14]`https://www.npmjs.com/package/nearley` (accessed 2022/08/22)
[15]`https://www.npmjs.com/package/apg-js` (accessed 2022/08/22)

An example of free layout representing the dependencies of the units of a logic program in Prolog-style has been proposed by Seipel et al. [SHH03]. Their approach suggests a directed graph representing predicates as circles and `not` arguments as a rhombus with four right angles (see Figure 2.3). These glyphs are then connected by arrow links showing the directions of the deductions. However, the graph does not contain any variables or constants. They are abstracted by the arity of the predicates. The arity is shown in Figure 2.3 by *name/arity* where *arity* represents the number of arguments and *name* the identifier of the predicate.

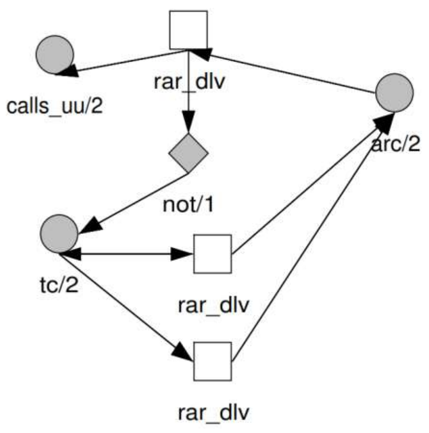
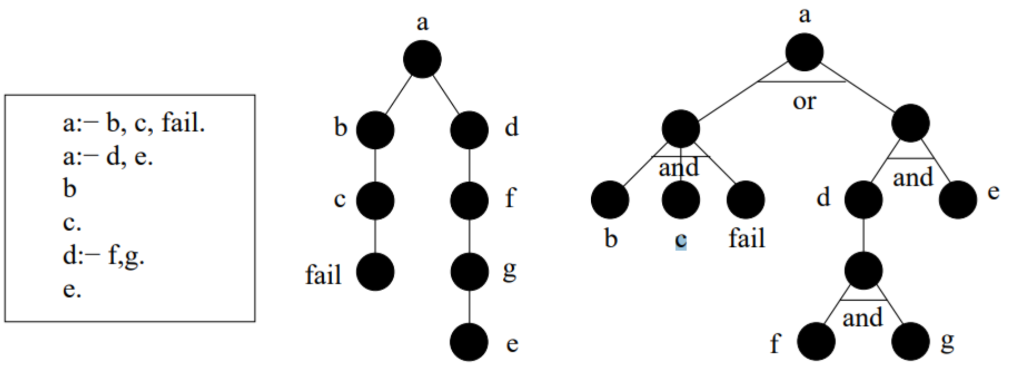


Figure 2.3: Acyclic dependency graph for predicates of Prolog programs in VISUR - reprinted from Seipel et al. [SHH03].

Another way to visualize logic programs in Prolog is by mapping the program to top-down trees, as mentioned by Cameron et al. in ViMer [CGDLBMM03]. They distinguish between Selective Linear Definite (SLD) trees and AND/OR trees representing the atoms of the program as nodes (see Figure 2.4). In the AND/OR tree, additional nodes are included that show AND links in the bodies of the rules. Multiple identical headers are branched out with an OR link. However, according to Adachi et al. [ATIY00], it is difficult to correlate the content of logic programs in Prolog to the corresponding nodes in the AND/OR tree.

The Integrated Development Environment (IDE) called ASPIDE [FRR11] for ASP integrates a visual editor for creating and modifying logic programs. The editor resembles a UML diagram typically used to describe classes and their properties in programming languages, like Java. Instead of classes and their relationships, the editor shows predicates and their use in rules, respectively constraints (see Figure 2.5). However, this representation is only applicable for visually developing logical programs. The editor does not provide any visualization giving an overview of the whole program and its dependencies.



```
a:– b, c, fail.
a:– d, e.
b
c.
d:– f,g.
e.
```

Figure 2.4: SLD (middle) versus AND-OR tree (right) formats for Prolog programs (left) - adapted from Cameron et al. [CGDLBMM03].

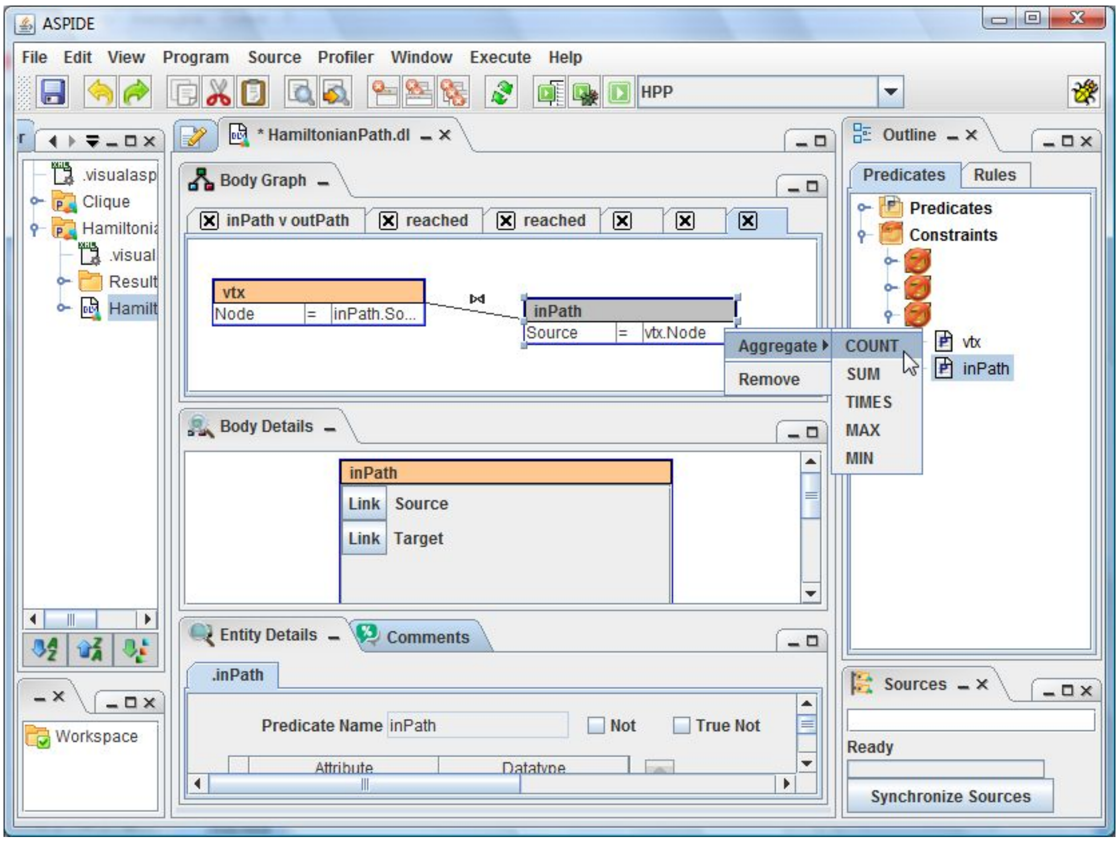


Figure 2.5: ASPIDE visual editor - reprinted from Febbraro et al. [FRR11].

Visualizations showing solving process

Several approaches exist to visualize the solving process of logic programs in general, but especially for CLP. However, due to the complexity of these programs and the corresponding solving process, the visualizations' scaleability depends strongly on either the number of logical expressions, constraints, predicates, or the number of variables. For instance, some approaches show the concrete propagation and evolution of variables and predicates for small programs on a very detailed level such as LogiChart [ATIY00, AF07, Ada09] (see Figure 2.6), or the DeLPViewer of Escarza et al. [ELCM09]. On the other hand, approaches, like Clavis of König and Schaub [KS13] as well as the approach of Simonis [SCD$^+$00] visualize the solving process more abstractly with better scalability. However, these approaches do not provide an overview of the original logical program and barely show dependencies between the entities of the unprocessed program.

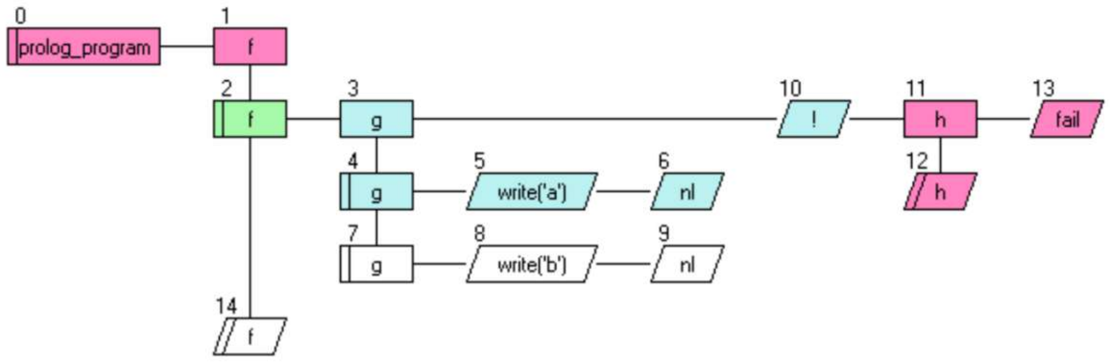


Figure 2.6: LogiChart - reprinted from Adachi et al. [ATIY00, AF07, Ada09].

Visualizations showing answer sets/dependencies between answer sets

To the third category, visualizations belong that either visualize the interpretations of computed answer sets of logic programs or dependencies between those answer sets. Since they do not provide any information on relations between arguments of the input program, they are of minor importance to this thesis. For sake of completeness, however, we want to mention some of these approaches, such as ASPViz [CVBP08], IDPDraw [Wit09], Kara [CVBP08] used by SeaLion [OPT13], and ARVis [ACJ$^+$13].

2.5 Network-Based Visualizations of Set Relationship

There are almost uncountable graph-based approaches and applications for showing relationships between a set of entities. To get an overview, Nobre et al. [NMSL19] categorizes them into *node-link* layouts, *tabular* layouts, and *implicit tree* layouts. According to Nobre et al., *node-link* layouts are the most common graphical representation for graphs and networks. Schulz and Schumann [SS06] further subdivide such layouts into *free* layouts, where the nodes' positions are not restricted, *styled* layouts, where the positions follow a predefined scheme, and *fixed* layouts, where the position is determined by nodes' attributes, such as latitude and longitude. Examples of free layouts are force-directed layouts where a node's position depends only on the links to its neighbors. Styled layouts typically make use of predefined schemes, like grids or axis-parallel and radial arrangements [NMSL19]. Besides, *on-node* and *on-edge* encoding is often used to encode additional information into graph visualizations [NMSL19].

The hair-ball problem mentioned by Schulz and Hurter [SH13] is typical in node-link layouts. When the number of nodes and links between those nodes exceeds a critical sum, the visualization suffers from cluttering and overplotting. The problem especially appears when a set of entities is strongly connected, which results in a very dense graph. An extreme example would be a complete graph. A different interpretation would be that many entities of the set share the same information.

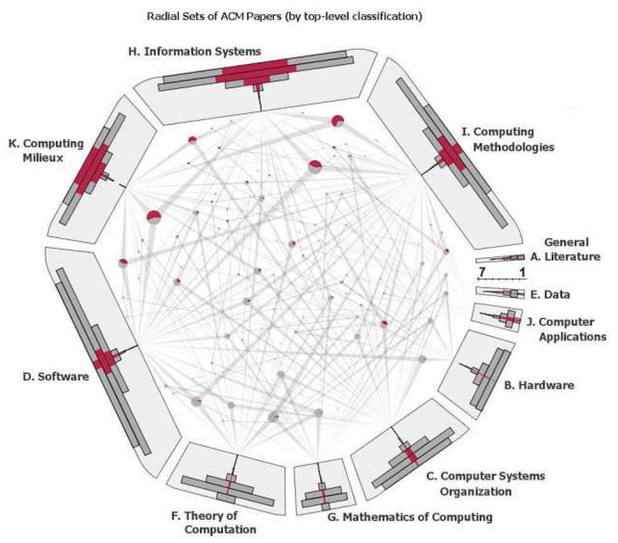
To address this problem, researchers developed various approaches to reduce the clutter by aggregating or filtering either the nodes, the edges, or both. For example, edge bundling [LHT17] and edge splatting [BBW12] are very commonly used techniques. A different approach would be the use of hyperedges. The edges of a completely connected subset of nodes of a graph are removed, a new hypernode is added, and this introduced node is newly connected with the original nodes of the subset [AAMH13] (see Figure 2.7a). Another technique for hyperedges was introduced by Kerren and Jusufi [KJ13]. In their approach, the nodes of the set are arranged in a radial layout, and the hyperedges of at least degree 3 are represented as arcs that enclose the circle (see Figure 2.7b). Nodes that are part of the hyperedge are reprinted on the edge at related arc positions.

Other typical visualizations for showing relationships between radial arranged entities of a set are for example arc diagrams, chord diagrams [HEAE16], non-ribbon chord diagrams [ZBA$^+$20] or tidy tree diagrams [VBW17] (also see Figure 2.10c).

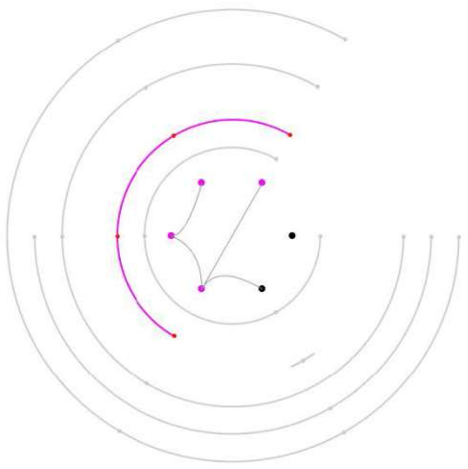
2.6 Hierarchical Data Structure-Based Visualizations

Hierarchical data can be visualized in many different ways. Schulz et al. [Sch11] suggests three design axis to describe such tree visualizations: *dimensionality*, *edge representation*, and *node alignement*. For *dimensionality*, they distinguish between 2D, 3D, and hybrid visualizations. Whereas in explicit *edge representations* the links between the nodes are clearly drawn (see Figure 2.8a), the nodes' connections in implicit representations are shown through related positioning (see Figures 2.8a-2.8d). To conclude the three design axis, the node alignment can be either radial, axis-parallel, or free.

Typical examples for *implicit* edge representations of hierarchical structures, mentioned by Woodburn et al. [SZ00], are icicle plots (see Figure 2.9a), sunburst charts (see Figure 2.9d) [SZ00], sundown charts (see Figure 2.9b), or treemaps (see Figure 2.9c). This list can be extended by circle packing approaches [WWDW06] which is a subcategory of treemaps, and hybrid approaches, like fore-directed tree layouts [Wal08] or phylogenetic

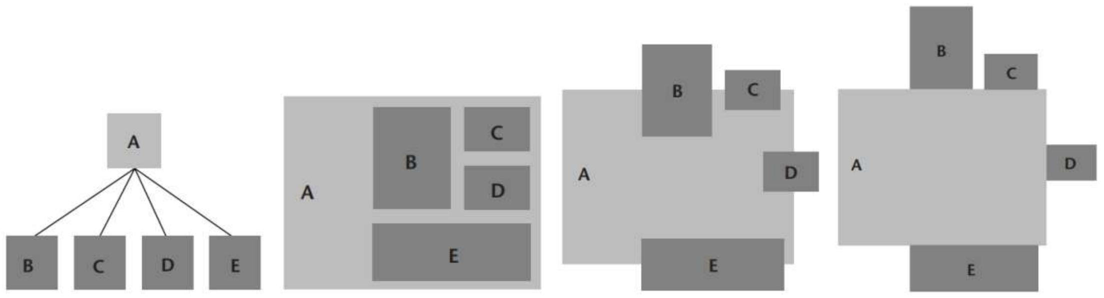


(a) Hyperedges embedded inside of radial arranged nodes with node-encoded bar charts - adapted from Alsallakh et al. [AAMH13].



(b) Hyperedges drawn as attached arcs to radial arranged nodes - adapted from Kerren and Jusufi [KJ13].

Figure 2.7: Radial hypergraph-based set overlap visualizations showing two techniques to represent the hyperedges.



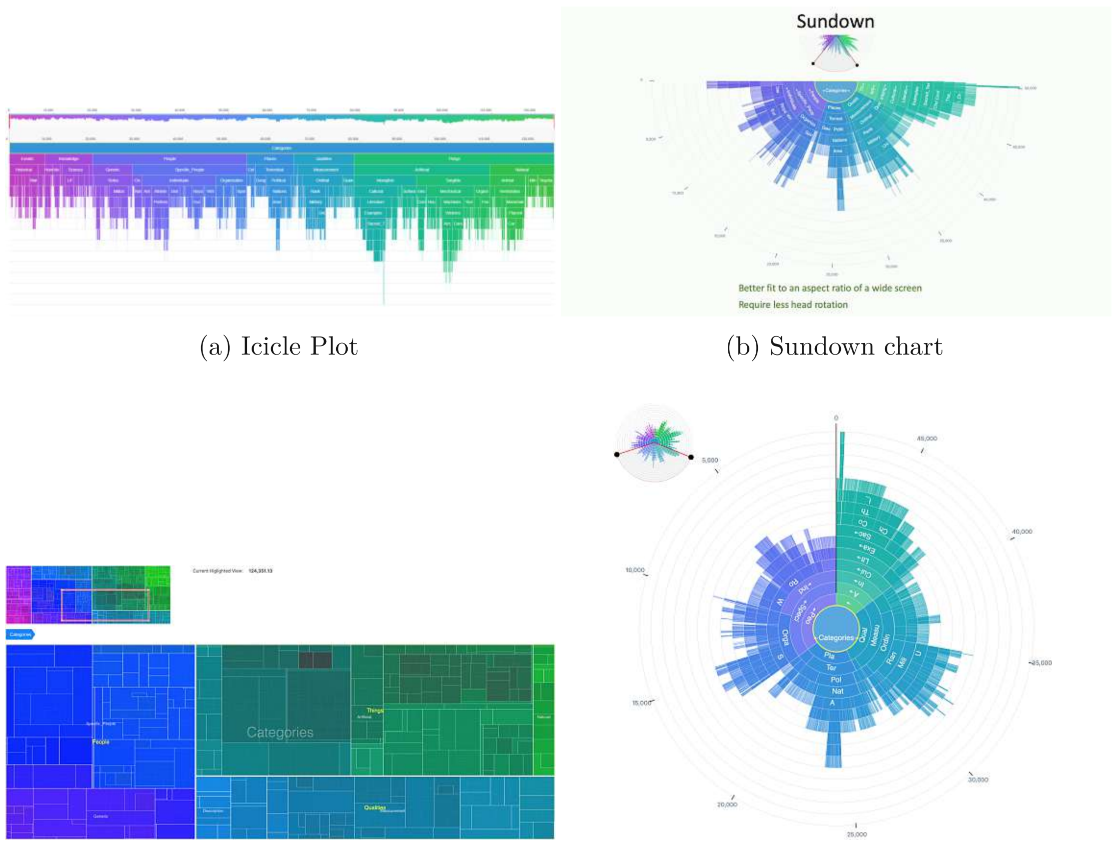
(a) Explicit inclusion (b) Implicit inclusion (c) Implicit overlap (d) Implicit adjacency

Figure 2.8: Four types of edge representations - adapted from Schulz et al. [Sch11].

trees [Nie16]. Examples for *explicit* edge representations are tidy trees[16] [SDS12], also known as dendograms[17].

Most relevant related to this thesis are visualization showing hierarchical structures in graphs. Vehlow et al. [VBW15, VBW17] summarized the state of the art in this field and categorized the visualizations into the approaches *visual node attributes*, *juxtaposed*, *superimposed*, and *embedded* visualizations. Whereas the category of *visual node attributes* is primarily used in flat group structures, many different approaches exist for embedded,

[16]`https://observablehq.com/@d3/tidy-tree` (accessed 2022/08/22)
[17]`https://www.d3-graph-gallery.com/dendrogram` (accessed 2022/08/22)



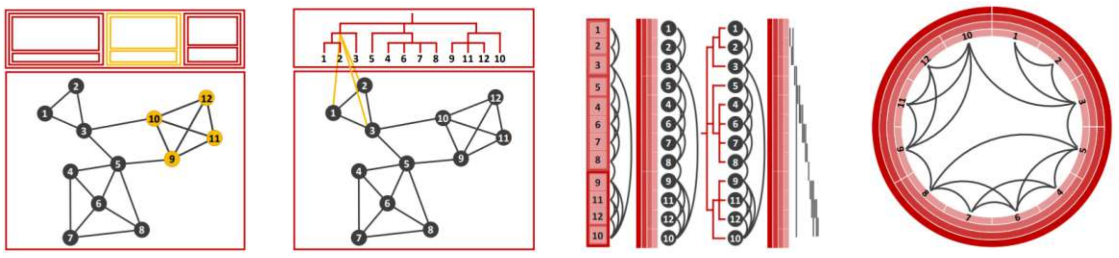
(a) Icicle Plot (b) Sundown chart

(c) Treemap (d) Sunburst chart

Figure 2.9: Implicit edge representations for visualizing hierarchical quantitative data - reprinted from Woodburn et al. [WYM19].

juxtaposed, and superimposed visualizations (seen in the table classifying different techniques by the taxonomy of Vehlow et al.). Because we have a hierarchical structure in our data domain, we focused on differences between mentioned approaches by Vehlow et al. in showing these structures. However, we omitted matrix-based approaches, as they cannot visualize hyperedges providing textual information about shared features of nodes.

For *juxtaposed* visualizations they distinguish between techniques separating the graph and the visualization showing the hierarchical structure respectively the group membership (see Figure 2.10a and 2.10b) and techniques attaching the visual encoding of the hierarchy structure to the nodes of the graph (see Figure 2.10c and 2.10d). In the separated views, either brushing and linking by color highlighting is used (see Figure 2.10a) or visual links between the views to show the group membership (see Figure 2.10b). For attached approaches, the nodes are arranged along a horizontal line (see Figure 2.10c) or in a circle building a radial layout (see Figure 2.10d). Therefore, the lowest level in the hierarchy is either presented in the same way as the attached encoding of the hierarchy or as separate glyphs representing the nodes.

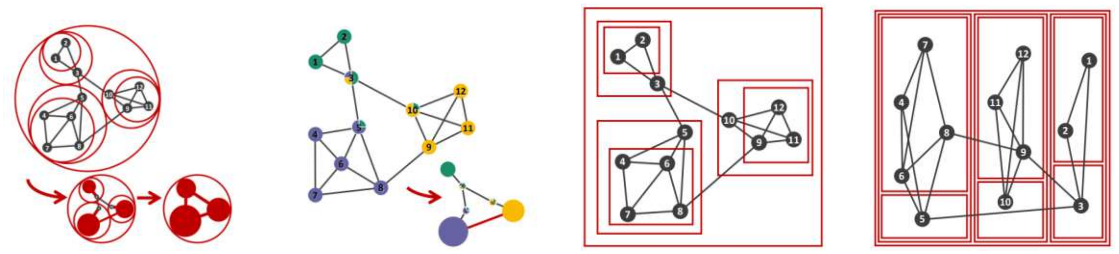


(a) Separate — brushing and linking (b) Separate — visual links (c) Attached — node-link (d) Attached — radial

Figure 2.10: Juxtaposed visualizations of disjoint hierarchical groups - adapted from Vehlow et al. [VBW15, VBW17].

In *embedded* views for disjoint groups, nodes are aggregated according to their group membership and hierarchy level (see Figure 2.11a). Aggregated nodes are again represented as circles where the number of containing nodes determines the size of the circle. If the groups overlap, each aggregation node is colored differently, and additional connection nodes are introduced in the form of pie charts. The slices of the pie charts show the division of affiliations to the groups and are colored according to the group (see Figure 2.11b).

For superimposed hierarchical structures, either nested contours (see Figure 2.11c) or screen space partitioning is used (see Figure 2.11d). Especially the screen space partitioning is very similar to the representation of tree maps, as shown in Figure 2.9c.



(a) Embedded disjoint hierarchical (b) Embedded overlapping hierarchical (c) Superimposed disjoint hierarchical using nested contours (d) Superimposed disjoint hierarchical using screen space partitioning

Figure 2.11: Embedded and superimposed visualizations for hierarchical group structures - adapted from Vehlow et al. [VBW15, VBW17].

2.7 Summary

To conclude, none of the existing recommendation systems enable researchers to test new design theories without spending great effort in learning and understanding the system (see Section 2.1). The used rules and theories are opaque and poorly referenced. Depending on the underlying system, the recommendation engine either relies on the quality of previous visualizations or on expensively created rules. The recommendation processes of these systems mostly resemble a black box where made decisions are hard to retrace.

Draco addresses these problems by simplifying the hand-grafted rules using a constraint-based logic programming notation and combining these rules with trained weights. The resulting hybrid-recommendation engine is less complex, allows real-time manipulation of the knowledge-base, and enables to retrace made decisions through printing violated design rules.

However, Draco still requires sufficient understanding of the used logic programming notation and lacks adding dynamic meta-data to the rules. Considering the different existing grammars and parser generators of the ASP notation, it is possible to split up the knowledge-base of Draco into its units to develop tools helping in understanding and maintainig it (see Sections 2.2 and 2.3). Yet, these ASP syntax grammars are either incomplete, written in a very language-specific notation, or are incorporating syntactical errors. Consequently, the grammars can not directly be used by modern parser generators without any modifications.

Besides, several attempts exist to find proper visualizations in context of logic programming. They aim to enhance the understanding of similar logic programming problems and subsequent enumeration processes (see Section 2.4). However, these visualizations represent the dependencies between the units of the logic programs either in a too detailed manner, address a different aspect and problem field like IDEs, or are showing the enumeration process or dependencies and similarities between answer sets. The enumeration process on the very low level is, again, too detailed for generating the overview and understanding of the knowledge-base in which we are interested. The majority of visualizations in the context of logic programming simply address a different domain or task to complete. Furthermore, none of the existing graph-based and hierarchical-based visualizations (see Sections 2.5 and 2.6) can be directly applied and used to visualize design rules respectively constraints to show their feature-based connections.

It can be concluded that several different domains must be addressed to enable developers to build a new tool that helps to understand, maintain, and enhance a hybrid-based recommendation engine such as Draco. These domains range from interpretation of the encoded design rules and their parsing mechanism to proper visualizations of the underlying knowledge-base.

CHAPTER 3

Data Processing and Visualization Approach

Any data visualization requires several steps, from the raw data to the final interactive visualization. These steps are described roughly by the Information Visualization (InfoVis) reference model of Card et al. [Car99] in Figure 3.1. It mainly consists of three viewing angles: the data processing perspective, the visual form perspective, and the users' perspective to feed back into the visualization pipeline by interaction to solve particular tasks. According to Card, the raw data has to be transformed into data tables having an appropriate data structure and organization for the subsequent visual mapping of the data to visual structures. View transformations describe the change of graphical parameters of the visualization to the users' needs to obtain different views and perspectives of the underlying data.

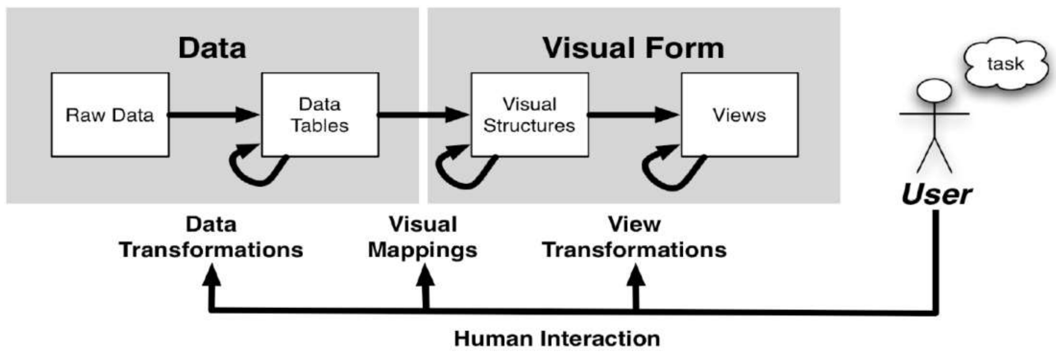


Figure 3.1: Information visualization reference model by Card et al. [Car99].

Every visualization has to serve a certain purpose and must not be an end in itself. Hence, three elementary components have to be taken into account in the context of Draco:

- **Data:** Design rules are expressed as logical constraints in the ASP notation, including variables, predicates, numbers, terminals, and other logical language-specific features. Rules having assigned weights form soft constraints, and rules with no weight belong to hard constraints.
- **Users:** The target users for our visualization respectively tool are visualization experts and persons in the research community of automated visualization recommendation and design.
- **Tasks:** Starting point of every design decision is to target the following tasks by the elaborated linked visualization:
  - T1: Analyze the distribution of costs
  - T2: Identify shared syntax features of multiple constraints
  - T3: Find violations of recommendations and relate them to the complete list of constraints
  - T4: Compare violations of multiple recommendations
  - T5: Identify recommendations sharing same violations of syntax features or constraints

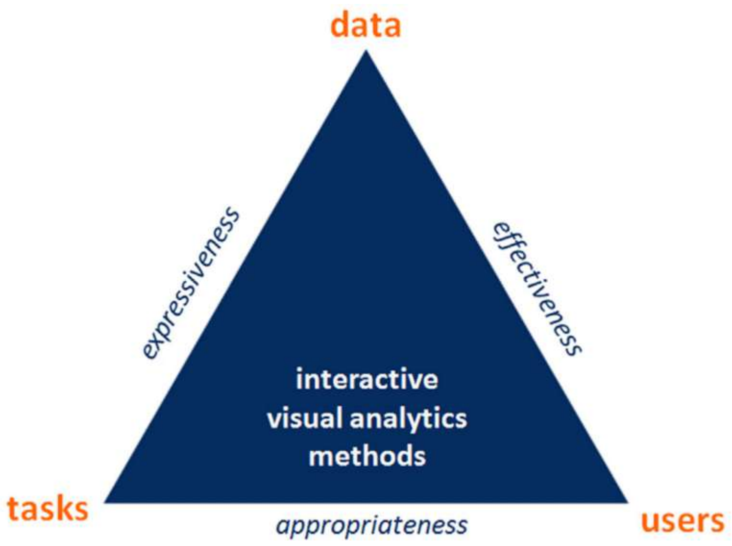


Figure 3.2: Design triangle: data - user - tasks - reprinted from Miksch and Aigner [MA14].

Based on these specifications, Miksch and Aigner [MA14] introduced the 'Design Triangle' framework to show qualitative dependency criteria between each of the three components (see Figure 3.2). Every visualization should be analyzed regarding its *expressiveness*, *effectiveness*, and *appropriateness*. Mackinlay [Mac86] stated that a visualization is considered to be expressive if and only if all relevant information is expressed by the visualization to achieve a certain goal. He further declares that a visualization is effective if the used graphical language respectively encoding addresses the capabilities of the output medium and the human visual system and best exploits these limitations. The last criterion of the design triangle describes a trade-off between the effort required to create the visual representation and the benefits yielded by it, as stated by Schumann and Müller [SM00].

Therefore, we conducted a problem and requirements analysis, analyzing Draco's current knowledge-base (the raw data) and its applied context. The goal is to identify the problems that users of Draco encounter when trying to solve respective tasks and, consequently, to derive a set of requirements. These requirements are necessary to properly build and subsequently evaluate the data processing and visualization procedure for the knowledge-base of Draco.

3.1 Problem and Requirements Analysis

To organize the knowledge-base and make it more readable, Draco's logical expressions have several peculiarities that are not mandatory requested by ASP. First, expressions sharing the same goal are grouped in sections. Next, rules called hard and soft constraints in Draco share the predicates 'hard' and 'soft' in the head of the rule. The first argument of each soft and hard predicate represents the identifier of the constraint. This identifier is a connected list of textual categories representing visual encodings or data transformations. Besides, rules, facts, and constraints do not have an identifier in ASP in general. Therefore, the first argument of the hard and soft constraints in Draco is used as an identifier to assign weights to the soft constraints and for maintainability reasons of the knowledge-base. However, the identifiers are not unique to depict logical OR constructions between multiple constraints. Hence, a hard or soft constraint is violated if only one of the multiple definitions is violated.

Whereas the predicate names are always written in long form, the variables are shortened to a maximum of three letters in Draco. This notation was chosen to increase the readability of the constraints. Furthermore, in the knowledge-base only single-line comments are used. Sections grouping the same intent are described in the head by comments of the form "*% == X ==*," where X is the title of the section. Each hard and soft constraint is described by a comment of the form "*% @constraint Y*," where Y is the description of the constraint.

The following problems can be derived from the Draco-specific notations encoded into its knowledge-base:

1 Syntax dictionary and overview
Although the notations of the constraints, predicates, and variables – the arguments; are well-considered, ambiguities arise for users who were not involved in

the development phase. First, it is not visible at first glance what predicates and variables are used in the knowledge-base and in which context they are used. There is no syntax dictionary or advanced documentation besides the single-line comments. Even though the knowledge-base developers did their best to find appropriate and expressive names for the predicates and variables, it is not always clear what these arguments are intended for. Most of the time, this purpose can only be derived by looking at the different contexts where this argument is used. This method of argument understanding is tough for variables since the most common forms are variables with only one or two characters. For example, the variable E indicates the name *encoding*. However, this derivation might be significantly more difficult for users dealing with this syntax for the first time.

Furthermore, the representation of the knowledge-base leads to difficulties in getting an overview of where different features respectively arguments are used. This overview knowledge is essential to comprehend if different arguments are used in the right way and at appropriate positions.

2 Overlaps of constraints
Currently, overlaps of constraints can only be derived through pairwise comparing the constraints. Therefore, it is difficult to perceive how a collection of constraints differ and to which degree. Comparing constraints is especially interesting when similar or faulty constraints should be detected.

3 Category overview
The identifiers of the constraints are already hierarchically structured in naming. Moreover, the constraints sharing similar hierarchical structures are grouped in the knowledge-base. This naming and structuring approach to the constraints is intended to increase their readability and maintainability. However, the constraints are statically defined in a string format and can not be rearranged, filtered, grouped, or sorted. This fact applies to both the content of the rules and the categories. Hence, it is not possible to see at a glance which categories and hierarchies of these categories exist. Such information would be especially interesting for perceiving a "big picture" of the knowledge-base and to enable users making a statement about which categories of visual encoding are covered and which are not.

4 Descriptions and sources
One of the many advantages of ASP is using comments in the syntax. The comments enable developers to describe the logical expressions at related positions. In Draco, the rules and constraints are primarily described in the comments by phrases in the form of "prefer to," "prefer not to," "should," and "should not." However, advanced explanations of the rules are missing. No statement is made about why something should be preferred or why not. Additionally, it is unclear where the rules come from, who defined them and how they have been verified to fulfill their purpose.

Although such additional information could be added via the comments, it would compromise the readability of the knowledge-base as the expressions lose prominence in the code editor. Furthermore, advanced meta-data can not be added because the logical expressions and comments are a sequence of textual lines.

5 **Readability**
The advantage that comments enable to describe the logical expressions is at the expense of readability. The comments can not be hidden, and the constraints can not be rearranged to get related positioning of constraints associated with similar hierarchical categories. Hence, it is hard to obtain an overview of the constraints and get insights into which syntax features are used and where they are used.

Hence, we can derive that neither the data format of the logical expressions nor their representation are optimal to comprehend the knowledge-base of Draco and subsequently maintain it. One possible way to improve this data format and representation would be to extract the essential information and reorganize it as desired. For such an information extraction process, the ASP parser of Draco could be used to produce an Abstract Syntax Tree (AST) of the knowledge-base with subsequent extraction of desired information. As the AST of the parser can not be easily accessed, the developers of Draco decided to use REGEX to extract the information. This procedure is sufficient to separate the constraints from the comments, extract constraints' identifiers, and map the constraints with the solver's results – as accomplished in Draco. However, it is unsuitable for extended analysis of the knowledge-base and advanced interactions and visualization techniques as the amount of extractable information is limited to the specificity of REGEX.

In addition to derived representation and maintainability problems of the knowledge-base of Draco, we analyzed the recommendation process regarding its traceability. To recommend visualizations, the ASP solver Clingo [GKK$^+$08a, GKK$^+$08b] reasons over the facts, rules, and constraints and prints valid answer sets to the query. One valid answer set is a combination of facts concerning visualization design variables and headers of violated soft constraints – so-called *witnesses* [MWN$^+$19]. Whereas the resulting facts can be directly used to construct a visualization, the soft constraints must be mapped back to the original expressions in the knowledge-base. Therefore, Draco's identifiers of the soft constraints in the knowledge-base are extracted with the mentioned REGEX approach and filtered by the violated soft constraints in the answer set. However, to compare the violations of different recommended visualizations, the user must inspect the visualizations' violations one after the other and can not compare them directly. Since the violations are also unsorted, it is a demanding task to obtain an overview of the violations and comprehend the ranking of the recommended visualizations.

3.1.1 Requirements Concerning Data Extraction

A different data extraction methodology has to be examined to overcome the identified limitations of the current extraction approach of knowledge-base features. To fulfill typical quality metrics as mentioned by Coulin et al. [CDMP19], we formulate the following general requirements concerning the data extraction process:

- **Extensibility:** The language features of ASP are regularly adapted and expanded to incorporate further developments in modern ASP solvers. Therefore, the supported language features must be easily expandable to enable users to explore new relationships between potentially changed formulations of design constraints or between existing constraints and newly incorporated ones.
- **Maintainability:** The grammar and the parser generator should be encapsulated to different modules to ensure maintainability of the feature extraction process. Therefore, both the grammar and the parser generator should be easily replaceable.
- **Simplicity, Understandability:** The architecture of the data extraction process should be as simple as possible to ensure ease of understanding and, consequently, its maintainability and reliability. Therefore, the system should be easier to communicate to new developers, such as visualization experts.
- **Reusability**: Extracted data should be reusable and therefore, stored in a permanent data storage. This approach is expected to reduce the time necessary for the initialization phase of the visualization and to ensure real-time data manipulation of the data in terms of filtering, sorting, grouping, and aggregating.
- **Reproducibility:** Repetition of extraction process of unchanged knowledge-base should lead to identical extracted features.

Besides, more specific requirements and criteria for the selection of ASP grammar and associated parser generator should be considered within the design of the data extraction process:

- **Completeness of ASP grammar**
  The grammar should be capable of describing all syntax features incorporated into Draco's soft and hard constraints (a minimum fulfillment requirement is the ASP-Core 2 standard). The existence of typographical errors or missing syntax features in the grammar must be minimal to ensure ease and direct use by the parser generator with minimal adaptation required.
- **Support of AST creation by parser generator**
  The parser generator must contain the functionality of creating and returning a recursive searchable AST preserving the grammar structure and identifiers for parsed ASP code.
- **Support of available ASP grammar by parser generator**
  The parser generator must support the grammar without extensive adjustments to the grammar.
- **Applicability of parser generator**
  The parser generator should be selected regarding the ease of integration into web-based frameworks to guarantee maintainability.
- **Performance of parser generator**
  The parser generator should be selected by taking into account the time necessary

to create the parser instance and subsequently to parse the ASP code. The parser should be capable of parsing the complete knowledge-base of Draco within a maximum of one second to guarantee real-time parsing in case of changes to knowledge-base during runtime.

3.1.2 Requirements Concerning Data Visualization

An enhanced representation respectively visualization of the knowledge-base should be elaborated to overcome the limitations of the current representation. The hard and soft constraints of the knowledge-base should be visualized in combination with the newly extracted features of the intended extraction process. Additionally, the recommendation process concerning constraints' violations should be made traceable by connecting the recommendations with the visualization limited to the constraints of the knowledge-base. Therefore, both tracing directions of the recommendation process should be supported. On the one hand, the respective violated constraints of selected recommendations should be highlighted. On the other hand, the knowledge-base constraints' influence on the recommendation results should be made visible.

Consequently, we formulate the following general quality metrics and corresponding requirements to guarantee their compliance in the design process:

- **Interactivity:** It is impossible to show every perspective of the data in one view at once. Therefore, high interactivity with the visualized data is required to handle complexity, gain the necessary insight, and show and extract the relevant information.
- **Intuitiveness:** Both graphical representations and interaction possibilities should be understandable by expert users having background knowledge in visualization design. To decrease the learning curve, complex commands, textual queries, and misleading representations should be avoided.
- **Accessibility:** The visualization should be easily accessible without any complex installation and setup process. Therefore, the visualization should be embedded into an Operating System (OS) independent web-based environment accessible through any modern web browser.
- **Performance:** The key to smooth interactivity is performance of the underlying algorithms and data processing strategies. Satisfactory performance must be guaranteed for loading and processing the data, initializing the visualization, and later manipulation. Changes to data and the visual appearance of the visualization through user interaction must result in immediate response times for visual changes. With increasing data entries, the performance may only decrease marginally without restricting the real-time interactivity.

3.2 Data Extraction and Processing

In this section, we theoretically elaborate on the data extraction and processing necessary to tackle the design constraints of Draco visually. As a result, we propose a data extraction architecture that fulfills the requirements defined in the previous section.

3.2.1 Feature Extraction Strategy

To find the most appropriate strategy for our extraction task, we analyzed the existing techniques in Draco according to their applicability and feasibility in parsing and extracting the features of interest. The idea is to re-use the logic of the current recommendation engine as much as possible to keep new dependencies and redundant logic to a minimum. As a result of this analysis, two extraction and parsing techniques could be identified in the whole ecosystem of the recommendation system behind Draco:

- **Extraction using REGEX**
  Draco uses REGEX to solve two distinct tasks. First, it validates if a selection of ASP code is a collection of valid hard and soft constraints (according Draco's specific notation style). Second, it splits the validated constraints and extract several features per constraint, such as documentation (constraint's comment), the type (hard or soft), the identifier, the constraint's weight, and the whole constraint itself. This approach enables Draco to quickly and easily validate and extract certain knowledge-base features without analyzing the ASP code on a syntax grammar-based level.

  However, this approach requires carefully designed REGEX expressions and has to be adapted for every change to Draco's specific notation style of the constraints. As a consequence, the complex REGEX expressions are prone to errors. Like most programming languages, ASP is extending CFGs and thus, can be classified to the Chomsky hierarchy 2 [Cho56]. Since REGEX applies to Chomsky hierarchy 3, it is not expressive enough to describe the complete syntax of ASP. Consequently, this approach is limited to consistent meta-data features (such as the extracted ones by Draco) and prohibits extended analysis of the knowledge-base beyond the hard and soft constraints.

- **Extraction using grammar parsing**
  To ground and reason over knowledge-base, Draco uses Gringo and Clingo [GKK$^+$08a, GKK$^+$08b]. Gringo integrates bison[1], a parser generator based on a CFG, to generate an ASP parser and parse, respectively, validate the ASP program provided by Draco. Additionally, an AST module is implemented into Clingo to classify and subsequently access certain attributes and features of the parsed program. This module can be directly used to parse a program and extract an AST by including Clingo in an external project.

[1]https://www.gnu.org/software/bison (accessed 2022/08/22)

However, the module can not be accessed via the Command-Line Interface (CLI). The CLI API of Clingo does not provide any argument which would allow returning an AST when running Clingo. The same applies to the output of bison. Since Draco runs Clingo over the CLI, it has no access to the parsed program. Nevertheless, Draco could be changed to include and use Clingo directly. Then it would have access to these AST-based parsing functions. Still, it is not guaranteed that the AST is accessible in the way necessary. Furthermore, due to the issue entry of the Clingo project[2], the AST module is still experimental and thus, unstable concerning further developments in future. This finding also explains why Draco introduced its own parsing strategy via REGEX to extract certain desired syntax features. Moreover, directly including Clingo into Draco would increase the coupling and, therefore, negatively influence its maintainability. Besides, it is not our goal to change Draco to fit our requirements. Draco should be developed and enhanced independently to our approach.

Hence, we can conclude that both existing parsing respectively syntax extraction implementations are not suitable or accessible to extract the knowledge-base features we are interested in. However, according to the existing strategies, generating a parser using a CFG looks most promising to receive a data processing unit that delivers the desired results.

Therefore, we developed a similar approach to Clingo using a parser generator capable of returning an AST of the parsed knowledge-base. In the following sections, we argue our decisions made concerning the selection of an ASP grammar and the corresponding parser generator and take a closer look into the proposed architecture of our parsing strategy.

3.2.2 ASP Grammar and Parser Generator

For selecting a suitable combination of an ASP grammar and a parser generator, we analyzed different available grammars and web-technology-based parser generators during the development phase of the prototype. The found grammars and generators are described in sections 2.2 and 2.3. Because of the incomplete and partially faulty definitions of the mentioned grammars in scientific publications [GKK+08b, CIR11, CFG+12] and the very language specific notations found in ASP solver documentation, such as Potassco[3] or in the Tweety Project[4], we looked for a grammar where least modifications to the notation are necessary. A grammar containing all the specifics of the described syntax features by Gringo [GKK+08b] would be ideal since Gringo is used for parsing Draco's knowledge-base and ground over it. However, their suggested EBNF notation of the grammar follows no common notation style, which is accepted by popular web-based parser generators. Moreover, not only the implicit grammar implementation of Gringo

[2] `https://github.com/potassco/clingo/issues/171` (accessed 2022/08/22)
[3] `https://potassco.org/clingo/python-api/5.4/ast` (accessed 2022/08/22)
[4] `http://tweetyproject.org/api/1.17/net/sf/tweety/lp/asp/parser/ASPCore2Parser.html` (accessed 2022/08/22)

in its source code is too language specific to C++, but also its documentation on the Potassco's website.

As a consequence that we did not find a suitable grammar notation of ASP integrating the syntax features of Gringo and simultaneously, fulfilling the derived requirements for a grammar noted in Section 3.1.1, we decided to adapt the grammar of the ASP-Core 2 standard to our needs. This approach is supported and justified, as many introduced syntax features of Gringo are not required to be able to parse the main part of the knowledge-base (rules, facts, and constraints). We will discuss those limitations of ASP-Core 2 grammar in relation to the knowledge-base in more detail in the evaluation chapter 5.

The ASP-Core 2 grammar is specified in a custom EBNF notation mixing the ISO/IEC-14977:1996[5] standard for EBNF with the BNF typical angle brackets for non-terminals. However, EBNF can not only be defined in the ISO/IEC standard, but also following the World Wide Web Consortium (W3C)[6] notation standard. Wheeler [Whe20] compares these two standards and concludes that the W3C standard should be preferred over the ISO/IEC standard due to its several weaknesses. One of the most important reasons to not use the standard, according to Wheeler [Whe20] is that it is not generally accepted by its related community. Furthermore, it is not even used in all ISO language standards [Whe20]. Consequently, many grammar parsers and generators, especially those we analyzed in this thesis, do not accept grammar following the ISO/IEC-14977 standard.

There are many other forms for defining an EBNF syntax as described in Chapter 2. However, by analyzing these custom notations, we concluded that these modified versions of the BNF or EBNF notations are either too specific to the respective parser generator or are not as easy to comprehend as the W3C standard of EBNF. Therefore, we decided to transform the ASP-Core 2 grammar notation from the ISO/IEC-14977 standard to the more solid and accepted W3C standard. This remodeled version enabled us to choose between a broader selection of parser generators.

For this transformation, the following adaptions to the ASP-Core 2 grammar notation are necessary:

- Define all lexical values in REGEX notation.
- Remove angle brackets at non-terminals.
- Add surrounding quotation marks to all terminals.
- Replace optional expressions of the form "[...]" with the postfix question mark character "?" of the REGEX notation having the identical meaning .
- Eliminate left recursions by using the occurrence postfix characters "+" (1...n) and "*" (0...n) of the REGEX notation, or by introducing new non-terminals.
- Add a lexical value for whitespaces.

[5] `https://www.iso.org/standard/26153.html` (accessed 2022/08/22)
[6] `https://www.w3.org/TR/xml/#sec-notation` (accessed 2022/08/22)

- Incorporate definitions for single-line and multi-line comments as well as whitespace definition and new-line definition wherever necessary and valid.

The resulting EBNF-based grammar of ASP having the ASP-Core 2 input syntax standard can subsequently be easily used by a parser generator accepting this W3C standard. However, concerning the list of relevant web-based parser generators as outlined in Chapter 2, there is only one parser generator fulfilling our requirements stated in Section 3.1.1. The EBNF parser generator[7] published by Mendez is capable of parsing W3C-based syntax and generates an easy processable AST. Every node or leaf of the resulting tree is a reference to the corresponding terminal definition in the grammar. Those hierarchical references enable to search the AST for certain desired features as described in the following section.

3.2.3 Extraction of Features

For further development and extension of the knowledge-base it is crucial to get insights into the existing defined visualization design facts, rules, and constraints over these facts. To gain this degree of understanding and overview of the knowledge-base, it is essential to get an overview of the used variables, predicates, and other syntax features. These features can be easily extracted from the generated AST of the knowledge-base. Exemplary, we show the generated AST in Figure 3.3 of the constraint with the identifier `bin_high` in Listing 3.1.

Listing 3.1: A soft constraint with the identifier `bin_high` which has a negative impact to a recommendation's ranking when one of its encodings contains more than 12 bins.

```
% @constraint Prefer binning with at most 12 buckets.
soft(bin_high,E) :- bin(E,B), B > 12.
```

One way to extract these features would be to define a method for every feature we are interested in. These methods search the resulting syntax tree for the feature and return all found entities. However, the list of features we are interested in might be changing or extending over time. Therefore, the more advanced strategy is to define a generic search function for any desired feature available in the grammar. This list of features is stored separately and hence, decoupled from the extraction process.

The search algorithm of the generic function, therefore, has three parameters (see pseudo code in Algorithm 3.1): The generated AST of the knowledge-base from the parser, the identifier of the desired feature, and a temporary feature list which is extended by newly found ones. In a recursive process, the method steps into the different branches of the AST. Then it compares the reference to the grammar of the tree node with the identifier of the desired feature. If they are equal and the feature is not already present in the list

[7]`https://www.npmjs.com/package/ebnf` (accessed 2022/08/22)

```
program
  statement
    comment Text=% @constraint Prefer binning with at most 12 buckets.
  statement
    rule
      head
        disjunction
          classical_literal
            predicate Text=soft
            terms
              term Text=bin_high
              terms
                term Text=E
      bodies
        body
          naf_literal
            classical_literal
              predicate Text=bin
              terms
                term Text=E
                terms
                  term Text=B
        body
          naf_literal
            builtin_atom
              term Text=B
              binop Text=>
              term Text=12
```

Figure 3.3: Generated AST of the constraint in Listing 3.1. It is a hierarchically decomposed structure of the test code where the identifiers in each level are described by the grammar.

of found features, the list is extended by this feature. The result of the search algorithm is a list of found features.

The crucial advantage of this generic approach is that if the grammar of ASP changes, like renaming terminal definitions, only the list of desired features must be modified accordingly. Hence, no code adaptions are necessary, and the modifications can be done in runtime.

3.2.4 Interpretation and Processing of Constraints' Categories

The textual encoded categories of the constraints' identifiers are structured hierarchically and separated through underlines (see Listing 3.2). The hierarchy levels decrease from left to right, where the category *value* represents the highest hierarchy level and x the lowest and most specific level. Thereby, the categories in the hierarchies do not depict formal categorizations of visual variables or encoding but rather informal hierarchical descriptions of the constraints' content. Therefore, the descriptions are chosen so that the descriptions map to similar hierarchy levels and are most descriptive and expressive in minimal words. Although 'category' is not the most accurate terminology for these

Algorithm 3.1: Recursive feature extraction from the generated syntax tree

```
1  Function extractFeatures(ast, identifier, features)
       Input: AST of knowledge-base and desired feature's identifier
       Output: List of found features
2      if $ast.identifier != identifer$ then
3          $n \leftarrow ast.children.length$;
4          for $i = 0$ to $n$ do
5              $child \leftarrow ast.children[i]$;
6              $extractFeatures(child, identifier, features)$;
7          end
8      else if $features$ does not already contain $identifier$ then
9          $features \leftarrow features + ast$
10     return $features$
11 end
```

hierarchically structured descriptions, we remain using this term. We expect that treating the identifiers as hierarchical categories best supports the user to distinguish between the constraints and explore the knowledge-base efficiently.

An example of such a hierarchical description of the constraint is shown by Listing 3.2. It declares that the content of the constraint defines restrictions concerning the visualization recommendation task, the quantitative data type, and a visual channel in the visualization. To be more precise, without looking at the content of the constraint, we can derive from the identifier that the constraint is violated if we have a value task and continuous data points on the x-axis.

Listing 3.2: Example of a soft constraint with three hierarchy levels in the identifier – task *value*, quantitative data type *continuous*, and channel *x*.

```
% @constraint Continuous x for value tasks.
soft(value_continuous_x,E) :- task(value), channel(E,x),
                              continuous(E), enc_interesting(E).
```

To visualize such hierarchical structured descriptions, the categorical-based description must be stored in a hierarchical data structure. Therefore, the categories of the constraints are extracted using the REGEX extraction approach of Draco. The resulting identifier of the constraint is then split by the underline character. Next, the constraints are sorted alphabetically across the hierarchical categories. The sorting is accomplished by first sorting them in the first level, then in the second, up to the lowest hierarchy level. This sorting procedure is required to aggregate the categories in the different levels in the last processing step of the constraints' category extraction.

3.2.5 Persistent Data Storage

As we do not want to newly extract all the information necessary to visualize the constraints of Draco's knowledge-base every time we draw or manipulate the visualization, the corresponding data must be stored in a persistent database. The database type and location are, however, not decisive. They can be selected due to personal preference. The only restriction is the availability of the respective database in the system's architecture. Of course, the database should be selected regarding its performance, as well as ease of integration and data access. In the chapter implementation 4 of the system's prototype, we argue our selection concerning the database and its structure.

3.2.6 The Final Data Extraction Architecture

The feature extraction architecture aims to reduce the external dependencies and necessary parsing and data processing logic to a minimum. More decisive than maximum performance is the maintainability and understandability of the grammar and the corresponding parsing and extraction operations. Therefore, the architecture consists of a few easy-to-understand units (see Figure 3.4). The pipeline is divided into two parallel strands. One strand extracts and stores the currently unknown features from the knowledge-base by Draco, and the second one extracts the hard and soft constraints. Then, the results of the two strands are merged and stored in the database.

In the first pipeline strand, the parser generator takes the ASP grammar as input and generates a parsing instance. This instance then parses the knowledge-base and generates an AST of the given ASP program in the second step. In the third unit, the recursive method extracts the features of interest (defined in a configuration file) from the generated AST.

The second pipeline strand extracts the hard and soft constraints and the corresponding weights with the same REGEX extraction approach applied by Draco. The reason, therefore, is that the desired constraints are already annotated in Draco, which considerably simplifies their classification and extraction process. Besides, the desired information follows a strict structure introduced by Draco, which enables to form simple REGEX expressions. By contrast, filtering these constraints from the first pipeline strand's resulting AST would require additional logic. The hierarchy can be easily computed from the resulting constraints by processing the constraints' identifiers resulting from the REGEX extraction approach.

The last step unites the two strands and connects the constraints based on the extracted features. The resulting hypergraph-based data is stored in a database together with the features, the constraints, and the hierarchy of the constraints. Whenever the knowledge-base, the grammar, or the list of desired features changes, the corresponding steps have to be repeatedly executed, including their subsequent pipeline steps.

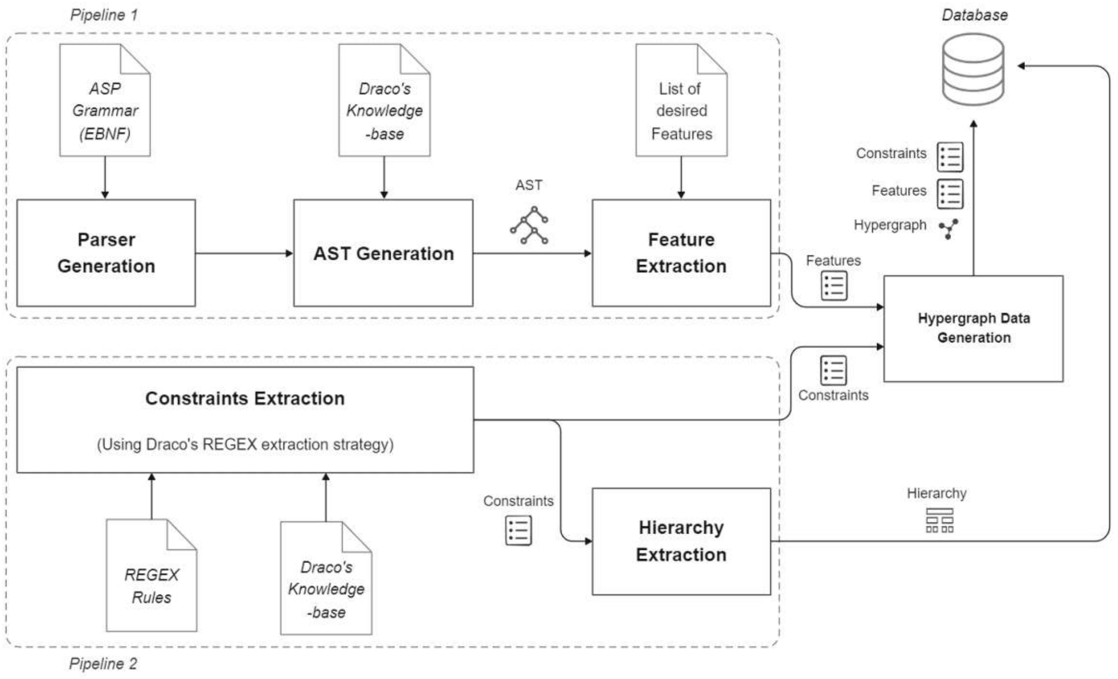


Figure 3.4: Data extraction architecture consisting of two pipelines. The first pipeline extracts the features of Draco's knowledge-base and the second pipeline the constraints, their weights, and the constraints' identifier hierarchy. A hypergraph data generation module processes the features and the constraints and creates the necessary graph-based structure for visualizing a hypergraph of the constraints' shared features.

3.3 Visualization and Interaction Design

This section elaborates on the visualization and interaction design for Draco's soft and hard constraints. We argue the taken design and interaction choices and discuss associated advantages and disadvantages. The final design in Section 3.4 puts together the different aspects and approaches, which are then implemented by the prototype described in the subsequent chapter 4.

3.3.1 Visual Design

An approach to designing a visualization for an unknown dataset is to iteratively try out different visual mappings and encodings to get to the desired result. First, we visualized all the constraints without any order in a free layout design, only showing the connections between the constraints. This exploratory approach gave us an initial overview of the dataset we created in the previous section.

As a starting point, we interpreted our dataset as graph-based and applied an appropriate visual mapping to it. In this approach, every constraint represents a node, and every feature-based connection between these nodes represents an edge. However, this kind of edge-mapping resulted in a very dense graph due to the number of shared features. So solve this issue, we introduced so-called hyperedges to reduce the number of edges. For every hyperedge, a new node is created. This node represents a feature and is connected to all the nodes sharing this feature. Using this new graph structure reduces the number of `e` edges for `n` nodes from `O(n*(n-1)/2)` edges to just `O(n)` edges. Besides, node-based color encoding is used to show the associated weight of the constraints. This coloring of the nodes gives a first impression of the incorporated weights in Draco's design constraints.

The result of this first iteration is a hypergraph seen in Figure 3.5. It is a free node-link layout without any ordering or attribute-based positioning of the nodes and edges. To show violated constraints of a recommended visualization of Draco in the left view, the affected nodes and edges are visually highlighted in the right view.

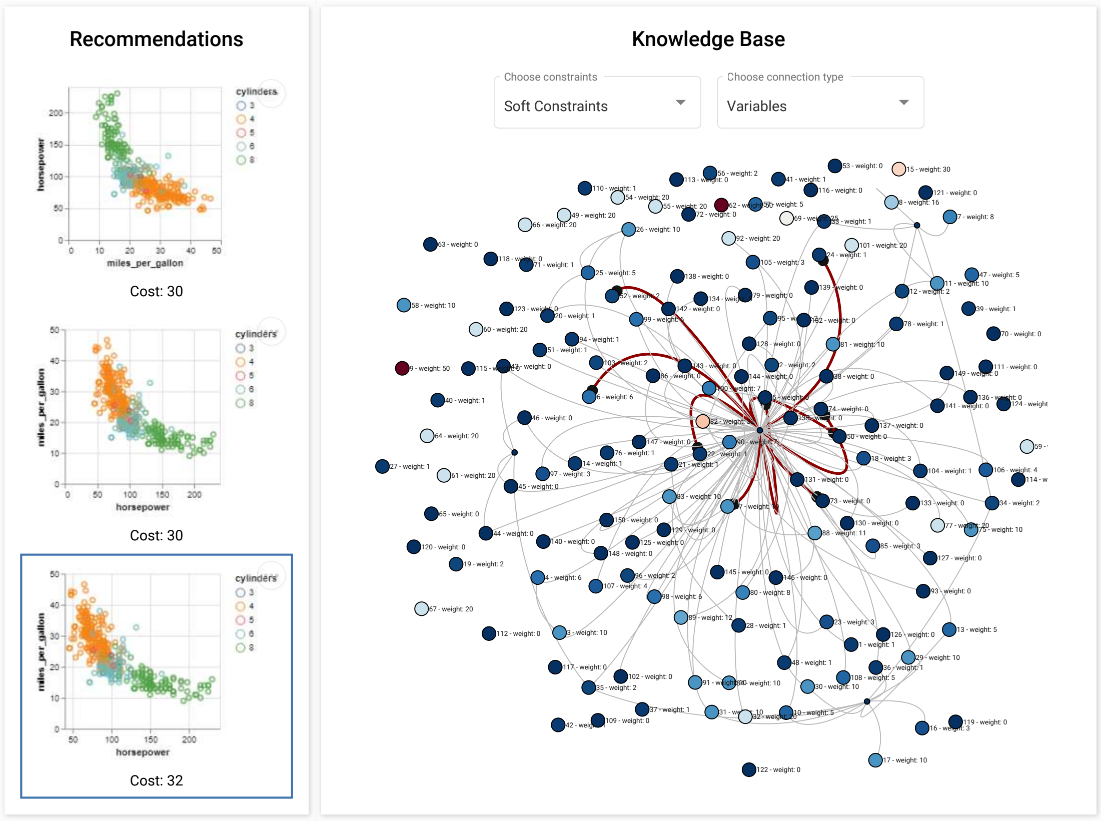


Figure 3.5: First visualization iteration of the constraints using a free node-link layout for the hypergraph. Even with few features and connections, the graph encounters the hairball problem of dense node-link graphs.

As noticeable, the resulting visualization of the first design iteration looks severely like a hairball and it is difficult to obtain any meaningful structure or order. Moreover, due to this flexible layout, each redrawing cycle of this graph leads to a repositioning of the nodes and edges. However, it is a crucial requirement for good visualization design that the same input parameters to a visualization must lead to the same visualization. Consequently, the time necessary to search and find a specific node or connection in the graph stays the same for each redrawing of the graph. Furthermore, it is hard to detect related nodes in this unstructured free node-link layout, and it is infeasible to perform any attribute-based sorting of the nodes. As a consequence of the abstraction of the constraints to nodes and edges, the content of the constraints respectively the logical expressions are hidden, and it is impractical to attach any meta-data to the nodes, such as descriptions or notes.

In the following sections we overcome the limitations and derived difficulties of the first iteration by proposing proper visual organizations and mappings.

Visual Mapping of Constraints

To get the desired consistency in the visual appearance of the graph, the first step is to add any kind of positioning restriction rule to the nodes. This rule is intended to ensure that the nodes retain their position and the context they are embedded in for every re-drawing cycle of the graph. A requirement, therefore, is that no data transformations were executed between these cycles.

Besides free node-link layouts, Nobre et al. [NMSL19] distinguish between styled layouts and fixed layouts. However, a fixed layout is not applicable, as the constraints do not have any attributes describing a x or y position on a map. Hence, a styled layout is required that adds a positioning pattern to the nodes, like arranging them in a line or along a circle. Both strategies have advantages and disadvantages, which must be weighed against each other.

Although the linear arrangement resembles the natural 1D line-based arrangement of the constraints in their code base, many connections between the nodes lead to cluttering in the visualization. The key to reducing the number of overlaps that causes this cluttering is to add a smart ordering to the nodes. However, the restriction to only one type of suitable ordering limits the positioning and arranging flexibility of the nodes. E.g., the constraints have multiple attributes, which could be used to sort the nodes accordingly. For instance, it might be interesting to sort the soft constraints based on their weight or based on the natural alphabetical order of the constraints applied to their identifiers.

Similar to the linear-based arrangement, the number of overlaps of the connections between the nodes also depends on the used sorting mechanism. In addition to the overlap problem of the connections, both layouts share further difficulties that must be tackled. One of the most critical aspects is scalability. The knowledge-base of Draco comprises around 150 soft constraints and 70 hard constraints. Since not every attribute respectively content of these constraints can be encoded as a visual variable or type of shape, a text representation is required. Because representing text is only meaningful if it is readable, it requires a minimum size on the printing media. Hence, the amount of visualizable data depends on the amount of text to be shown and its arrangement and positioning. This problem is known, for example, in 2D time-series charts where the corresponding time point labels every point on the `x-axis`. With an increasing number of time points, the labels have to be either grouped or arranged side-by-side by adding a certain angle to the labels. Rotating the labels enables positioning them closer to each other by improving the use of the available 2D space of the printing medium.

A similar strategy is required for both styled layouts since the number of constraints is sufficient that the labels of the nodes can not simply be added next to the nodes without any rearrangement. However, rotating a text makes it less readable, especially when the rotation angle is between 90 and 270 degrees. In the worst case, at 180 degrees, the text is upside-down and hence, very hard to read.

In addition to the scalability problem of labels, the visualization size likewise increases with the number of constraints by a constant factor. This factor depends on the dimensions selected for one node of the graph. In the case of circular nodes used to represent the constraints, the diameter parameter of the circle is decisive. Whereas the linear representation of the nodes increases only in one direction respectively axis, the size of the visualization in the radial arrangement of the nodes increases by two dimensions. In both cases, the visualization inevitably reaches the dimension limitations of the printing media at a specific number of constraints.

Since both layouts face similar limitations and problems to be solved, we selected the radial layout, as it best illustrates a self-contained system where the order of the constraints is not decisive. Furthermore, the constraints enclose the shared features, which gives the impression that they are part of these features and not floating around. The radial approach is additionally supported by many scientific papers and application areas where radial layouts are used to visualize flows of dependencies between the entities of a system.

In our radial layout (see Figure 3.6), the constraints of the knowledge-base are represented as nodes. In the case of soft constraints, a node encodes its weight by coloring the node accordingly and showing a textual representation of the weight in the center of the node (using node-encoding). Since hard constraints have no weight, all nodes share the same color. To be able to locate and associate the constraints, their textual identifiers are added in the form of labels next to the nodes.

Following the radial arrangement style, the node's weight label is rotated according to the angle of the node in relation to the center of the visualization. The problem of poorly readable labels caused by their rotation is tackled by mirroring them between the angles of 90 and 270 degrees (supported by similar layouts, as shown by Humayoun et al. [HEAE16]).

Furthermore, the radius of the constraints regarding the center of the visualization depends on multiple factors. Selecting an appropriate radius is crucial to avoid visual clutter of the hypergraph inside the radial arranged constraints and to avoid generating too big visualizations that exceed the visually representable area of the output medium.

An appropriate radius can be derived by taking into account the number of surrounding nodes (number of constraints), their radii, their spacings, and the number of shared features of the constraints.

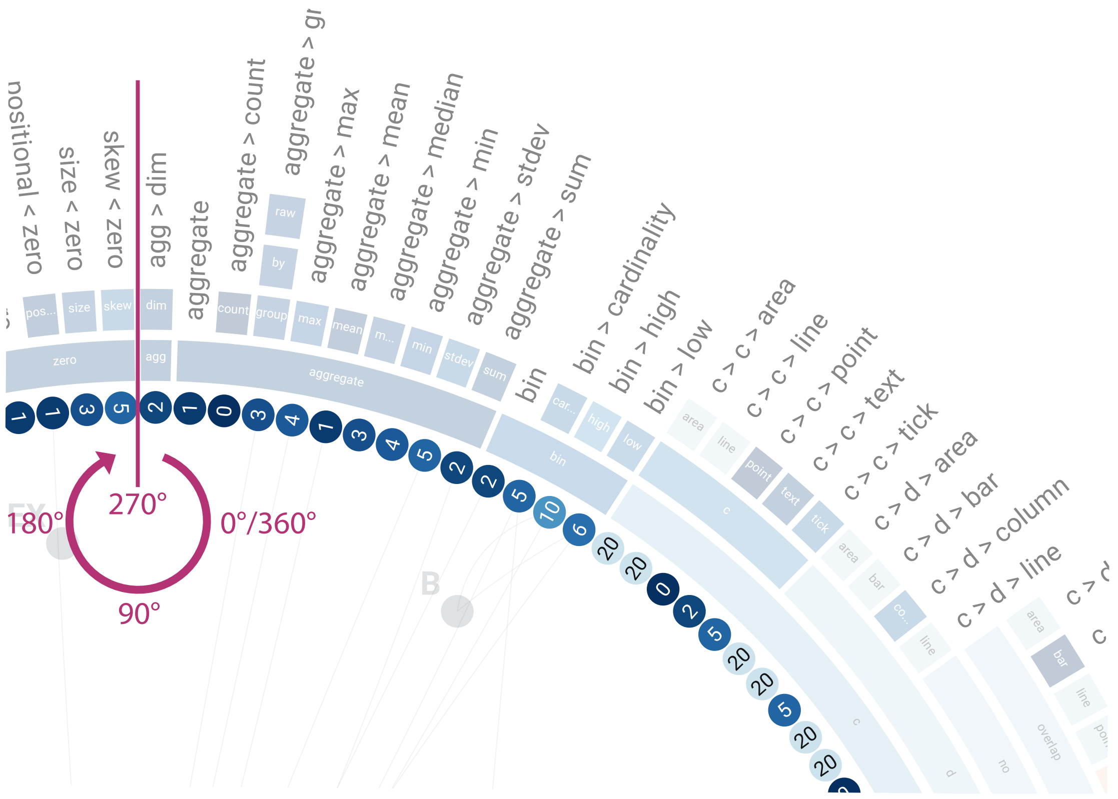


Figure 3.6: Radially arranged constraints abstracted by round nodes which are encoding the constraints' weight by number and color. The identifiers of the constraints are positioned next to the nodes and similarly arranged along the circle with the corresponding angles of the nodes.

For the color encoding of numerical values, a color map is required. It defines the mapping between the values and colors. Since various types of colormaps are available, the selection must be well-founded. A good overview of different examples and types of colormaps is given by the documentation of the d3-scale-chromatic plugin[8] of d3js. For positive real, integer, or natural numbers, single hue sequential color interpolations are usually applied. Sequential schemes interpolate between either Black or White and one other color hue. For data ranges including a clear middle like zero, diverging colormaps should be preferred[9]. However, the selection not only depends on the type of underlying data. It also depends on which properties of the data should be highlighted, respectively,

[8]`https://github.com/d3/d3-scale-chromatic` (accessed 2022/08/22)
[9]`https://blog.datawrapper.de/diverging-vs-sequential-color-scales` (accessed 2022/08/22)

in which data characteristics the user or viewer of the data visualization is interested. Hence, a common approach in scientific visualizations is to provide multiple colormaps to enable the user to explore different data characteristics by choosing an appropriate colormap.

For the color encoding of Draco's constraints' weights, we suggest a diverging colormap between Blue and Red (see Figure 3.7). The color blue identifies soft constraints with zero weight, and the color red the constraints with the maximum weight related to the total range of the weights 0-50. Although the soft constraints incorporate only positive weights, the diverging colormap supports rapidly identifying extrema and weights, which correspond to the average of the entire range.

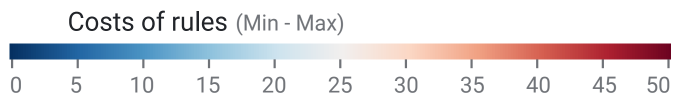


Figure 3.7: Colormap to encode the weights of the constraints as colors. The map interpolates between the two colors Blue and Red in a diverging scheme from the lowest assigned weight to the highest one.

Visual Mapping of Hierarchical Categories

Categories, groups, or similar structures in node-link graphs in either hierarchical or flat schemes can be visualized in different forms [VBW15]. Their visual mapping mainly depends on the used layout of the graph. Since we chose a radial layout for the nodes with a clear and consistent structure and positioning scheme, we apply an attached juxtaposed layout for the hierarchical encondings as suggested by related scientific visualizations [Hol06, TKE12, VBSW13] and summarized by Vehlow et al.[VBW15]. In such a layout, a group of entities is encoded as an arc radially aligned to the entities of the graph visualization. In case of Draco, an arc encloses its including constraints (see Figure 3.8). Depending on the hierarchy level of the constraints' identifiers, the length of the corresponding arc is determined by the number of constraints it encloses.

In general, there are two possibilities for placing the hierarchical groups next to the graph visualizations. We distinguish between an inside-out and outside-in strategy. In the inside-out layout, the root of the hierarchy is placed next to the nodes of the inner circle and the leaf nodes at the outermost circle depending on the number of levels of the hierarchy. Such a layout is often called a sunburst layout, as mentioned by Woodburn et al. [WYM19]. In the second case, the outside-in strategy reverses this direction. Leaf nodes of the hierarchy are placed next to the nodes representing the constraints. Hence, the leaf nodes could be merged with the constraints whereby the lowest level would be omitted.

Although the outside-in strategy benefits from one hierarchy level less than the inside-out strategy, it requires a hierarchical group structure with the same depth for each leaf node, as shown by Figure 4e in Vehlow et al. [VBW15]. Since in Draco, the constraints'

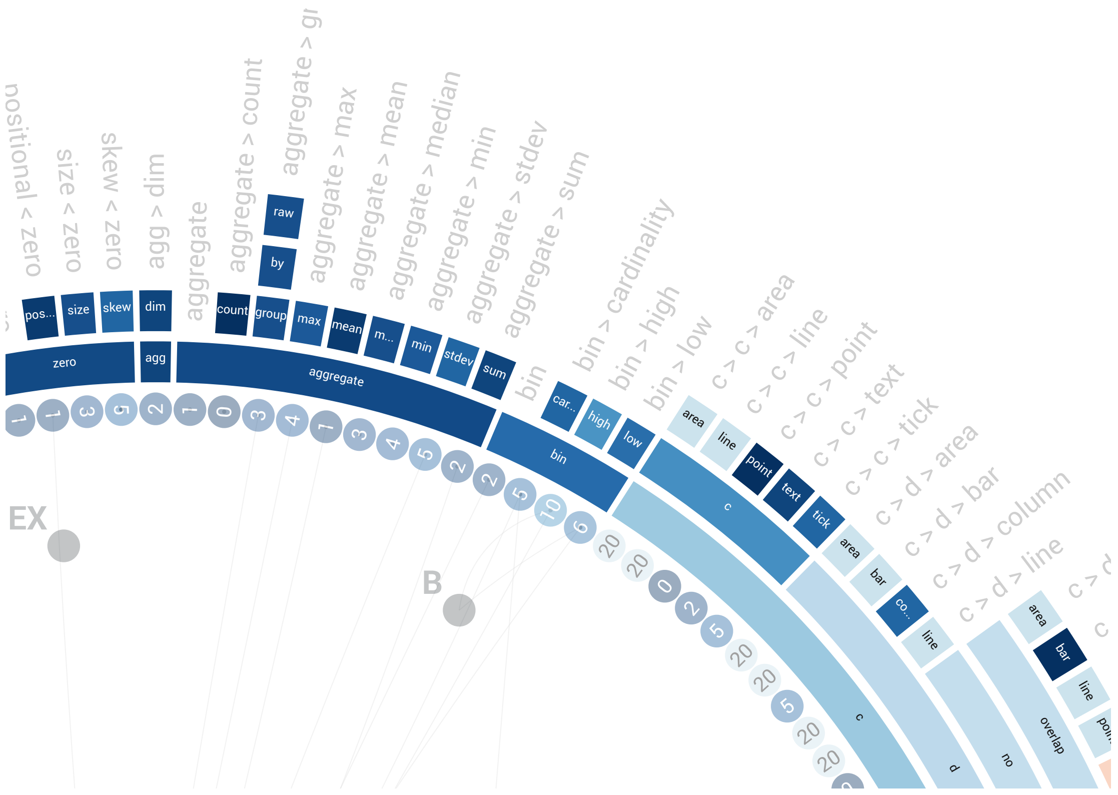


Figure 3.8: Circular attached arcs showing the hierarchical categories of the constraints.

identifiers form a hierarchy with variable depths in the leaf levels, an outside-in layout would introduce empty visual holes towards the center of the graph visualization. As a consequence, the visually detached arcs make it perceptually hard to assign the constraints to their corresponding group, respectively, category.

For these reasons, we employ the inside-out strategy and attach the hierarchical layers to the introduced nodes for the constraints in the innermost circle. The labels of the constraints are moved outwards so that they are not hidden under the surrounding arcs. Furthermore, the arcs are colored according to the average weight of the respective enclosed constraints. These colored arcs simplify finding groups of constraints with very low or very high weights. They encode their underlying name as a textual label in the arcs' center to identify them. These labels are rotated according to the arcs and shortened when the labels exceed the arcs (see Figure 3.8).

Visual Mapping of Constraints' Relationships

Each set of hard and soft constraint shares components in different levels of non-terminals. These non-terminal and terminal-based relationships, which we call feature-based dependencies of the constraints, can be shown by connecting the respective constraints using a visual link.

Connecting a subset of constraints sharing the same features with such simple links results in a fully connected sub-graph. Hence, we introduce a hypernode inside the circle of the radial arranged constraints for every shared feature of the constraints. Such a node is visually labeled by the feature's content and connected to every constraint, including this particular feature.

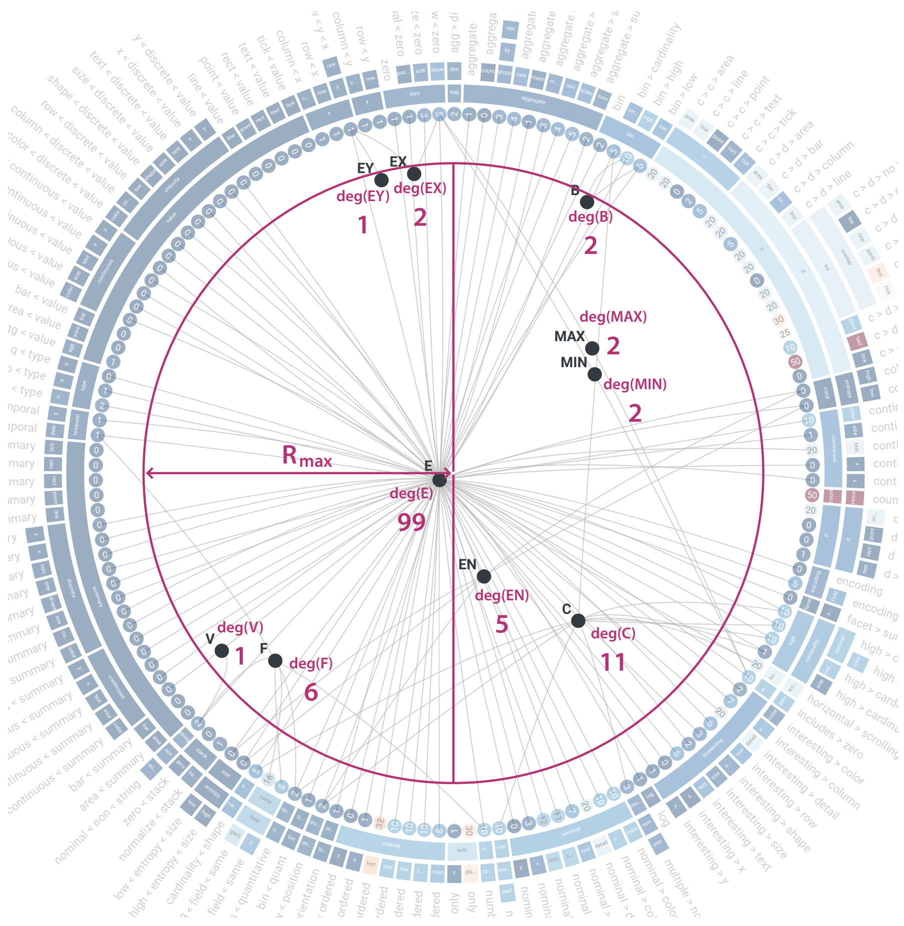


Figure 3.9: Hypergraph showing the used variables within the soft constraints of the knowledge-base. The variables are represented as feature nodes with different degrees based on the number of connected constraints. A maximum radius R_{max} ensures a minimum distance between the constraints and their shared feature nodes.

However, a common hypergraph strategy is introducing only new nodes for hyperedges connecting at least three nodes. Therefore, relationships between two nodes can be represented by a single line that does not require a hypernode. Since we are mainly interested in the included features, we introduce a hypernode for every feature used at least by one constraint. Hence, every included feature, no matter to which degree of connected constraints, is represented the same way inside the graph (see *deg(vertex)* in Figure 3.9).

A feature node's position in the hypergraph depends on its connections to the constraints. The locations of the corresponding constraints lead to a weighted centroid to which the node converges. Since at least three surrounding constraints always form a regular-shaped polygon, the centroid is invariably inside the outer circle. However, to ensure that feature nodes connecting only one or two nodes also fit this positioning restriction, we introduced a maximum radius to which feature nodes can converge.

To further reduce visual clutter and the number of feature-based relationships, we only visualize the features and their connections based on one specified feature type. In the case of Figure 3.9, only the variables of the constraints are shown.

3.3.2 Interaction Design

Few [Few09, p. 59] describes the effectiveness of information visualization based on its ability to clearly and accurately represent information and the user's ability to interact with it to gain the desired insight. Whereas static information visualization fulfills the goal of giving a first impression of the data and is sufficient for visualizing simple data, complex data requires user interaction with the different stages of the visualization process. Spence [Spe07, p. 136] states that "usually a corpus of data is so large that no single all-inclusive view is likely to lead to insight." Hence, being able to interactively explore the data and change one's view of the corpus of data is crucial for acquiring the "a ha!" experience, according to Spence.

A frequent and often practiced procedure for an exploratory approach to an unknown data domain is to apply Shneiderman's *Visual Information-Seeking Mantra* [Shn03]: 'overview first, zoom and filter, then details on demand.' Although this Mantra lacks a scientifically studied and proven methodological approach according to Craft and Cairns [CC05], it serves as a well-established and accepted guideline for designing an interactive approach to tackle complex data domains. Besides this Mantra, other taxonomies such as proposed, for example, by Kang and Stasko [YKSJ07] address the interaction design from a different point of view. Their proposed interaction methods especially tackle the users' intents:

- *Select*: mark something as interesting
- *Explore*: show me something else
- *Reconfigure*: show me a different arrangement
- *Encode*: show me a different representation
- *Abstract/Elaborate*: show me more or less details
- *Filter*: show me something conditionally
- *Connect*: show me related items

Many other authors of scientific studies related to interaction design extend this list of methods. Although these mentioned techniques fulfill a particular goal and the individuals can not be generally negated to be valuable in certain contexts, they should not be implemented at any expense [YKSJ07]. Not every potential interaction is beneficial or necessary for users to accomplish their tasks. Some interaction techniques are only meaningful in specific scenarios dependent on the data domain and its representability by different encodings.

Hence, we focused on implementing Shneiderman's guideline and using advanced interaction strategies wherever such an interaction technique is necessary to support the user in solving specific analytical and exploratory tasks. This approach is supported by the argumentation of Craft and Cairns [CC05] that this guideline can be beneficial for developing novel visualizations such as our proposed one.

Overview

Generating an *overview* of Draco's constraints and rules is implicit given through the design and intention of the static visualization design proposed in the previous section. The constraints are abstracted to the required degree to represent the two sets of constraints (soft and hard constraints) within the dimensions of a typical visual output device like a notebook or desktop monitor. A constraint is abstracted by its given meta-data weight (see Figure 3.6), hierarchical description of the body, respectively context of the constraint (see Figure 3.8), and its comprised features like variables, predicates, and similar. The proposed radial design of the visualization qualifies to create a dependency graph between these constraints to figure out related constraints and their shared features (see Figure 3.9).

Zoom/Pan and Filter

An increasing number of constraints requires reducing the size of the visual glyphs and textual labels to be still able to represent them on the output device. Especially the textual labels get worse readable due to the decreased size. Hence, the interaction technique *zooming* is required to enlarge such information. However, zooming is at the expense of presentable information within the available view space. Therefore, the zooming technique is often coupled with the interaction technique *panning* (see 1 in Figure 3.10). Panning immediately adjusts the information in focus by moving the view space to show different parts of the visualization. While panning over the visualization, the zoom level stays unchanged.

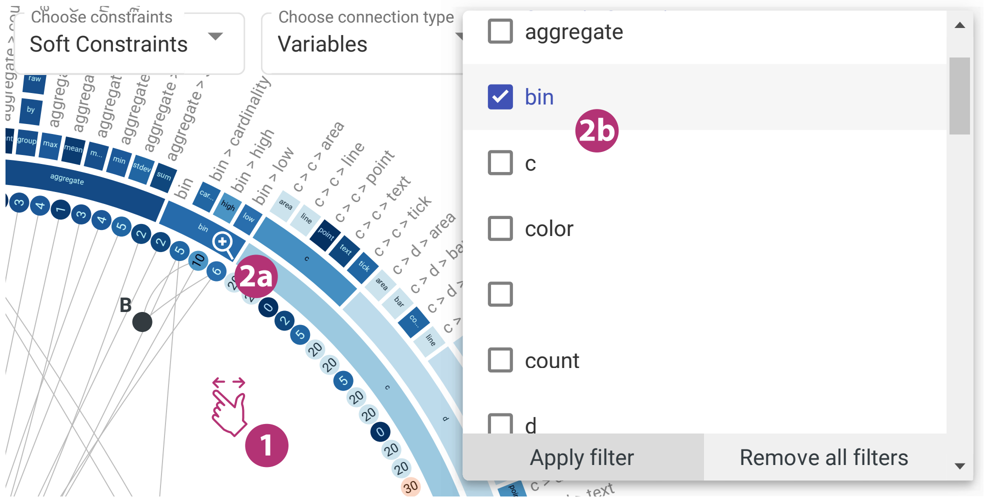


Figure 3.10: Interaction technique *zooming* and *panning* to navigate certain areas of interest of the visualization and enlarge them (1) and direct *filter* operations on specific constraint categories (2a) and indirect filter operations over a dropdown to reduce the corpus of shown constraints by selecting or deselecting constraint categories (2b).

Since some analytical tasks require concentrating only on a certain subset of constraints or one or more specific categories of constraints, *filter* out irrelevant data helps to focus on the desired information. Our approach offers two different filter options (see 2a and 2b in Figure 3.10). The visualization incorporates a direct filter operation on the constraint categories by clicking on the category of interest. Subsequently, the visualization is redrawn only showing the filtered category and its included constraints (see Figure 3.11). Furthermore, the dependency graph inside the radial layout is likewise reduced to the set of features shared by the resulting constraints. In addition, the new maximum weight is determined, and the colormap is adjusted accordingly to depict the shown range of weights in the visualization.

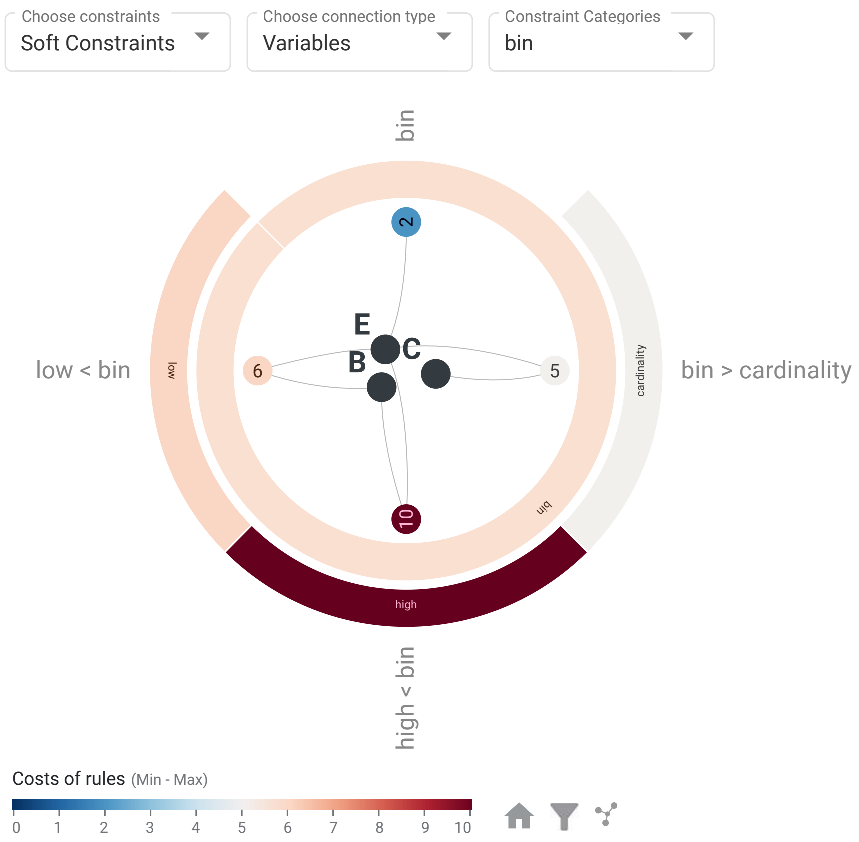


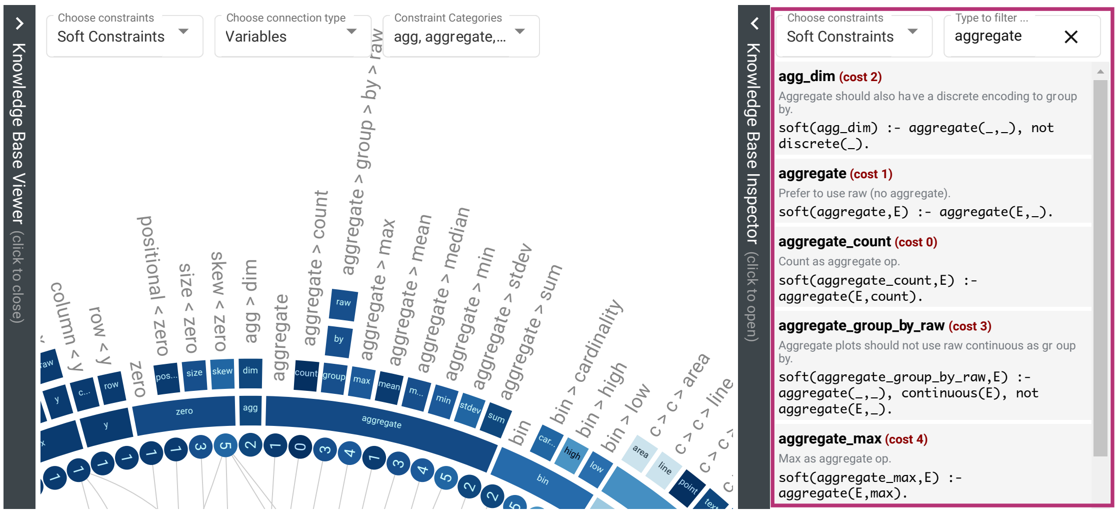
Figure 3.11: Constraints visually filtered by the category 'bin' in the highest level of the category hierarchy.

Details on Demand

Although the zooming interaction enables to enlarge parts of the visualization of interest, it does not affect the degree of detail of the focused and shown data items. By abstracting the constraints to their weight and identifier, the decisive part of the constraint for the visualization recommendation process, the body, is hidden and, therefore, inaccessible within the static visualization. Equally to the body of the constraint, other meta-data like its description and comments are hidden as well. Hence, the static visualization must become dynamic by incorporating any *details on demand* interaction functionality or view that makes such information accessible again for the user.

In theory, several possibilities exist to incorporate additional information into a visualization. A well-established approach is to increase the level of detail of a constraint when zooming into the visualization. Since we abstracted the constraints by circles, the on-node encoding technique would be an evident candidate to show such a level of detail functionality [NMSL19]. However, it is obvious that a node cannot visually include many textual descriptions and comments as required for Draco's meta-data in the knowledge-base. Another option would be to attach those meta-data to the constraints by labels. Since the necessary visual space to show such information as labels is already occupied by the labels and arcs, as seen in Figure 3.8, this approach is also impracticable.

Another possibility to make such information accessible is to detach it from the static visualization by incorporating it into a tooltip or showing it in a separate view. The difference between these two approaches is that the tooltip is a floating container within the possible drawing area of the visualization. In contrast, the separate view is entirely outside of this area. An advantage of the tooltip over the separate view is that it can be drawn directly next to the hovered or selected data item. Thereby, related information is not visually disjoint by a considerable visual distance, as is the case for the separated view. However, tooltips possibly overlap other parts of the visualization and should not exceed a certain visual extent.



agg_dim (cost 2)
Aggregate should also have a discrete encoding to group by.
```
soft(agg_dim) :- aggregate(_,_), not
discrete(_).
```

aggregate (cost 1)
Prefer to use raw (no aggregate).
```
soft(aggregate,E) :- aggregate(E,_).
```

aggregate_count (cost 0)
Count as aggregate op.
```
soft(aggregate_count,E) :-
aggregate(E,count).
```

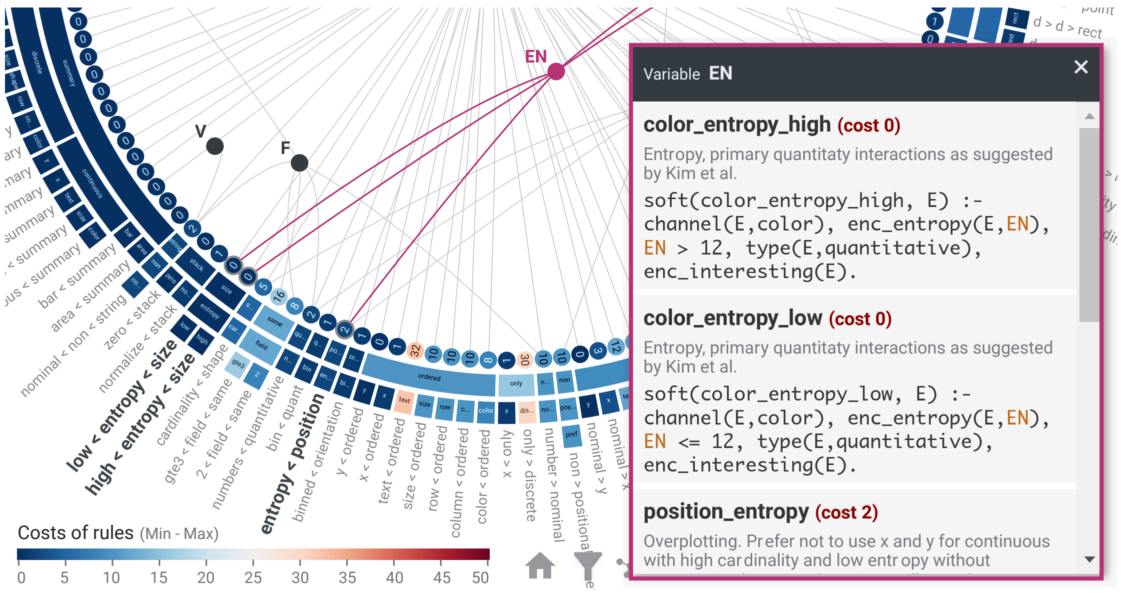
aggregate_group_by_raw (cost 3)
Aggregate plots should not use raw continuous as gr oup by.
```
soft(aggregate_group_by_raw,E) :-
aggregate(_,_), continuous(E), not
aggregate(E,_).
```

aggregate_max (cost 4)
Max as aggregate op.
```
soft(aggregate_max,E) :-
aggregate(E,max).
```

Figure 3.12: Details on demand for constraints' data through searching over the list of constraints in a separate view right next to the main visualization.

Since we want to employ the advantages of both strategies, we implemented both in our multi-view visualization to incorporate additional and more detailed data regarding the constraints and their shared features. In a separate view right next to the main visualization, all of Draco's constraints are listed among each other (see Figure 3.12). Each element of the list contains the hierarchical identifier of the constraint, its description, its logical expression in ASP-notation, and in the case of soft constraints, its weight. The type of listed constraints can be switched between soft and hard constraints using the select button at the top of the view. A search functionality enables the user to search over these constraints and their meta-data and filter out irrelevant information to the search query.

Since a user might be interested in the details of a set of constraints sharing a specific feature of interest, a tooltip incorporating this information is shown at the bottom right corner of the visualization when such a feature is selected (see Figure 3.13). The fixed position at the corner of the visualization reduces the amount of overlap within the visualization and hence, enables to enlarge the tooltip and show more information at once. The downside is that the visual distance between the clicked feature inside the graph and the tooltip is increased. However, minimized overlaps and more shown information is more beneficial than directly drawing the tooltip next to the selected feature. Besides, the feature is highlighted within the list of constraints sharing this feature. Such highlighting enables the user to quickly identify the feature within the list and retrieve the desired insights.



Variable EN

color_entropy_high (cost 0)

Entropy, primary quantitaty interactions as suggested by Kim et al.

```
soft(color_entropy_high, E) :-
channel(E,color), enc_entropy(E,EN),
EN > 12, type(E,quantitative),
enc_interesting(E).
```

color_entropy_low (cost 0)

Entropy, primary quantitaty interactions as suggested by Kim et al.

```
soft(color_entropy_low, E) :-
channel(E,color), enc_entropy(E,EN),
EN <= 12, type(E,quantitative),
enc_interesting(E).
```

position_entropy (cost 2)

Overplotting. Prefer not to use x and y for continuous with high cardinality and low entropy without

Figure 3.13: Details on demand for constraints sharing a certain feature of interest that is selected within the visualization. To minimize the overlapped area of the visualization, the tooltip has a fixed position at the bottom right corner of the possible drawing area of the visualization.

In general, information of interest can be highlighted using two techniques: emphasizing the corresponding information of interest and de-emphasizing irrelevant information. Both techniques make use of changing the encoding, respectively, the visual appearance of the information to reach the goal. In the case of textual information, either the size of the text, its color, its thickness, and several other meaningful attributes can be changed to emphasize, respectively, de-emphasize it from the rest of the shown information.

Within our visualization, we applied the emphasize strategy to highlight selected, hovered, or linked information of interest within the proposed visualization. The advantage of this strategy is that other parts of the visualization which are currently not highlighted do not get worse readable. Hence, highlighted information can be compared with other aspects of the visualization without any required additional interaction.

Comparing multiple recommendations

Likewise interesting to the elaboration of Draco's knowledge-base for analytical tasks is the analysis of interrelations between the resulting visualizations of the recommendation engine and the knowledge-base based on the constraints' violations. Since the underlying recommendation engine prints this list of violated constraints, they can be mapped back to those constraints the system is fed. Hence, it is possible to establish visual links between the recommendation results and the proposed visualization for Draco's constraints. These links can be emphasized by highlighting respective violated constraints when hovering or selecting a recommended visualization in the recommendation view (left view in Figure 3.14). Conversely, one or multiple recommended visualizations can be highlighted when selecting or hovering over one or multiple constraints in the proposed visualization (right view in Figure 3.14). Connecting these related aspects refers to the 'connect' interaction, which enables the user to view relationships usually hidden by the static visualization.

Each recommended visualization has its own set of violated constraints. Since we want to compare multiple visualizations based on their violations, we use different colors to highlight the recommendations and their constraints. Whereas framing the recommendations by color works well in the left view of Figure 3.14, it is challenging to find an appropriate highlighting method for the constraints in the right view. The reason, therefore, is that the space is limited and the nodes can be tiny. Since one constraint can be violated by multiple recommended visualizations, framing the nodes with colored borders is inappropriate. Multiple differently colored borders are difficult to distinguish at this scale. Furthermore, one constraint can be violated by a recommended visualization multiple times. Hence, another type of encoding than colored borders is necessary, which reveals the number of violations.

Therefore, we propose using a textual label next to the node disclosing the number of violations of each constraint per recommended visualization (see 2 in Figure 3.14). Each label shares the same color as its corresponding recommendation to determine the visual link between these two views. Additionally, the links to the connected features of the violated constraints are likewise highlighted to determine the affected features by the violations. The problem of overlapping colors is solved using different dash and gap patterns for each highlighting (see 3 in Figure 3.14).

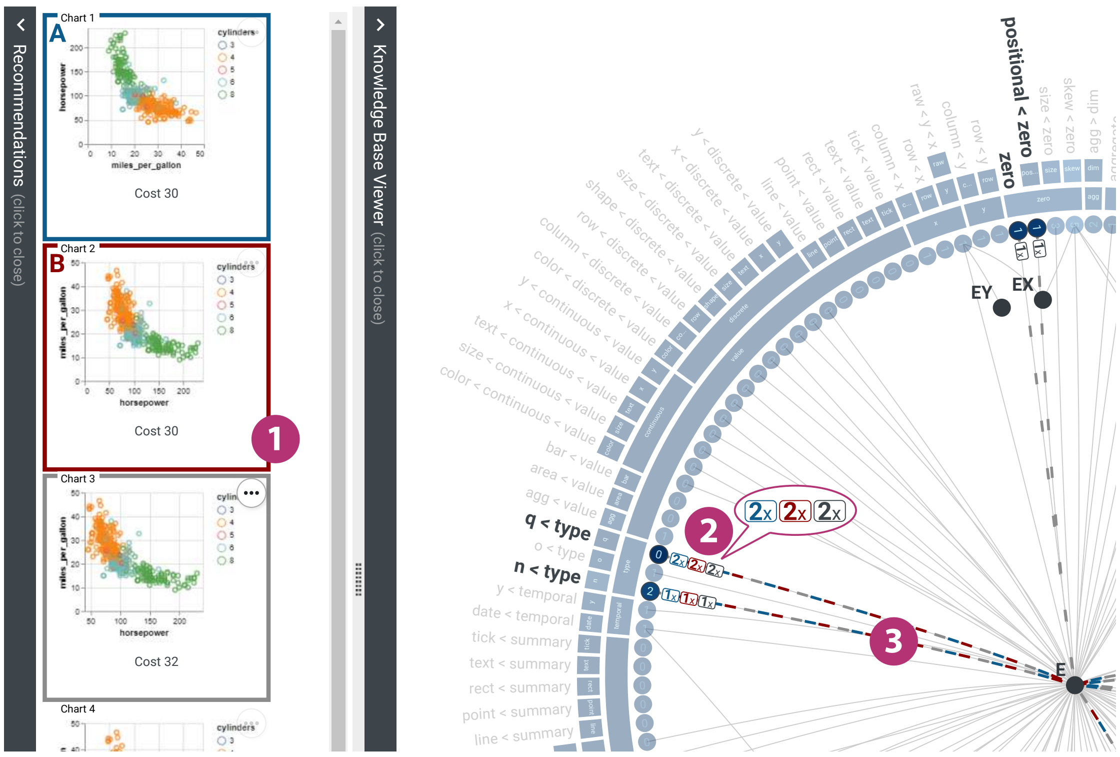


Figure 3.14: Illustration showing selected (A and B) and hovered visualization recommendations (highlighted greyish) on the left side (1), the number of constraints' violations in badges (2), and the violations of the constraints' features through highlighting the corresponding links to these features using colored dashes on the right side (3).

3.4 The Final Design

The final design combines the static visualization approach to Draco's constraints and the proposed interaction design in Figure 3.15. It targets multiple aspects concerning Draco's recommendation process and its underlying knowledge-base to enable users to find answers to the scientific questions we stated in Chapter 1.2.

Recommendation view

In the left view of Figure 3.15, the recommended visualizations for a given input query to a data domain are listed in descending order from top to down based on their accumulated costs (1a). The visualization with the lowest total cost and hence, the best-rated visualization is presented in the top left corner of the recommendation view. The underlying facts crucial to constructing these visualizations and additional meta-data, such as their violating soft constraints, can be accessed through the tooltip at the bottom of this view (1b). This tooltip is shown when a specific visualization is selected. The proposed design allows for comparing up to three recommended visualizations. Two can be selected (2a) and one additionally hovered (2b). Each one is highlighted by a colored border. They can be compared regarding their violated constraints in the knowledge-base view on the right side of Figure 3.15. Since these two views are connected and, therefore, selected or hovered parts of one view lead to highlighted aspects in the other view (2a-c), users can quickly derive the desired relationships between the recommended visualizations and Draco's soft constraints.

Knowledge-base view

In the right view of Figure 3.15, the constraints of Draco's knowledge-base are visually abstracted by nodes positioned along with a radial layout (3). Constraints' weights are incorporated into the nodes by text and color encoding. The nodes are naturally ordered by the first letters of their hierarchical categories. These hierarchical categories are abstracted by attached arcs to the respective nodes (4). Each arc represents one category and is colored according to the average weight of its including constraints. The dependency graph in the form of a hypergraph inside this radial layout connects the constraints by their shared features (5). Selecting such a feature opens the proposed tooltip, which lists all its connected constraints (6a-6b). This tooltip allows the user to explore the meta-data of the constraints and their ASP notation for deeper analysis. The colormap legend at the bottom left corner of the knowledge-base view enables visualization experts to classify the color range used for the weight encoding in the visualization. Several controls allow changing the shown type of constraints or features and limiting the number of shown constraints by filtering irrelevant categories out. The constraints can be additionally filtered by selecting a category within the visualization. After re-rendering the visualization, only the constraints and categories of interest are shown. Additional controls allow to modify the visualization's appearance and enable users to reset the view.

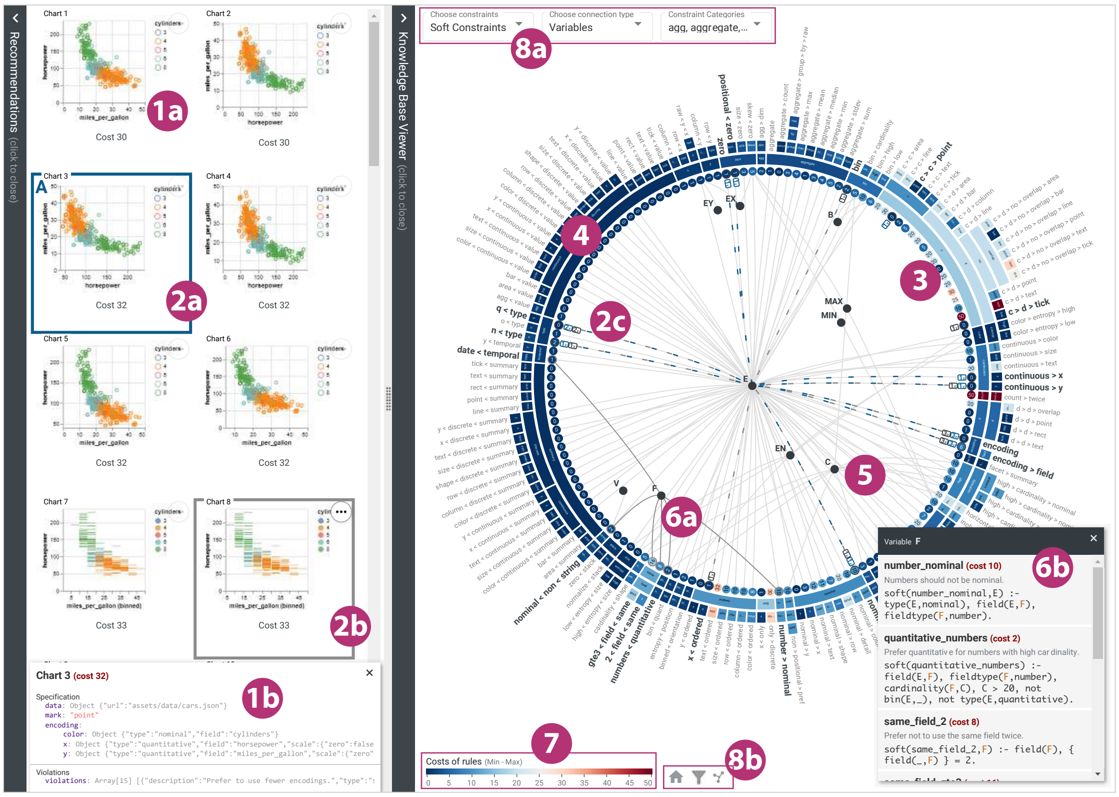


Chart 3 (cost 32)

Specification
```
data: Object {"url":"assets/data/cars.json"}
mark: "point"
encoding:
    color: Object {"type":"nominal","field":"cylinders"}
    x: Object {"type":"quantitative","field":"horsepower","scale":{"zero":false
    y: Object {"type":"quantitative","field":"miles_per_gallon","scale":{"zero"
```
Violations
```
violations: Array[15] [{"description":"Prefer to use fewer encodings.","type":"s
```

Variable F

number_nominal (cost 10)

Numbers should not be nominal.
```
soft(number_nominal,E) :-
type(E,nominal), field(E,F),
fieldtype(F,number).
```

quantitative_numbers (cost 2)

Prefer quantitative for numbers with high cardinality.
```
soft(quantitative_numbers) :-
field(E,F), fieldtype(F,number),
cardinality(F,C), C > 20, not
bin(E,_), not type(E,quantitative).
```

same_field_2 (cost 8)

Prefer not to use the same field twice.
```
soft(same_field_2,F) :- field(F), {
field(_,F) } = 2.
```

Figure 3.15: Final design of the linked visualization consisting of a recommendations list (left) resulting from the recommendation engine and the hierarchical radial hypergraph (right) visualizing the knowledge-base constraints and its connections. (1a-1b) a visualization recommendation and its specification, (2a-2b) selected recommendations and corresponding highlighted violations in the knowledge-base (2c), (3) constraints of the knowledge-base, (4) hierarchical categories of constraints, (5) feature-based connections of the knowledge-base, (6a) highlighted constraints and its feature-based connections, and detailed information about them (6b), (7) colormap legend to the used color encoding of constraints' weights, (8a-8b) controls to modify shown data and the visualization's appearance.

CHAPTER 4

Implementation

Within this thesis, we implemented a working web-based prototype of the proposed concepts for the data extraction architecture and the visualization and interaction design. The prototype's goal is to serve as an operational analysis tool for Draco's recommendation domain. It aims to maximize the user experience regarding the users' possibility to execute and fulfill the analytical tasks identified in Chapter 3.

4.1 Prototype Design

The prototype implements the theoretical elaborated split views proposed in the previous Chapter. However, the final design is supplemented by two more views, as shown in Figure 4.1: recommendation query editor and the constraint inspector. Each view is intended to be independent with weak coupling to the other views. This division takes place not only on the surface but also on the technical level by subdividing the views into standalone modules. The goal is to make the views reusable for future projects.

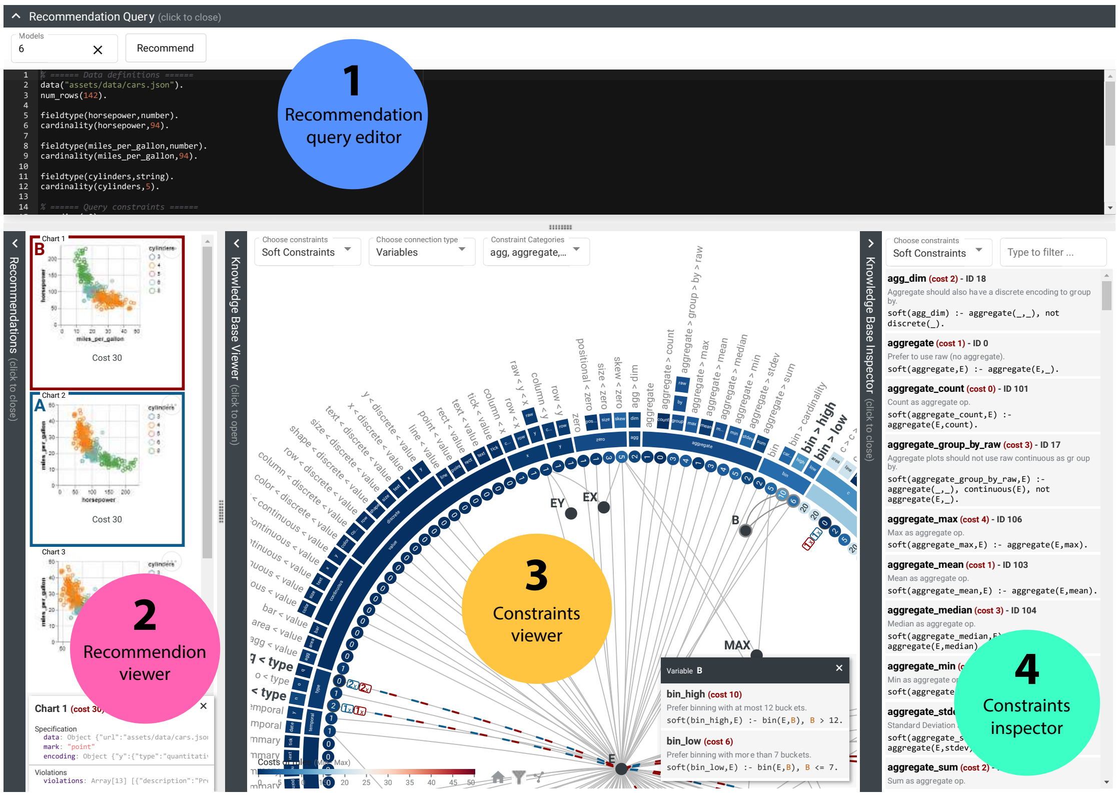
1 **Recommendation query editor**
The recommendation query editor enables the user to define an input query to the recommendation engine of Draco. It contains a definition of the data source, its column fields and data types, and statements defining the explorational task regarding the dataset. Additionally, the query editor allows setting a number of visualizations to be recommended by Draco. Finally, a button enables the user to start the recommendation process.

2 **Recommendation viewer**
The recommendation viewer shows the resulting recommendations of Draco in a tabular view. Depending on the viewer's width, multiple recommendations columns are shown. The visualizations are row-wise sorted according to their cost. A

recommendation's details are shown in a tooltip below the viewer and can be accessed by selecting the recommendation.

3 Constraints viewer
The knowledge-base viewer shows Draco's soft and hard constraints by the proposed visualization. Multiple controls are available to manipulate the shown data and its representation.

4 Constraints inspector
The constraints inspector shows the raw data of the constraints in a scrollable list view. Two controls allow to switch between the soft and hard controls and to filter respectively search them using a free text search field.



```
% ====== Data definitions ======
data("assets/data/cars.json").
num_rows(142).

fieldtype(horsepower,number).
cardinality(horsepower,94).

fieldtype(miles_per_gallon,number).
cardinality(miles_per_gallon,94).

fieldtype(cylinders,string).
cardinality(cylinders,5).

% ====== Query constraints ======
```

Variable **B**

bin_high (cost 10)
Prefer binning with at most 12 buckets.
`soft(bin_high,E) :- bin(E,B), B > 12.`

bin_low (cost 6)
Prefer binning with more than 7 buckets.
`soft(bin_low,E) :- bin(E,B), B <= 7.`

agg_dim (cost 2) - ID 18
Aggregate should also have a discrete encoding to group by.
`soft(agg_dim) :- aggregate(_,_), not discrete(_).`

aggregate (cost 1) - ID 0
Prefer to use raw (no aggregate).
`soft(aggregate,E) :- aggregate(E,_).`

aggregate_count (cost 0) - ID 101
Count as aggregate op.
`soft(aggregate_count,E) :- aggregate(E,count).`

aggregate_group_by_raw (cost 3) - ID 17
Aggregate plots should not use raw continuous as group by.
`soft(aggregate_group_by_raw,E) :- aggregate(_,_), continuous(E), not aggregate(E,_).`

aggregate_max (cost 4) - ID 106
Max as aggregate op.
`soft(aggregate_max,E) :- aggregate(E,max).`

aggregate_mean (cost 1) - ID 103
Mean as aggregate op.
`soft(aggregate_mean,E) :- aggregate(E,mean).`

aggregate_median (cost 3) - ID 104
Median as aggregate op.
`soft(aggregate_median,E) aggregate(E,median)`

aggregate_min
Min as aggregate op
`soft(aggregate`

aggregate_std
Standard Deviation
`soft(aggregate_s aggregate(E,stdev)`

aggregate_sum (cost 2)
Sum as aggregate op.

Figure 4.1: Prototype consisting of a recommendation query editor (1), a recommendation viewer (2), a constraints viewer (3), and a constraints inspector (4).

Since not every view is in the user's focus at any time, the prototype allows hiding each view independently. For example, all views except the constraint viewer can be hidden to maximize the available space for the radial visualization. However, the containers are not only collapsable but also resizeable. This resizable feature further improves the user's control over the views to optimize the used space regarding the user's task. Furthermore, the prototype is intended to be visually appealing and mature enough so that test users are not distracted from bugs from both technical and interaction perspectives.

4.2 Selected Technology Stack

A web-based technology stack with the established languages Hypertext Markup Language (HTML), Cascading Style Sheets (CSS), and JavaScript allows to quickly develop a modern, powerful, and functional prototype. Angular[1] is used as the underlying TypeScript-based framework to structure the code base in distinct self-contained parts. The framework supports fast and organized development, unit testing, and improving code maintainability with several out-of-the-box functionalities.

Furthermore, the following libraries and dependencies are necessary to incorporate Draco's recommendation system into the prototype and to extract, store, and visualize the desired information of Draco's knowledge-base. The ASP solver Clingo is integrated into the prototype as a compiled WebAssembly (WASM) module[2]. This module is accessed by Draco's web-friendly TypeScript class[3] to execute its knowledge-base and, consequently, to generate the recommendations for a given input query. An EBNF parser library[4] is integrated to parse Draco's knowledge-base and create the desired AST. The extracted data is persisted in an Indexed Database (IndexedDb) – a widely supported database by modern browsers. This database is easily accessible through simple libraries, such as ngx-indexed-db[5] and does not require a complex client-server architecture to store and access data. Additionally, the JSON formatted data can be stored directly in an IndexedDb without further processing as it would be necessary for relational databases.

For fast prototyping of the desired visualization, the used JavaScript library D3.js is a powerful toolset to manipulate Document Object Model (DOM) elements and to create complex Scalable Vector Graphics (SVG). Angular Material[6] serves as a complementary component and styling library to quickly create common UI components, such as buttons, input fields, lists, and more. To be able to modify and view the input query in the recommendation query editor, the code editor library Ace[7] is used. Lastly, the Angular friendly library ngx-json-viewer[8] is used to visually format the JSON output of the Draco's recommendation system.

[1]`https://angular.io` (accessed 2022/08/22)
[2]`https://www.npmjs.com/search?q=wasm-clingo` (accessed 2022/08/22)
[3]`https://www.npmjs.com/package/draco-core` (accessed 2022/08/22)
[4]`https://www.npmjs.com/package/ebnf` (accessed 2022/08/22)
[5]`https://www.npmjs.com/package/ngx-indexed-db` (accessed 2022/08/22)
[6]`https://material.angular.io` (accessed 2022/08/22)
[7]`https://ace.c9.io` (accessed 2022/08/22)
[8]`https://www.npmjs.com/package/ngx-json-viewer` (accessed 2022/08/22)

All of these dependencies are managed and kept up-to-date by Node Package Manager (NPM) for Node.Js[9], a cross-platform JavaScript runtime. Further dependencies of the project are either required by Angular and, therefore, not custom selected or are additional libraries to facilitate the development of the web application.

4.3 Limitations of the Prototype

Currently, the prototype faces a small set of open limitations and weaknesses. Nonetheless, these "nice to haves" are negligible to successfully accomplish the tasks given in Chapter 3 or to answer the research questions of this thesis. A crucial prerequisite is the understanding of ASP as simplifying these formal notations would go beyond the scope of this thesis.

1. Recommendation query editor requires users' to know which definable facts of Draco's knowledge-base are available.
2. No single-/multi-selection possibility of constraints (only hover interaction).
3. Only three recommendations can be compared at once.
4. Only root categories of constraints can be filtered by the given dropdown.
5. No direct comparison of the underlying raw visualization facts of the recommendations.
6. Only the constraint's features in the inner hypergraph can be selected but not the constraints themselves.
7. Only natural alphabetical ordering of constraints.
8. Only one available colormap.

[9]`https://nodejs.org` (accessed 2022/08/22)

CHAPTER 5

Evaluation

This chapter covers the conducted evaluation to verify the contributions made within this thesis. In Section 5.1 the proposed data parsing and extraction strategy are analyzed regarding its capabilities. Subsequently, the selection process of the evaluation method for the proposed visualization and the conducted evaluation and its findings are described in detail.

5.1 Evaluation of the Grammar

Evaluating context-free grammars is a challenging task. Quality characteristics of CFGs can be divided into decidable and undecidable problems.

Decidable problems[1,2] [Sip97]:

- Parsing
- Reachability, productiveness, nullability
- Regularity and LL(k) checks
- Emptiness, finiteness, membership

[1] `https://www.site.uottawa.ca/~zaguia/csi3104/Chapter18.pdf` (accessed 2022/08/22)
[2] `https://www3.cs.stonybrook.edu/~cse350/slides/decide2.pdf` (accessed 2022/08/22)

Undecidable problems[3,4]**:**

- Universality
- Language equality
- Language inclusion
- Classification of CFG according Chomsky hierarchy
- Grammar ambiguity
- Post correspondence problem
- Language disjointness

However, analyzing our grammar regarding these problems would go beyond the scope of this thesis. Especially the language equality of our grammar compared to the one incorporated into Clingo is a decisive factor. However, not only that the language equality problem is generally undecidable [Yeh80], it is also unfeasible in our case based on a formal notation basis of the grammars. The proposed EBNF notation of Clingo [GKK$^+$08a] does not follow any typical notation style of EBNF and would therefore also have to be transferred into a common formal form. For example, alternative expressions are not specified by vertical bars but are defined separately for each expression. Furthermore, their notation resembles more a BNF notation than the notation of EBNF.

Hence, the best way to evaluate grammar equality is to test whether both grammars can parse the same programs, respectively, languages. Since the whole knowledge-base of Draco is parsable by Clingo, it must be ensured that our grammar parses the knowledge-base within a reasonable time. The time needed, therefore, is dependent on the implementation of the parser, the used formal notation (e.g., BNF or EBNF), and the particular formal definition of the grammar.

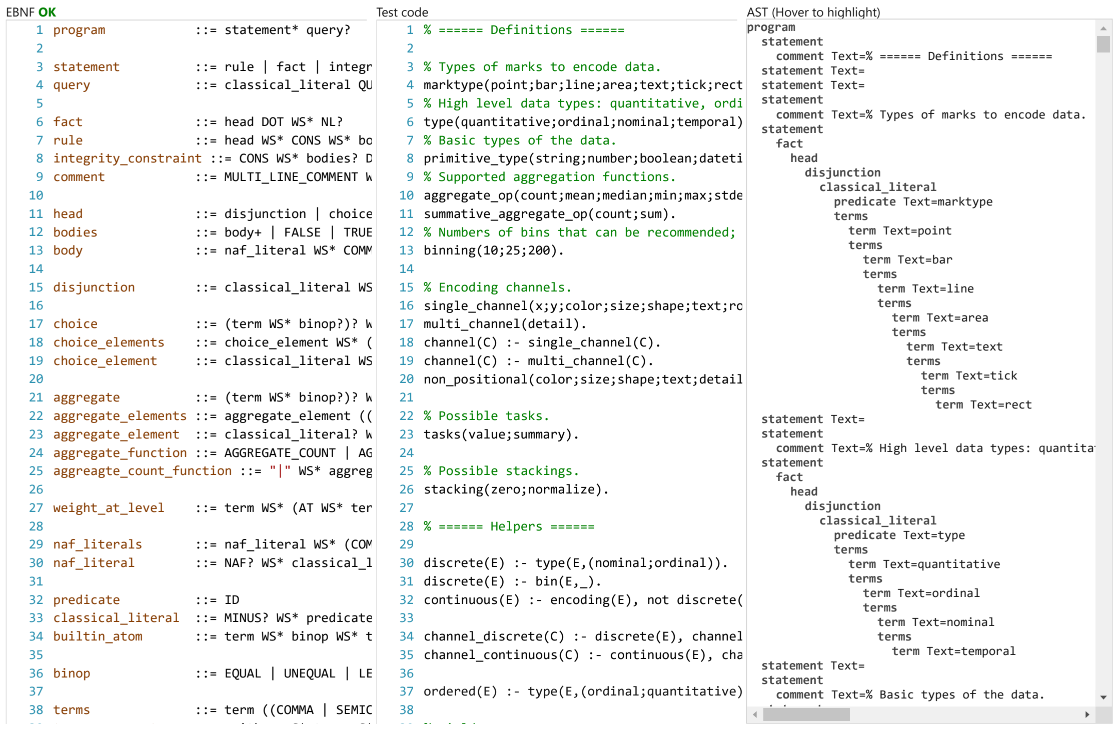
The knowledge-base of Draco is divided into several logic programming files described in their GitHub repository[5]. It consists of *topk_lua.lp*, *define.lp*, *generate.lp*, *hard.lp*, *hard_integrity.lp*, *soft.lp*, *weights.lp*, *assign_weights.lp*, *optimize.lp*, and *output.lp*. These definitions, combined with the input query provided by the user, are then parsed by Clingo and enumerated by Clingo.

Although we only need to parse the *define*, *soft*, *weight*, and *hard* statements statements to generate our visualization, we tested our grammar with all these definitions except *topk_lua*. This program is an additional script telling Clingo how to solve and minimize the costs of its recommendations. It is written in Lua[6] notation style instead of ASP and therefore, can not be described by our grammar.

[3] `http://www.cs.columbia.edu/~aho/cs3261/Lectures/L18-Undecidable_Problems.html` (accessed 2022/08/22)
[4] `https://liacs.leidenuniv.nl/~hoogeboomhj/second/codingcomputations.pdf` (accessed 2022/08/22)
[5] `https://github.com/uwdata/draco/tree/master/asp` (accessed 2022/08/22)
[6] `http://lua-users.org/wiki/LuaStyleGuide` (accessed 2022/08/22)

The results show that our grammar is capable of parsing the ASP statements *define*, *soft*, and *hard*, and *hard_integrity*. Generating the parser instance with the selected parser mentioned in Section 3.2.2 took between 18ms and 40ms. The subsequent parsing of these statements took between 300ms and 500ms. Since the browser executes this parsing operations on the Central Processing Unit (CPU), the actual performance is dependent on the current workload of the CPU. The used CPU for this performance test was an Intel© Core™ i7-6700K CPU with 4.0 GHz.

We furhter checked the correctness of the parsing result using the tool *ebnf-highlighter*[7] created as an Proof of Concept (POC) for the underlying parser *node-ebnf* – which is used by our prototype. This tool takes an EBNF grammar and test program as input and prints the resulting AST for the given input program (see Figure 5.1).



```
program             ::= statement* query?

statement           ::= rule | fact | integr
query               ::= classical_literal QU

fact                ::= head DOT WS* NL?
rule                ::= head WS* CONS WS* bo
integrity_constraint ::= CONS WS* bodies? D
comment             ::= MULTI_LINE_COMMENT W

head                ::= disjunction | choice
bodies              ::= body+ | FALSE | TRUE
body                ::= naf_literal WS* COMM

disjunction         ::= classical_literal WS

choice              ::= (term WS* binop?)? W
choice_elements     ::= choice_element WS* (
choice_element      ::= classical_literal WS

aggregate           ::= (term WS* binop?)? W
aggregate_elements  ::= aggregate_element ((
aggregate_element   ::= classical_literal? W
aggregate_function  ::= AGGREGATE_COUNT | AG
aggreagte_count_function ::= "|" WS* aggreg

weight_at_level     ::= term WS* (AT WS* ter

naf_literals        ::= naf_literal WS* (COM
naf_literal         ::= NAF? WS* classical_l

predicate           ::= ID
classical_literal   ::= MINUS? WS* predicate
builtin_atom        ::= term WS* binop WS* t

binop               ::= EQUAL | UNEQUAL | LE

terms               ::= term ((COMMA | SEMIC
```

```
% ====== Definitions ======

% Types of marks to encode data.
marktype(point;bar;line;area;text;tick;rect
% High level data types: quantitative, ordi
type(quantitative;ordinal;nominal;temporal)
% Basic types of the data.
primitive_type(string;number;boolean;dateti
% Supported aggregation functions.
aggregate_op(count;mean;median;min;max;stde
summative_aggregate_op(count;sum).
% Numbers of bins that can be recommended;
binning(10;25;200).

% Encoding channels.
single_channel(x;y;color;size;shape;text;ro
multi_channel(detail).
channel(C) :- single_channel(C).
channel(C) :- multi_channel(C).
non_positional(color;size;shape;text;detail

% Possible tasks.
tasks(value;summary).

% Possible stackings.
stacking(zero;normalize).

% ====== Helpers ======

discrete(E) :- type(E,(nominal;ordinal)).
discrete(E) :- bin(E,_).
continuous(E) :- encoding(E), not discrete(

channel_discrete(C) :- discrete(E), channel
channel_continuous(C) :- continuous(E), cha

ordered(E) :- type(E,(ordinal;quantitative)
```

```
program
  statement
    comment Text=% ====== Definitions ======
  statement Text=
  statement Text=
  statement
    comment Text=% Types of marks to encode data.
  statement
    fact
      head
        disjunction
          classical_literal
            predicate Text=marktype
            terms
              term Text=point
              terms
                term Text=bar
                terms
                  term Text=line
                  terms
                    term Text=area
                    terms
                      term Text=text
                      terms
                        term Text=tick
                        terms
                          term Text=rect
  statement Text=
  statement
    comment Text=% High level data types: quantita
  statement
    fact
      head
        disjunction
          classical_literal
            predicate Text=type
            terms
              term Text=quantitative
              terms
                term Text=ordinal
                terms
                  term Text=nominal
                  terms
                    term Text=temporal
  statement Text=
  statement
    comment Text=% Basic types of the data.
```

Figure 5.1: Tool to parse a test program and generate an AST based on a given EBNF grammar[7].

[7]`https://menduz.com/ebnf-highlighter` (accessed 2022/08/22)

However, the grammar was not capable of parsing the statements of *generate*, *weights*, *optimize*, and *output*. In the following we elaborate some of these statements which are invalid expressions according our grammar:

- `#const max_extra_encs = 5.` (part of *generate*)

  This statement represents a meta-syntax and does not originally belong to the ASP notation. It defines a constant variable `max_extra_encs.` which is used as additional information to the solver Clingo. Since the ASP-Core 2 standard does not describe such syntax, it is also not covered by our grammar.

- `obj_id(1..max_extra_encs).` (part of *generate*)

  The double dot operator in this statement represents a range operator. It allows defining several atoms that stretch on one or more parameters over a given range. Although such syntax is described in some sources found online, it is not part of the ASP-Core 2 standard and, thus, not part of our grammar.

- `#minimize { W,F,Q: soft_weight(F,W),` [...]`}.` (part of *optimize*)

  This optimized statement is used to find a minimal set of the logic program. In Draco, it minimizes the costs of violating soft constraints for a recommended set of visualizations facts. Like all meta-syntax described by Clingo, the expression is not part of the ASP-Core 2 standard and, therefore, not part of our grammar.

Since the statements in *weights* and *output* almost exclusively consist of meta-syntax described by Clingo, they are also invalid languages according to our grammar. However, if there is a need to parse such statements, our grammar can be easily extended to support such syntax.

To conclude, we have shown that our grammar can describe all ASP language features necessary to generate the essential parts of the knowledge-base of Draco. Moreover, the selected EBNF parser generates a parser and parses the provided logic program within a reasonable time. Finally, since the parser is also easily accessible within a web-based environment, all requirements related to the grammar and a respective parser described in Section 3.1.1 are met.

5.2 Evaluation of the Visualization

Evaluating InfoVis and determining a visualization's value, respectively, utility is a challenging task ([KSFN08], as cited in [WAM$^+$19]).

In the following section, we briefly overview existing evaluation techniques derived from the literature research. This overview of methods gives us the necessary decision support when choosing an appropriate evaluation method to verify our proposed visualization design.

5.2.1 Strategies and Techniques in InfoVis

Various evaluation techniques have been applied to verify new visualization approaches and designs in the past. Munzner [Mun09] proposed a nested model of four levels characterizing different domains relevant to InfoVis evaluation.

- Domain problem characterization
- Data/operation abstraction design
- Encoding/interaction technique design
- Algorithm design

Each level addresses different aspects of the evaluation objective and reaches from the validation of problem definition in the outmost level to the validation of the most specific aspect – the algorithm design, respectively, the system's implementation.

According to Munzner, two evaluation methods can test the correctness of the identified problems and tasks within a specific domain. The first method is a semi-structured interview with target users. The second is to observe them in their natural environment of dealing with the data, respectively, the target domain.

After the problem characterization, the consecutive created data and operation abstractions can be validated by executing field studies with target users. The users work on their problems in their normal daily environment. The field study evaluation comprises observations with field notes, video or audio tapes, and field logs. Such an evaluation is often concluded by a qualitative interview to receive final impressions.

For evaluating the visualization's encoding design and interaction techniques, Munzner suggests three groups of methods: controlled user study, inspections performed by experts, or qualitative results inspection. A controlled user study can be a comparative lab study, a crowd-sourcing study, or an eye-tracking study. The method inspections performed by experts can be either a heuristic informal and holistic evaluation or a cognitive walk-through of predefined tasks.

The evaluation of the algorithm design strongly depends on its expectations and complexity. Whereas small components and algorithms can be tested by unit tests or black-box testing, larger entities require methods such as end-to-end testing or integration testing.

Furthermore. a crucial factor when testing an implementation is often the system time needed to execute the implementation. The time needed allows making a statement on whether the visualization is real-time capable or not.

Besides, the literature suggests a few more methods to evaluate information visualization. They can also be categorized differently as the subdivision into analytical and empirical methods described by Mazza [Maz09]. Most of these methods have already been mentioned. However, our subject's list of relevant evaluation methods is concluded with the 'Thinking aloud' method of Lewis [Lew82]. It helps to identify and understand arose difficulties and open limitations during the execution of tasks.

5.2.2 Evaluation Method

This thesis identifies several problems and difficulties regarding Draco's knowledge-base while working with the recommendation system and its underlying constraints. Since Draco is in an early initial stage of development and use, the community behind this system is very limited. Hence, interviewing and observing target users as proposed by Munzner to verify identified problems would go beyond the scope of this thesis. The same applies to possible field studies, which would verify the data and operation abstraction design and would allow making contribution claims about it. Such a study would be time-consuming and pragmatically and logistically impractical in evaluating such a novel visualization as proposed in this thesis.

Since the implementation of the visualization is only prototype-based, the underlying algorithms' design and performance play a minor role in this thesis. Accordingly, there is no need to test and evaluate the prototype's performance regarding the refresh rate of the visualization, the number of maximum representable data entries before entering performance issues, or similar measurements.

Following the nested evaluation model of Munzner [Mun09], only the evaluation of the visualization's encoding and interaction design is left. However, we do not want to evaluate subjective factors such as the assessment of the beauty of the visualization based on its used colors or glyphs. Similar insignificant to the research questions is the evaluation of the ingenuity of the interactions.

The decisive question is whether the visualization fulfills its goals regarding the users' ability to find answers to the identified tasks and their ability to gain the desired insight into the knowledge-base of Draco. To test the visualization regarding its abilities, strengths, weaknesses, and value, we decided to combine four established evaluation methods to get broad feedback:

- Observational task performance analysis
- Thinking aloud [Lew82]
- Heuristic value-driven questionnaire [WAM$^+$19]
- Qualitative interview

These methods are relatively low-cost in their implementation expense and required equipment. Additionally, only a small number of participants are required since studies show that five evaluators are enough to obtain more than 75 percent of the problems [Nie92, WAM+19].

5.2.3 Process

The applied evaluation process consists of five consecutive parts. Due to the difficulty of the topic and the number of evaluation steps, 45 minutes are allotted for the entire process.

1. **Introduction:** Collecting necessary meta-data of the test person and introducing the user to the topic and evaluation procedure.
2. **Free exploration:** The test person freely explores the visualization and receives answers to upcoming questions.
3. **Task performance analysis:** The test person has to solve a set of 12 pre-defined tasks.
4. **Heuristic evaluation:** The test person has to fill out the heuristic value-based survey.
5. **Qualitative interview:** The test person is asked to give answers to open-ended questions about the tool and the evaluation.

Before the test persons are introduced to the topic and the procedure of the evaluation, they have to declare their academic degree, their sex, date of birth, number of years of experience in data science, respectively, information visualization, and if they participated in User Experience (UX) courses. In addition, they have to agree to the evaluation's data recording, privacy, and anonymity terms. The subsequent introduction contains an explanation of the general idea behind visualization recommendation systems, the different existing recommendation systems, the hybrid recommendation system Draco and its knowledge-base, and a presentation of the main components of the proposed visualization.

After the introduction, the test person can freely explore the tool and gets answers to uncertainties. This free exploration ensures a minimum understanding of the tool, the tool's behaviors, and possibilities. During this exploration, the participant is encouraged to ask questions and to think out loud.

As soon as the test person reaches a minimum confidence level with the tool and the data domain, s/he is asked to solve the prepared tasks. Each task starts with an introduction to the task, the targeted context, and its goals. As an aid, parts of the tool are hidden that are not required to solve the respective task. Some tasks can be solved in different ways. Although the order of the tasks is irrelevant since they are independent, all participants solve the tasks in the same order to get comparable results. While solving a task, the task's description is always visible to the user. The tasks are completed when the test person enters an answer in a designated field and moves on by clicking a button. Additionally, the test person can adjust an answer to a previous question by stepping back in the list of tasks.

Since the test persons should conduct the tasks independently, they are requested only to ask questions when they do not understand the task's goal. However, they were again asked to think aloud during the evaluation by communicating their solving procedure and possible difficulties.

After completing the tasked-based evaluation, the test person is asked to answer the survey of Wall et al. [WAM+19]. To ensure that the test person is not negatively or positively influenced in terms of the outcome of the survey, the screen recording is stopped during the survey's execution. Additionally, the test person is requested only to ask questions if s/he struggles to understand the survey's statements.

In the final step of the evaluation, the qualitative questionnaire, the test person is asked to give general feedback on the tool, its visualizations, and the evaluation.

5.2.4 Setup

The full evaluation was conducted via the video communication tool Zoom[8]. Audio and screen have been recorded using Zoom's recording functionality. Since web browsers are independent of the OS and the differences between the available browsers are insignificant, there was no special requirement about it. The test persons are asked to use an external 24-inch monitor with Full High Definition (FHD) resolution.

A separate web-based evaluation tool has been created to guide the participants through the evaluation steps and to guarantee a uniform process. Every test person receives an evaluation token which allows him/her to enter the evaluation. The token guarantees that all results can be assigned to the test person and nobody else while ensuring the person's anonymity. Furthermore, the tool incorporates visual examples in addition to the spoken introduction, a live example of the prototype for the free exploration, the task-based evaluation with textual introductions to the tasks, and a final page with links to the tool and the final questionnaire.

Above all, we use JSONBIN[9] to store anatomized automated received results of the test persons and the measured times needed to complete the tasks. This online storage is secure and only accessible via a personal account and access token.

[8]https://zoom.us (accessed 2022/08/22)
[9]https://jsonbin.io (accessed 2022/08/22)

5.2.5 Participants

Since the chosen evaluation methods require only a small number of participants, we decided to pick six test users according to their former knowledge and experience in information visualization. An understanding of rudimentary concepts of InfoVis is a minimal requirement in selecting appropriate test persons when considering the degree of specificity of the topic. The number of participants, however, is slightly above the minimum requirement of five test persons to ensure more significant results.

- P1: 24 years old male master student of Visual Computing having four years of experience in data science and information visualization and participated in UX, respectively, information visualization courses.
- P2: 31 years old male who scientifically works in the field of information visualization with four years of professional experience in InfoVis who participated in respective courses. The test person stated that he is already familiar with Draco and its concepts but has no in-depth knowledge or did not directly work with it.r
- P3: 39 years old female with a doctoral degree and eight years of experience in data science and information visualization who did not participate in information visualization courses yet.
- P4: 25 years old male master student of Visual Computing having six years of experience in information visualization and participated in InfoVis lectures.
- P5: 26 years old female master student of data science having three years of experience in information visualization and participated in InfoVis lectures.
- P6: 32 years old male researcher in the field of biomedical image informatics with 11 years of experience in information visualization also participated in InfoVis lectures.

5.2.6 User Test

Within the first task **T1** the test users had to identify soft constraints with exceptionally high costs. This task aims to test if users can find outliers in the soft constraints regarding the constraints' weights. Whereas P1, P3, P4, and P6 quickly identified the outliers based on the constraints' eye-catching red color, P2 needed additional time to get confident with the tool. Moreover, P2 had difficulties understanding the task's goal. He thought he had to identify outliers of violated constraints that were not part of the task's goal. After clearing up this misinterpretation, P2 quickly identified the searched outliers. P5, however, would have preferred to search in the list of constraints and sort them according to their weight. Since such a sorting mechanism was not implemented for the list, she then quickly solved the task using the knowledge-base viewer. It can be concluded that all participants solved this task without major problems.

In **T2** the goal was to find one category of constraints with a very low average weight. This task aims to find out if users can quickly identify categories of constraints that have little to no influence on the recommendation results and vice versa. The participants P1, P2, P4, P5, and P6 quickly identified the category `summary` as a category of constraints with a very low average weight. P3 first struggled to understand where the average cost of multiple constraints is depicted within the visualization but then quickly found the category `summary`. All participants except P6 almost only focused on the categories `summary` and `value` and picked the respective category by briefly comparing the weights of these included constraints. These categories have very low average weight. P6 was also the only person who chose `value` as the answer, although it was not the category with the lowest average. However, `value` is also perfectly fine since only one category with a very low average weight was the task's target. A possible explanation for why the participants were focused on these two categories only might be that they are also two of the three categories that include most of the soft constraints.

Since we were aware of the fact that `summary` and `value` are categories which are including many constraints and, therefore, might distract the participants, the goal of task **T3** is to identify categories with a very high average weight. Based on the weights of Draco, the best answer to this task is a category with only one constraint. Hence, this task shows whether the size of the arcs negatively influences the users or not. The results show that P1 seems to confirm this assumption since he selected the category `c` as the answer, including the third most constraints within the knowledge-base. When comparing all categories, however, `c` is only in the middle of the possible range of averages. Although not all other participants found the best answer, their answer and communicated solving strategy for this task show that they were not distracted by the size of the arcs. An interesting observation was that the majority of the participants mainly focused on the base categories of constraints only – the categories at the highest level of the hierarchy. However, since two participants explained that they also could select a sub-category to answer this question, it can not be concluded that the hierarchy of categories shown by the arcs is entirely misleading.

In task **T4** the test persons had to identify a variable that is part of most soft constraints. The task aims to determine if users can identify occurrences of features within the knowledge-base. This knowledge is of importance to be able to make a statement about whether some features have an extensive or relatively little influence on the outcome of the recommendation system. All participants quickly detected the variable `E` to be the variable most used within the system without any problem. It could be observed that they had no issues understanding how the variables are represented within the visualization and that they flawlessly used the dropdown select to switch between the features represented within the hypergraph. P6 stated that he works a lot with graph theory in his daily business and, therefore, wished to have the vertex degree (sum of incident edges to a vertex) either visualized by text or node color encoding to answer this question.

Since the variable `E` is obviously seen in the visualization as a node with most incident edges, the task of **T5** is oppositely required to identify a variable used by only one constraint. All participants quickly and correctly answered this task. Merely, P3 needed two attempts to find a correct answer. A possible reason, therefore, might be that she first focused more on variables closer to the middle of the hypergraph than on the outer ones.

Task **T6** required the participants to name all constraints using the predicate 'aggregate,' which does not have `aggregate` in their name. Except for P1 and P2, all other participants used the graph to answer this question. This task aims to determine whether users understand the difference between the identifiers of the constraints and their content. The participants solved this task using different strategies. P1 preferred the tooltip showing all constraints related to this predicate to answer this task. P5 was first confused by the task's description. But then she answered the task also using the tooltip after selecting the node `aggregate`. P2, P4, and P6 used the smartest way to solve this task. They filtered out the category `aggregate` using the filter functionality and then answered the task by writing down all the constraints shown in the tooltip when selecting the predicate inside the hypergraph. However, P6 first deselected all categories of constraints except `aggregate`. However, he then correctly determined that the remaining constraints are exclusively those that have `aggregate` in their name. Thus, he realized that he had to change his solving strategy. It was also noticeable that some test persons had problems understanding the filter dropdown. They were confused by which categories are visible within the filter and what selecting or deselecting options means to the filter. However, this task showed that the tooltip played a major role in solving the task.

In task **T7** the participants had identify how often the constraint `encoding>field` is violated by the recommended visualization. This task aimed to determine if users understand the connection between the recommended visualizations and the constraints shown in the proposed visualization for the constraints and if they can identify the number of constraints' violations. Whereas P1, P3, P4, and P5 had no issues identifying the violations based on the shown badge next to the constraints, P2 and P6 were confused by the colored edges connected to the constraint `encoding>field` and the number shown in the badge. The reason for this confusion was that the number of these edges is the same as the number of violations when the predicate connections are shown inside the graph. This scenario was deliberately created to test whether this confusion occurs or not. A noticeable side-observation was that some test users read the symbol '>' inside the name as "encoding larger than field." Since this symbol has multiple meanings in different domains, especially in mathematics, it can be concluded that the use of this symbol is not necessarily interpreted equally by all users in this context.

To investigate the extent of this possible confusion, task **T8** aimed to identify affected variables by the violations of a recommended visualization. The correct answers to the task are the variables `E` and `F`. Except for P1, who only found the `E` variable as an answer to this task, all other participants had no issues solving this task. Hence, this confusion only exists for the number of violations of the constraints. This confusion would, therefore, undoubtedly be resolved if nodes and edges are hidden when answering task **T7**. However, the users did not use the hide functionality explicitly created for this purpose. The reason, therefore, might be that they were not aware of this functionality.

In task **T9** the participants had to name at least one constraint by which the two recommended visualizations differ. The goal of this task was to clarify if users can find differences between the recommendations based on their violations. Although all participants found differences, P2, P4, and P7 first identified differences based on the colored edges. This observation shows that the confusion between the violations of the constraints and the affected variables remains. After showing those particular test users the functionality of hiding the nodes and edges, they had the "a ha!" experience [Spe07, p. 136]. P4 and P6 stated that the hypergraph is so prominent, especially the colored edges that highlighted edges drew the main focus.

Again, to test the opposite case, in task **T10** the participants had to identify a variable affected by both recommended visualizations. Except for P1, who likely misunderstood the question and declared a constraint to be the answer to the task, all other participants had no issues finding the correct answer. Within this task, P5 stated that the logical expressions of the shown constraints inside the tooltip are misleading since the symbol ':-' is missing. Therefore, she was not able to read and understand the logical expression.

So far, in tasks **T7** to **T10**, the goal was to identify violated constraints and affected variables of recommended visualizations. Task **T11** and **T12**, on the other hand, deal with the opposite direction between the two views. The goal of **T11** is to identify how many recommended visualizations by Draco are violated by the constraint `continuous>x`. In this task, six visualizations are recommended by the system, and the requested constraint is violated by two of the six visualizations. P3 and P6 first looked at the problem from the wrong side. They hovered one recommendation after the other to discover if the requested constraint was violated or not. Whereas P3 needed a hint to change the strategy, P6 correctly identified that there must be a smarter way to answer this question. After confirming this assumption, he correctly solved the task by hovering over the constraint. All other participants had no problem solving the task. However, P4 and P5 recognized a problem with the current functionality. Since the constraints in the knowledge-base viewer can only be hovered and not selected, it is impossible to scroll in the recommendations view simultaneously. Hence, this task can only be easily solved for the currently visible recommendations on the screen.

In the last task **T12** the participants had to find out how many recommended visualizations commonly violate the sub-category `d` of the base category `c`. P1 had problems understanding the task in the first place but then solved it correctly. P5 also misunderstood the task and thought that, again, a constraint containing `c>d` is the target of the task rather than a category. Hence, she did not find any visualizations which violate the sub-category `d`. However, the reason for this confusion could be in the task's description. The exact wording of the description contains "category `c>d`". This expression could be misleading if one is not already sufficiently familiar with the hierarchies of the categories.

5.2.7 Value-Driven Survey

The heuristic value-driven survey was conducted by using the tool Excel. The sheet contained an introduction area where essential terminologies used within the survey are explained in more detail. The introduction also explains what is meant by 'data cases' and 'data attributes' with respect to the visualization. These explanations are intended to avoid misleading interpretations of the participants with the consequence of meaningless results.

However, even with this short introduction and explanations of difficult terminologies, some participants, especially P1, struggled to understand some statements made in the survey. According to the asked questions during the answering of the survey, P1 sometimes did not understand which aspect of the tool the statements referred to. The reason for this confusion could be that our tool consists of two connected views, each having its own data domain.

The results of the survey are shown in Table 5.1. For each statement, the participants P1 - P6 submitted a score between 1 (strongly disagree) and 7 (strongly agree). The full descriptions of the survey's statements are shown in Table A1.

As noticeable, we received the lowest average value score by participant P1. All participants except P1 rated the visualization based on statements S4, S9, and S14 between 5 and 7. P1 rated S4 and S9 with 2 (disagree) and S14 with 3 (somewhat disagree). S4 declares that "The visualization helps generate data-driven questions," S9 "The visualization provides a meaningful spatial organization of the data," and S14 "The visualization provides a comprehensive and accessible overview of the data." Since it is one of the main goals of the visualization to support these three statements, it is surprising that P1 has an opposite valuation. A possible explanation for those bad scores of P1 might be that he was not able to set the context and meaning of some terminologies correctly.

However, it is less surprising that the participants P2 and P6 achieved the best average score for all statements. P2 was already familiar with some characteristics of Draco, and P6 deals a lot with graph visualizations and graph-based data in his daily business. Furthermore, they are used to the typical terminologies in the field of information visualization and have a lot of experience in related research fields.

By contrast, we received the lowest average scores in S18 and S19. These statements are part of the confidence component and claim that the visualization helps avoid making incorrect inferences. This valuation might lead back to the misleading classified colormap by some participants, and the confusion occurred between violated constraints and violated features. P3 could not find an appropriate rating for S18 and S19 at all. Besides, S21 did not receive a rating at all, and thus, we could not determine an average, median, or standard deviation value. The participants correctly argued that the visualization does not cover data quality inspection, and therefore, it is not possible to encounter unexpected, duplicate, missing, or invalid data.

According to the determined standard deviations, the participants obtained the greatest

								μ	x̃	σ
		P1	P2	P3	P4	P5	P6	Average	Median	Stand.Dev.
I	S1	6	7	7	6	5	7	6.3	6.5	0.8
	S2	5	7	6	7	5	7	6.2	6.5	1.0
	S3	5	7	7	6	7	7	6.5	7.0	0.8
	S4	2	6	6	5	7	6	5.3	6.0	1.8
	S5	4	4	4	6	7	7	5.3	5.0	1.5
	S6	6	6	6	5	5	7	5.8	6.0	0.8
	S7	4	6	5	4	7	5	5.2	5.0	1.2
	S8	5	7	7	6	7	7	6.5	7.0	0.8
T	S9	2	7	6	5	6	7	5.5	6.0	1.9
	S10	3	7	6	6	6	7	5.8	6.0	1.5
	S11	7	6	7	6	7	7	6.7	7.0	0.5
	S12	5	5	7	7	6	6	6.0	6.0	0.9
	S13	6	6	7	7	6	7	6.5	6.5	0.5
E	S14	3	7	6	6	7	7	6.0	6.5	1.5
	S15	5	7	6	5	7	7	6.2	6.5	1.0
	S16	4	6	4	6	4	7	5.2	5.0	1.3
	S17	3	7	5	6	5	7	5.5	5.5	1.5
C	S18	5	7	N/A	5	2	4	4.6	5.0	1.8
	S19	3	6	N/A	2	5	4	4.0	4.0	1.6
	S20	5	7	6	3	3	6	5.0	5.5	1.7
	S21	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
Avg.		4.4	6.4	6.0	5.5	5.7	6.5	5.7	5.9	1.2

Table 5.1: Survey's results showing each participant score made for the statements proposed by Wall et al. [WAM+19]. The table also includes average, median, and standard deviation scores. The leftmost column shows the four base components of the survey Insight `(I)`, Time `(T)`, Essence `(E)`, and Confidence `(C)` with each subcategories presented by the statements S1 - S21. N/A states that there was no answer in the meaning that the participants were not able to find a suitable rating for the respective statement. The score range is between 1 (strongly disagree) and 7 (strongly agree).

agreement in statements S11 and S13, with an average score of 6.7 and 6.5. These statements declare that the interface supports using different attributes of the data to reorganize the visualization's appearance (S11) and that the visualization avoids complex commands and textual queries by providing direct interaction with the data representation (S13). Especially the high average score of S11 is surprising since the current implementation of the prototype does not cover all reorganization possibilities proposed within this thesis. For example, the prototype does not yet have a sorting functionality.

On the other hand, the participants had the smallest agreement in statement S9 according to the standard deviation of 1.9. Although the majority of the participants agreed with statement S9, P1 is an outlier on the downside with a rating of 2. S9 declares that the visualization provides a meaningful spatial organization of the data. Based on the feedback received from P1, the reason for this valuation cannot be derived. However, it can also be observed that P1 submitted the lowest score on almost all statements of the survey. Hence, we assume that he was the least able to cope with the visualization.

	Insight	Time	Essence	Confidence
P1	4.6	4.6	3.8	4.3
P2	6.3	6.2	6.8	6.7
P3	6.0	6.6	5.3	6.0
P4	5.6	6.2	5.8	3.3
P5	6.3	6.2	5.8	3.3
P6	6.6	6.8	7.0	4.7
Avg.	5.9	6.1	5.7	4.7

Table 5.2: Results of the heuristic value-driven evaluation summarized for the four different components of the survey of Wall et al. [WAM$^+$19].

Furthermore, when considering the average values for each component per participant in Table 5.2, it can be observed that P1 did not receive an overall essence of the data. However, based on the other participants' ratings for this essence component, the visualization certainly provides a big picture of the data and an understanding of the data beyond individual data cases. Besides, it is not surprising that P4, P5, and P6 worst-rated the confidence component since we received the most feedback from them regarding applied visual encodings, potential issues, and misleading representations.

5.2.8 Questionnaire

In the final questionnaire round, all test persons stated that they had a pleasing overall impression of the tool and its visualizations. They mentioned that it is easy to use, intuitive, and understandable. P4 pointed out that the visualization especially creates a good overview of which constraints, categories of constraints, and feature-based connections exist within the knowledge-base of Draco. P6 noted that the domain and its data are, in general, very difficult to comprehend and thus, emphasized the importance of the introduction in the beginning of the evaluation – even for visualization experts. However, P6 states that he does not dare to make any statement on the generalizability of the visualization. He thinks that the visualization solves the tasks but does not know if the visualization could be applicable in a different scenario. For example, P6 states that he could not find any direct dependencies between the constraints, although such radial graphs are often used to show such dependencies between data entries. However, the constraints do not have such a characteristic that one constraint is dependent on the outcome of another one. Therefore, such non-existent dependencies cannot be visualized.

Some test persons, especially P4 and P5, stated that they had difficulties in using and understanding the filter functionality. They were confused by the checkboxes within the dropdown and the shown filter possibilities. P5 suggested showing all filter possibilities within one view without having to scroll them. Otherwise, it is difficult to tell which base categories of constraints are currently filtered. P5 also did not expect to be able to select features by clicking on their labels. Sometimes she struggled to select nodes which are placed very close to each other.

P2, P4, and P5 stated that the applied colormap from Blue to Red does not perfectly fit in this scenario. First, the colors used for highlighting the recommended visualizations in the recommendation viewer are the same as applied within the colormap used for constraints' weights. Second, the diverging colormap for the weights might be misleading. Although those participants acknowledge that the diverging colormap better exposes differences between weights in the low range, P2 stated that it might convey that constraints having weights in the middle range are neutral.

P4 and P6 suggested unifying the click behavior within the visualization. On the one hand, clicking on the categories leads to filtering the data, and on the other hand, clicking on the features leads to more detailed information. Furthermore, they missed this selection behavior of the features also for the constraints. To be able to select constraints would allow to simultaneously view the constraints' details and scroll in the recommendation view. Besides, since P5 heavily used the constraints inspector during the evaluation, she missed an advanced filter and sorting functionality for this list. She stated that for some tasks, she would have preferred to directly use the list of raw data to answer the task instead of the visualization.

5.2.9 Summary

Overall, the evaluation has shown that the proposed interactive visualization of the knowledge-base viewer is suitable to answer questions regarding the visualization constraints and their shared features. The participants were able to identify the interrelations between the constraints and Draco's recommended visualizations based on highlighted violations. However, the evaluation has also shown that a detailed introduction to the topic is essential, even for visualization experts.

Identified difficulties that arose during the task performance analysis are reflected in the heuristic value-driven survey. Whereas the participants performed well in acquiring insights, using the tools abilities to quickly seek information of interest and obtain an overall essence of the data, they had problems feeling confident with the data domain in general. However, the participants stated that many difficulties during the evaluation are caused by a lack of time dealing with the visualization and the topic in general. This argument is supported by the observation of a steep learning curve during the evaluation. It was clearly noticeable that participants improved in understanding the idea behind the tool and its visualizations while solving the tasks. They felt more confident in interacting with the visualization and understanding the individual visual mappings towards the end of the user test.

The evaluation has also shown that interaction possibilities are still missing or too little mature, such as the filter and constraints selection options. In addition, the current prototype comprises minor misleading representations and double occupancies of color encodings.

Above these identified problems and observations, the participants pointed out the overall intuitiveness of the prototype and its appealing representations. They also acknowledged the advanced stage of the prototype regarding its completeness and the absence of serious bugs.

CHAPTER 6

Discussion and Future Work

6.1 Contributions

Identifying incorporated predicates, variables, and other syntax features in Draco's soft and hard constraints and large logic programs, in general, is a complex and time-consuming task. However, knowing and understanding these features and their distributions across the set of logical statements of a program is crucial to comprehending its goal and characteristics. Such knowledge is essential to adjust, maintain, and extend the program according to current needs. The evaluation has shown that expert users can identify those features and interrelations between the constraints using the proposed hypergraph within seconds (research question RQ1.1).

The costs, respectively the weights of the soft constraints, play a decisive role in the recommendation process and its outcome. The distribution of those weights significantly impacts the order of the recommended visualizations. These weights can either be hand-tuned or derived from the learning mechanism proposed by Draco. Understanding the weights' distribution and assignment is crucial for adjusting the respective weight determination methods to fit the user's needs and improving the preferences within recommendation results. The proposed visualization incorporates these weights by the soft constraints' color encoding. Additionally, the exact value of the weights is textually shown within those nodes. According to the evaluation results of the visualization, the users are able to quickly identify constraints having low costs and outliers having exceptionally high costs. By inspecting the colored arcs representing hierarchical categories of constraints, the users can identify whole categories of constraints that play a minor role in the recommendation process. Although some evaluation participants were distracted by the arcs' size when inspecting average values, the majority submitted correct answers to the respective evaluation tasks. Hence, we assume that users can identify and understand weight distributions within the soft constraints and, consequently, can adjust the weights according to their needs (research question RQ1.2).

Understanding the constraint-based violations of recommended visualizations and identifying shared violations of multiple recommendations is essential to comprehend the system's recommendation behavior and identify possible weaknesses. Considering the evaluation results, the test persons could recognize shared violations and their degrees (research question RQ2.1). However, some participants were confused by highlighting the affected connections within the hypergraph and, therefore, identified affected features by violations, although we sought for violated constraints. This confusion is reflected in the heuristic value-driven evaluation, where weaknesses in the confidence component have been identified. We assume that if this confusion is eliminated in a further development of the prototype, expert users will have no issue identifying shared violations of multiple recommendations. This assumption is supported by the fact that as soon as the evaluation participants were aware of the respective differences, they could quickly identify the violated constraints.

So far, to the best of our knowledge, no tool or visualization exists in the context of Draco that supports selecting a constraint and simultaneously inspecting recommendations that violate this particular constraint. Especially this direction of inspection is intended to allow users of the system to identify a constraint's influence on the recommendation results (research question RQ2.2). Although the test persons of the evaluation identified a bottleneck within the current version of the prototype, namely the lack of a feature to select a constraint within the visualization and simultaneously previewing the list of recommendations, they were able to complete the respective tasks successfully.

6.2 Generalizability

The proposed visualization was designed to visualize decisive attributes and interrelations between Draco's soft and hard constraints. However, the generalizability of the visualization remains unclear. According to the characteristics of the visualization, it can be used to visualize data structures that consist of a set of entities where the individual entities are part of hierarchical categories. Additionally, shared features of the entities can be represented by the proposed hypergraph within the radial arranged entities.

6.3 Modular Version of Draco

Since the publication of Draco [MWN+19], the team behind Draco is elaborating on an experimental modular version of Draco in *Draco 2*[1]. In *Draco 2*, the visualization facts are defined more modularly. The reason, therefore, is still not communicated, and no related scientific-related paper exists. According to the current state of the project[2], the original notation style of the soft and hard facts did not change much. One of the more striking changes is the renaming of 'soft' into 'preference' and 'hard' into 'violation' and the modular definition of the visualization facts. We assume that without any future

[1]`https://dig.cmu.edu/draco2/intro.html` - last accessed 2022/08/22
[2]`https://github.com/cmudig/draco2` - last accessed 2022/08/22

elementary changes to the notation of the knowledge-base in Draco2, it should still be visualizable as proposed in this thesis. However, the data extraction strategy must be adjusted according to the modular structure.

6.4 Limitations

The data extraction approach and the visualization proposed within this thesis still face some limitations:

- **Data Processing:** The proposed grammar is only capable of describing selected parts of Draco's knowledge-base. For simplicity reasons, Draco's knowledge-base processing methodology is still used to extract the constraints' weights.
- **Understanding:** The visualization still requires the user to have a minimal understanding of ASP and the knowledge-base behind Draco. Users of the system must be familiar with how Draco encodes information visualization facts and how it builds hard and soft constraints over these facts.
- **Incompleteness:** The visualization only shows Draco's hard and soft constraints and not the whole knowledge-base. The visualization facts and optimization statements, as well as the user-dependent input query, are not depicted by the visualization.
- **Abstraction:** Although the visualization gives an overview of the constraints based on their identifier, categories, and weights, it abstracts their content. As a consequence, the encoded logical expressions can not be inspected directly. The separate constraints inspector designed for this purpose overcomes this issue by directly showing the constraints' formulations in ASP. However, the constraints viewer and the constraints inspector are currently not interactively connected within the prototype.

6.5 Future Work

During the research within this thesis, we identified several possibilities and suggestions for future work in the context of Draco's recommendation system. They concern not only the proposed constraint viewer in the form of the radial visualization but also the whole recommendation process itself, advanced interactions, and the documentation of Draco's knowledge-base. Since there are so many ideas for improvement, we tautly summarized them by the following list:

Knowledge-base viewer

- Summary of knowledge-base features (variables, predicates, numbers, and others).
- Summary of design space definition (mark types, channels, aggregate operations, data types, and others).

- Improve scalability with ideas for radial sunburst visualization/interaction approaches proposed by Stasko and Zhang [SZ00].
- Eliminate confusion between violated constraints and affected connections within the hypergraph.

Recommendation process and recommendation results

- Graphical interface for recommendation query and data set loader with automatic data interpretation and recommendation query generation.
- Clustering of recommendation results by shared encodings or violations to improve the understanding of the results.
- Pin selected recommendations to the top of the recommendation viewer to improve direct comparisons.

Advanced interactions

- Advanced selection options (selection of categories, or one/multiple constraints)
- Explore/show me something else (e.g. different attribute of a constraint than its weight)
- Reconfigure (e.g. swap axis, rotate, rearrange view, panning, sorting items)
- Encode (change representation, switch to a different visualization method)
- Abstract/elaborate (e.g. details on demand, show more information in zoomed level)
- Advanced filter operations (dynamic search query with live data manipulation/highlighting)
- History operations (undo/redo)

Documentation and data processing

- Extend constraints and incorporate features with in-depth documentation and scientific sources.
- Extend the proposed grammar to support all language features and syntax characteristics of the input language to Clingo.

Regarding the conducted evaluation, we believe that especially eliminating the misleading representations would help to make our prototype more comprehensible. Particularly the confusing highlighting of the violations should be targeted in future work. To conclude, all these ideas and their possible implementation should bring us one step closer to a user-centered recommendation tool for data visualization.

CHAPTER 7

Summary and Conclusion

In this thesis, we proposed a novel visualization approach to tackle the visualization constraints of the rule-based recommendation system Draco. The radial-based visualization approach is embedded into a tool that enables visualization experts to inspect Draco's soft and hard constraints. Additionally, violated soft constraints and further interrelations between the system's recommendations and the soft constraints can be interactively explored by selecting or hovering details of interest. We further proposed a hypergraph incorporated into the radial visualization to show shared features by the constraints. This hypergraph enables visualization experts to identify the use and distribution of variables, predicates, and other features – a very time-consuming and challenging task concerning the original raw data of Draco.

To acquire the necessary data for this hypergraph, we developed a data processing architecture that allows us to extract features of interest from Draco's knowledge-base. A central part of this data extraction is our proposed formal EBNF notation of the ASP-Core 2 standard based on W3C. We eliminated faulty components of the original grammar and extended it to language features described in Clingo. An EBNF parser applies this polished grammar and generates an AST of the knowledge-base. The extracted features of interest from this AST are finally stored together with the constraints in a database that serves as the visualization's data source.

The conducted evaluation verifies the capabilities of the elaborated ASP grammar to describe all necessary parts of Draco's knowledge-base. It has been proven by generating an AST of decisive parts of Draco's knowledge-base using the proposed combination of a parser generator with the grammar. Furthermore, the evaluation of the developed prototype has shown that expert users are able to solve identified tasks regarding Draco's knowledge-base and its interrelations with recommendations. The heuristic-value-driven survey gives evidence to the overall value of our visualization approach and confirmed identified difficulties during the user test.

Although there are open limitations and topics to elaborate in future work, we hope our work supports the visualization community in maintaining and extending the rule-based visualization recommendation system Draco.

List of Figures

List of Tables

Listings

Acronyms

ABNF	Augmented Backus-Naur Form. 8, 10
API	Application Programming Interface. 7, 9, 27
ASP	Answer Set Programming. ix, xi, 2, 4, 5, 7–11, 18, 20–30, 32, 46, 49, 53, 54, 56–58, 75, 77, 79, 99
AST	Abstract Syntax Tree. 4, 9, 23, 24, 26, 27, 29–32, 53, 57, 77, 79, 81
BNF	Backus-Naur Form. 8–10, 28, 56
CFG	Context-Free Grammar. 8, 26, 27, 55, 56
CLI	Command-Line Interface. 27
CLP	Constraint Logic Programming. 10, 13
CPU	Central Processing Unit. 57
CSS	Cascading Style Sheets. 53
DOM	Document Object Model. 53
EBNF	Extended Backus-Naur Form. 8–10, 27–29, 53, 56–58, 77, 81, 99
FHD	Full High Definition. 62
HTML	Hypertext Markup Language. 53
IDE	Integrated Development Environment. 11, 18
IndexedDb	Indexed Database. 53
InfoVis	Information Visualization. 19, 59, 63
JSON	JavaScript Object Notation. 7, 53

ML	Machine-Learning. 2, 6
NPM	Node Package Manager. 54
OS	Operating System. 25, 62
POC	Proof of Concept. 57
RankSVM	Support Vector Machine. 7
REGEX	Regular Expression. 9, 23, 26–28, 31, 32
SLD	Selective Linear Definite. 11, 12, 79
SVG	Scalable Vector Graphics. 53
UX	User Experience. 61, 63
VL	Vega-Lite. 7, 8, 79
W3C	World Wide Web Consortium. 28, 29, 77
WASM	WebAssembly. 53

Bibliography

[AAMH13] Bilal Alsallakh, Wolfgang Aigner, Silvia Miksch, and Helwig Hauser. Radial sets: Interactive visual analysis of large overlapping sets. *IEEE Transactions on Visualization and Computer Graphics*, 19(12):2496–2505, 2013.

[ACC+17] Farid Alborzi, Surajit Chaudhuri, Rada Chirkova, Pallavi Deo, Christopher Healey, Gargi Pingale, Juan Reutter, and Vaira Selvakani. Dataslicer: Task-based data selection for visual data exploration. *arXiv preprint arXiv:1703.09218*, 2017.

[ACJ+13] Thomas Ambroz, Günther Charwat, Andreas Jusits, Johannes Peter Wallner, and Stefan Woltran. Arvis: Visualizing relations between answer sets. In *Proceedings of the International Conference on Logic Programming and Nonmonotonic Reasoning*, pages 73–78. Springer, 2013.

[Ada09] Yoshihiro Adachi. Prolog visualization system using logichart diagrams. *arXiv preprint arXiv:0903.2207*, 2009.

[AF07] Yoshihiro Adachi and Yudai Furusawa. Logichart: A prolog program diagram and its layout. *Electronic Communications of the EASST*, 7, 2007.

[ATIY00] Yoshihiro Adachi, Kensei Tsuchida, Takanori Imaki, and Takeo Yaku. Logichart—intelligible program diagram for prolog and its processing system. *Electronic Notes in Theoretical Computer Science*, 30(4):276–288, 2000.

[BBW12] Michael Burch, Fabian Beck, and Daniel Weiskopf. Radial edge splatting for visualizing dynamic directed graphs. In *GRAPP/IVAPP*, pages 603–612, 2012.

[BET11] Gerhard Brewka, Thomas Eiter, and Mirosław Truszczyński. Answer set programming at a glance. *Communications of the ACM*, 54(12):92–103, 2011.

[BRBF14a] Jeremy Boy, Ronald A Rensink, Enrico Bertini, and Jean-Daniel Fekete. A principled way of assessing visualization literacy. *IEEE Transactions on Visualization and Computer Graphics*, 20(12):1963–1972, December 2014. Publisher Copyright: © 2014 IEEE.

[BRBF14b] Jeremy Boy, Ronald A. Rensink, Enrico Bertini, and Jean-Daniel Fekete. A principled way of assessing visualization literacy. *IEEE Transactions on Visualization and Computer Graphics*, 20(12):1963–1972, 2014.

[Car99] Mackinlay Card. *Readings in information visualization: using vision to think*. Morgan Kaufmann, 1999.

[Cas91] Stephen M. Casner. Task-analytic approach to the automated design of graphic presentations. *ACM Trans. Graph.*, 10(2):111–151, April 1991.

[CC05] Brock Craft and Paul Cairns. Beyond guidelines: what can we learn from the visual information seeking mantra? In *Proceedings of the 9th International Conference on Information Visualisation*, IV '05, pages 110–118, 2005.

[CDMP19] Théo Coulin, Maxence Detante, William Mouchère, and Fabio Petrillo. Software architecture metrics: a literature review. *CoRR*, abs/1901.09050, 2019.

[CFG$^+$12] Francesco Calimeri, Wolfgang Faber, Martin Gebser, Giovambattista Ianni, Roland Kaminski, Thomas Krennwallner, Nicola Leone, Francesco Ricca, and Torsten Schaub. Asp-core-2: Input language format. *ASP Standardization Working Group*, 2012.

[CGDLBMM03] M. Cameron, M. García De La Banda, Kim Marriott, and Peter Moulder. Vimer: a visual debugger for mercury. In *Proceedings of the 5th ACM International Conference on Principles and Practice of Declaritive Programming*, pages 56–66, 2003.

[Cho56] Noam Chomsky. Three models for the description of language. *IRE Trans. Inf. Theory*, 2:113–124, 1956.

[CIR11] Francesco Calimeri, Giovambattista Ianni, and Francesco Ricca. Third asp competition-file and language formats. Technical report, Tech. rep., Universita della Calabria, 2011.

[CM03] William F. Clocksin and Christopher S. Mellish. *Programming in Prolog*. Springer Science & Business Media, 2003.

[CVBP08] Owen Cliffe, Marina De Vos, Martin Brain, and Julian Padget. Aspviz: Declarative visualisation and animation using answer set programming. In *Proceedings of the International Conference on Logic Programming*, pages 724–728, Berlin, Heidelberg, 2008. Springer.

[DD19] Victor Dibia and Çağatay Demiralp. Data2vis: Automatic generation of data visualizations using sequence-to-sequence recurrent neural networks. *IEEE Computer Graphics and Applications*, 39(5):33–46, 2019.

[DNK97] Yannis Dimopoulos, Bernhard Nebel, and Jana Koehler. Encoding planning problems in nonmonotonic logic programs. In *Proceedings of the European Conference on Planning*, pages 169–181. Springer, 1997.

[ELCM09] Sebastián Escarza, Martín Leonardo Larrea, Silvia Mabel Castro, and Sergio R. Martig. Delp viewer: a defeasible logic programming visualization tool. In *XV. Congreso Argentino de Ciencias de la Computación*, 2009.

[ESC16] Humaira Ehsan, Mohamed A. Sharaf, and Panos K. Chrysanthis. Muve: Efficient multi-objective view recommendation for visual data exploration. In *Proceedings of the 32nd IEEE International Conference on Data Engineering*, ICD '16, pages 731–742, 2016.

[Few09] Stephen Few. *Now you see it: simple visualization techniques for quantitative analysis*. Analytics Press, USA, 1st edition, 2009.

[FRR11] Onofrio Febbraro, Kristian Reale, and Francesco Ricca. Aspide: Integrated development environment for answer set programming. In *Logic Programming and Nonmonotonic Reasoning*, pages 317–330, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[GKK$^+$08a] Martin Gebser, R. Kaminiski, Benjamin Kaufmann, M. Ostrowsky, Torsten Schaub, and Sven Thiele. Using gringo, clingo and iclingo, 2008.

[GKK$^+$08b] Martin Gebser, Roland Kaminski, Benjamin Kaufmann, Max Ostrowski, Torsten Schaub, and Sven Thiele. A user's guide to gringo, clasp, clingo, and iclingo. 2008.

[Gna81] Sakunthala Gnanamgari. Information presentation through default displays. 1981.

[GW09] David Gotz and Zhen Wen. Behavior-driven visualization recommendation. In *Proceedings of the 14th International Conference on Intelligent User Interfaces*, IUI '09, page 315–324, New York, NY, USA, 2009. Association for Computing Machinery.

[HBL$^+$19] Kevin Hu, Michiel A. Bakker, Stephen Li, Tim Kraska, and César Hidalgo. VizML: A Machine Learning Approach to Visualization Recommendation. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, CHI '19, pages 1–12, New York, NY, USA, 2019. Association for Computing Machinery.

[HEAE16] Shah Rukh Humayoun, Hafez Ezaiza, Ragaad AlTarawneh, and Achim Ebert. Social-circles exploration through interactive multi-layered chord layout. In *Proceedings of the International Working Conference on Advanced Visual Interfaces*, AVI '16, page 314–315, New York, NY, USA, 2016. Association for Computing Machinery.

[HOH18] Kevin Hu, Diana Orghian, and César Hidalgo. Dive: A mixed-initiative system supporting integrated data exploration workflows. In *Proceedings of the Workshop on Human-In-the-Loop Data Analytics*, HILDA '18, New York, NY, USA, 2018. Association for Computing Machinery.

[Hol06] Danny Holten. Hierarchical edge bundles: Visualization of adjacency relations in hierarchical data. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):741–748, 2006.

[KFD19] Petra Kubernátová, Magda Friedjungová, and Max van Duijn. Constructing a Data Visualization Recommender System. In Christoph Quix and Jorge Bernardino, editors, *Data Management Technologies and Applications*, pages 1–25, Cham, 2019. Springer International Publishing.

[KHPA12] Alicia Key, Bill Howe, Daniel Perry, and Cecilia Aragon. VizDeck: Self-Organizing Dashboards for Visual Analytics. In *Proceedings of the ACM International Conference on Management of Data*, SIGMOD '12, pages 681–684, New York, NY, USA, 2012. Association for Computing Machinery.

[KJ13] Andreas Kerren and Ilir Jusufi. A novel radial visualization approach for undirected hypergraphs. In *Proceedings of the EG/VGTC Conference on Visualization (Short Papers)*, EuroVis '13, 2013.

[KO17] Pawandeep Kaur and Michael Owonibi. A review on visualization recommendation strategies. In *Proceedings of the 12th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications - Volume 3: IVAPP, (VISIGRAPP 2017)*, pages 266–273. INSTICC, SciTePress, 2017.

[KS13] Arne König and Torsten Schaub. Monitoring and visualizing answer set solving. *Theory and Practice of Logic Programming*, 13:4–5, 2013.

[KSFN08] Andreas Kerren, John Stasko, Jean-Daniel Fekete, and Chris North. *The Value of Information Visualization*, pages 1–18. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

[Lew82] Clayton Lewis. *Using the" thinking-aloud" method in cognitive interface design.* IBM TJ Watson Research Center Yorktown Heights, 1982.

[LHT17] Antoine Lhuillier, Christophe Hurter, and Alexandru Telea. State of the art in edge and trail bundling techniques. *Computer Graphics Forum*, 36(3):619–645, 2017.

[Lif99] Vladimir Lifschitz. Action languages, answer sets, and planning. In *The Logic Programming Paradigm*, pages 357–373. Springer, 1999.

[LQTL18] Yuyu Luo, Xuedi Qin, Nan Tang, and Guoliang Li. Deepeye: Towards automatic data visualization. In *Proceedings of the 34th IEEE International Conference on Data Engineering*, ICD '18, pages 101–112. IEEE, 2018.

[MA14] Silvia Miksch and Wolfgang Aigner. A matter of time: Applying a data–users–tasks design triangle to visual analytics of time-oriented data. *Computers & Graphics*, 38:286–290, 2014.

[Mac86] Jock Mackinlay. Automating the Design of Graphical Presentations of Relational Information. *ACM Transactions on Graphics*, 5(2):110–141, apr 1986.

[Maz09] Riccardo Mazza. *Introduction to information visualization.* Springer Science & Business Media, 2009.

[MHS07] Jock Mackinlay, Pat Hanrahan, and Chris Stolte. Show me: Automatic presentation for visual analysis. *IEEE Transactions on Visualization and Computer Graphics*, 13(6):1137–1144, 2007.

[MT99] Victor W. Marek and Miroslaw Truszczyński. Stable models and an alternative logic programming paradigm. In *The Logic Programming Paradigm*, pages 375–398. Springer, 1999.

[Mun09] Tamara Munzner. A nested model for visualization design and validation. *IEEE Transactions on Visualization and Computer Graphics*, 15(6):921–928, 2009.

[MWN$^+$19] Dominik Moritz, Chenglong Wang, Greg L. Nelson, Halden Lin, Adam M. Smith, Bill Howe, and Jeffrey Heer. Formalizing Visualization Design Knowledge as Constraints: Actionable and Extensible Models in Draco. *IEEE Transactions on Visualization and Computer Graphics*, 25(1):438–448, 2019.

[Nie92] Jakob Nielsen. Finding usability problems through heuristic evaluation. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '92, page 373–380, New York, NY, USA, 1992. Association for Computing Machinery.

[Nie16] Frank Nielsen. Hierarchical clustering. In *Introduction to HPC with MPI for Data Science*, pages 195–211. Springer, 2016.

[NMSL19] Carolina Nobre, Miriah Meyer, Marc Streit, and Alexander Lex. The state of the art in visualizing multivariate networks. *Computer Graphics Forum*, 38(3):807–832, 2019.

[OPT13] Johannes Oetsch, Jörg Pührer, and Hans Tompits. The sealion has landed: An ide for answer-set programming—preliminary report. In Hans Tompits, Salvador Abreu, Johannes Oetsch, Jörg Pührer, Dietmar Seipel, Masanobu Umeda, and Armin Wolf, editors, *Applications of Declarative Programming and Knowledge Management*, pages 305–324, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[RKMG94] Steven F. Roth, John Kolojejchick, Joe Mattis, and Jad Goldstein. Interactive graphic design using automatic presentation knowledge. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '94, page 112–117, New York, NY, USA, 1994. Association for Computing Machinery.

[SCD$^+$00] Helmut Simonis, Trijntje Cornelissens, Véronique Dumortier, Giovanni Fabris, F. Nanni, and Adriano Tirabosco. *Using Constraint Visualisation Tools*, pages 321–356. Springer Berlin Heidelberg, Berlin, Heidelberg, 2000.

[Sch11] Hans-Jorg Schulz. Treevis.net: A tree visualization reference. *IEEE Computer Graphics and Applications*, 31(6):11–15, 2011.

[SDS12] João Miguel Santos, Paulo Dias, and Beatriz Sousa Santos. Implementation and evaluation of an enhanced h-tree layout pedigree visualization. In *Proceedings of the 16th International Conference on Information Visualisation*, IV '12, pages 24–29, 2012.

[SH13] Hans-Jörg Schulz and Christophe Hurter. Grooming the hairball-how to tidy up network visualizations? In *Proceedings of the IEEE Information Visualization Conference*, INFOVIS '13, 2013.

[SHH03] Dietmar Seipel, Marbod Hopfner, and Bernd Heumesser. Analyzing and visualizing prolog programs based on xml-representations. In *Proceedings of the 13th International Workshop on Logic Programming environments*. Citeseer, 2003.

[Shn03] Ben Shneiderman. The eyes have it: A task by data type taxonomy for information visualizations. In *The Craft of Information Visualization*, Interactive Technologies, pages 364–371. Morgan Kaufmann, San Francisco, 2003.

[Sip97] Michael Sipser. Introduction to the theory of computation. *Thomson Course Technology, Boston, MA, USA*, 1997.

[SM00] Heidrun Schumann and Wolfgang Müller. *Visualisierung - Grundlagen und allgemeine methoden*. Springer-Verlag, Berlin Heidelberg New York, 2000.

[SML$^+$18] Bahador Saket, Dominik Moritz, Halden Lin, Victor Dibia, Çagatay Demiralp, and Jeffrey Heer. Beyond heuristics: Learning visualization design. *CoRR*, abs/1807.06641, 2018.

[SMWH17] Arvind Satyanarayan, Dominik Moritz, Kanit Wongsuphasawat, and Jeffrey Heer. Vega-lite: A grammar of interactive graphics. *IEEE Transactions on Visualization and Computer Graphics*, 23:341–350, 2017.

[SN98] Timo Soininen and Ilkka Niemelä. *Formalizing configuration knowledge using rules with choices*. Helsinki University of Technology, 1998.

[Spe07] Robert Spence. *Information Visualization: Design for Interactions*. Pearson Education Limited, Essex, UK, 2nd edition, 2007.

[SS05] Jinwook Seo and Ben Shneiderman. A rank-by-feature framework for interactive exploration of multidimensional data. *Information visualization*, 4(2):96–113, 2005.

[SS06] Hans-Jörg Schulz and Heidrun Schumann. Visualizing graphs - a generalized view. In *Proceeding of the 10th International Conference on Information Visualisation*, IV '06, pages 166–173, 2006.

[SZ00] John Stasko and Eugene Zhang. Focus+ context display and navigation techniques for enhancing radial, space-filling hierarchy visualizations. In *Proceedings of the IEEE Symposium on Information Visualization*, INFOVIS '00, pages 57–65. IEEE, 2000.

[TKE12] Raga'ad M. Tarawaneh, Patric Keller, and Achim Ebert. A general introduction to graph visualization techniques. In *Visualization of Large and Unstructured Data Sets: Applications in Geospatial Planning, Modeling and Engineering-Proceedings of IRTG 1131 Workshop 2011*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2012.

[VBSW13] Corinna Vehlow, Michael Burch, Hansjorg Schmauder, and Daniel Weiskopf. Radial layered matrix visualization of dynamic graphs. In *Proceedings of the 17th International Conference on Information Visualisation*, IV '13, pages 51–58, 2013.

[VBW15] Corinna Vehlow, Fabian Beck, and Daniel Weiskopf. The state of the art in visualizing group structures in graphs. In *Proceedings of the EG/VGTC Conference on Visualization (STARs)*, EuroVis '15, pages 21–40, 2015.

[VBW17] Corinna Vehlow, Fabian Beck, and Daniel Weiskopf. Visualizing group structures in graphs: A survey. In *Computer Graphics Forum*, volume 36, pages 201–225. Wiley Online Library, 2017.

[VHS+17] Manasi Vartak, Silu Huang, Tarique Siddiqui, Samuel Madden, and Aditya Parameswaran. Towards Visualization Recommendation Systems. *Proceedings of the ACM International Conference on Management of Data*, 45(4):34–39, 2017.

[VRM+15] Manasi Vartak, Sajjadur Rahman, Samuel Madden, Aditya Parameswaran, and Neoklis Polyzotis. Seedb: Efficient data-driven visualization recommendations to support visual analytics. In *Proceedings of the VLDB Endowment International Conference on Very Large Data Bases*, volume 8, page 2182. NIH Public Access, 2015.

[VWS+18] Fernanda Viégas, Martin Wattenberg, Daniel Smilkov, James Wexler, and Daniel Gundrum. Generating charts from data in a data table, 2018. US 20180088753 A1.

[Wal08] Guenter Wallner. Force directed embedding of hierarchical cluster graphs. In *Proceedings of the International Conference on Relations*, ROGICS '08, 2008.

[WAM+19] Emily Wall, Meeshu Agnihotri, Laura Matzen, Kristin Divis, Michael Haass, Alex Endert, and John Stasko. A heuristic approach to value-driven evaluation of visualizations. *IEEE Transactions on Visualization and Computer Graphics*, 25(1):491–500, 2019.

[Whe20] David Wheeler. Don't use iso/iec 14977 extended backus-naur form (ebnf), 2020.

[Wit09] Johan Wittocx. IDPDraw, a tool used for visualizing answer sets. 2009.

[WMA+16a] Kanit Wongsuphasawat, Dominik Moritz, Anushka Anand, Jock Mackinlay, Bill Howe, and Jeffrey Heer. Towards a General-Purpose Query Language for Visualization Recommendation. In *Proceedings of the Workshop on Human-In-the-Loop Data Analytics*, HILDA '16, New York, NY, USA, 2016. Association for Computing Machinery.

[WMA+16b] Kanit Wongsuphasawat, Dominik Moritz, Anushka Anand, Jock Mackinlay, Bill Howe, and Jeffrey Heer. Voyager: Exploratory Analysis via Faceted Browsing of Visualization Recommendations. *IEEE Transactions on Visualization and Computer Graphics*, 22(1):649–658, 2016.

[WQM+17] Kanit Wongsuphasawat, Zening Qu, Dominik Moritz, Riley Chang, Felix Ouk, Anushka Anand, Jock Mackinlay, Bill Howe, and Jeffrey

Heer. Voyager 2: Augmenting Visual Analysis with Partial View Specifications. In *Proceedings of the CHI Conference on Human Factors in Computing Systems*, CHI '17, pages 2648–2659, New York, NY, USA, 2017. Association for Computing Machinery.

[WWDW06] Weixin Wang, Hui Wang, Guozhong Dai, and Hongan Wang. *Visualization of Large Hierarchical Data by Circle Packing*, page 517–520. Association for Computing Machinery, New York, NY, USA, 2006.

[WYM19] Linda Woodburn, Yalong Yang, and Kim Marriott. Interactive visualisation of hierarchical quantitative data: an evaluation. In *Proceedings of the IEEE Visualization Conference*, VIS '19, pages 96–100. IEEE, 2019.

[Yeh80] Amiram Yehudai. The decidability of equivalence for a family of linear grammars. *Information and Control*, 47(2):122–136, 1980.

[YKSJ07] Ji Soo Yi, Youn ah Kang, John Stasko, and Julie A. Jacko. Toward a deeper understanding of the role of interaction in information visualization. *IEEE Transactions on Visualization and Computer Graphics*, 13(6):1224–1231, 2007.

[ZBA$^+$20] Annamaria Zoppini, Lucia Bongiorni, Nicoletta Ademollo, Luisa Patrolecco, Tamara Cibic, Annalisa Franzo, Marco Melita, Matteo Bazzaro, and Stefano Amalfitano. Bacterial diversity and microbial functional responses to organic matter composition and persistent organic pollutants in deltaic lagoon sediments. *Estuarine, Coastal and Shelf Science*, 233:106508, 2020.

[ZF01] Michelle Xhou Zhou and Steven K. Feiner. Improvise: Automated generation of animated graphics for coordinated multimedia presentations. In *Cooperative Multimodal Communication*, pages 43–63. Springer Berlin Heidelberg, 2001.

Appendix

ASP Grammar

Listing L1: Modified ASP grammar of *ASP-CORE-2* standard in EBNF notation

```
program                ::= statement⋆ query?

statement              ::= rule | fact | integrity_constraint | comment | NL
query                  ::= classical_literal QUERY_MARK

fact                   ::= head DOT WS⋆ NL?
rule                   ::= head WS⋆ CONS WS⋆ bodies? DOT WS⋆ NL?
integrity_constraint ::= CONS WS⋆ bodies? DOT WS⋆ NL? |
                         WCONS WS⋆ bodies? DOTSQUARE_O weight_at_level
                         SQUARE_C WS⋆ NL?
comment                ::= MULTI_LINE_COMMENT WS⋆ NL? | COMMENT WS⋆ NL?

head                   ::= disjunction | choice
bodies                 ::= body+ | FALSE | TRUE
body                   ::= naf_literal WS⋆ COMMA WS⋆ |
                         NAF? WS⋆ aggregate WS⋆ COMMA WS⋆ |
                         naf_literal WS⋆ | NAF? WS⋆ aggregate WS⋆

disjunction            ::= classical_literal WS⋆ (PIPE WS⋆ disjunction)? WS⋆

choice                 ::= (term WS⋆ binop?)? WS⋆ CURLY_O WS⋆ choice_elements?
                         WS⋆ CURLY_C WS⋆ (binop? WS⋆ term)? WS⋆
choice_elements        ::= choice_element WS⋆
                         (SEMICOLON WS⋆ choice_elements)? WS⋆
choice_element         ::= classical_literal WS⋆ (COLON WS⋆ naf_literals?)? WS⋆

aggregate              ::= (term WS⋆ binop?)? WS⋆ agg_function? WS⋆ CURLY_O WS⋆
                         agg_elements? WS⋆ CURLY_C WS⋆
                         (binop? WS⋆ term)? WS⋆ | aggreagte_count_func WS⋆
agg_elements           ::= agg_element ((COMMA | SEMICOLON) WS⋆
                         agg_elements)?
agg_element            ::= classical_literal? WS⋆ (COLON WS⋆ naf_literals)?
                         WS⋆ | basic_terms? WS⋆ (COLON WS⋆ naf_literals)? WS⋆
agg_function           ::= AGGREGATE_COUNT | AGGREGATE_MAX |
                         AGGREGATE_MIN | AGGREGATE_SUM
aggreagte_count_func ::= PIPE WS⋆ agg_elements? WS⋆ PIPE
```

```
                         WS* (binop? WS* term)? WS*

weight_at_level      ::= term WS* (AT WS* term)? WS* (COMMA WS* terms)? WS*

naf_literals         ::= naf_literal WS* (COMMA WS* naf_literals)? WS*
naf_literal          ::= NAF? WS* classical_literal WS* |
                         NAF? WS* builtin_atom WS*

predicate            ::= ID
classical_literal    ::= MINUS? WS* predicate
                         (PAREN_O WS* terms? WS* PAREN_C) WS* |
                         MINUS? WS* ID WS*
builtin_atom         ::= term WS* binop WS* term WS*

binop                ::= EQUAL | UNEQUAL | LESS_OR_EQ | GREATER_OR_EQ |
                         LESS | GREATER

terms                ::= term ((COMMA | SEMICOLON) WS* terms)? WS*
term_op              ::= arithop WS* term WS* term_op? WS*
term                 ::= NUMBER WS* term_op? WS* |
                         STRING WS* term_op? WS* |
                         VARIABLE WS* term_op? WS* |
                         ANONYMOUS_VARIABLE WS* term_op? WS* |
                         PAREN_O WS* terms? WS* PAREN_C WS* term_op? WS* |
                         MINUS WS* term WS* term_op? WS* |
                         ID (PAREN_O WS* terms? WS* PAREN_C)?
                         WS* term_op? WS*

basic_terms          ::= basic_term WS* (COMMA WS* basic_terms)? WS*
basic_term           ::= ground_term WS* | variable_term WS*
ground_term          ::= SYMBOLIC_CONSTANT | STRING | MINUS? NUMBER
variable_term        ::= VARIABLE | ANONYMOUS_VARIABLE
arithop              ::= PLUS | MINUS | TIMES | DIV

NUMBER               ::= "0" | [1-9] [0-9]*
SYMBOLIC_WITHOUT_STAR ::= [^(a-zA-Z0-9_)]
SYMBOLIC_CONSTANT    ::= "*" | SYMBOLIC_WITHOUT_STAR
UNDERLINE            ::= "_"
DOUBLE_QUOTE         ::= '"'
ID                   ::= [a-z] [a-zA-Z0-9_]*
VARIABLE             ::= [A-Z] [a-zA-Z0-9_]*
STRING_CONTENT       ::= ([#x20-#x21] | [#x23-#x5B] | [#x5D-#xFFFF]) |
                         #x5C (#x22 | #x5C | #x2F | #x62 | #x66 | #x6E |
                         #x72 | #x74 | #x75 HEXDIG HEXDIG HEXDIG HEXDIG)
STRING               ::= DOUBLE_QUOTE ([^#x22] | #x5c #x22)* DOUBLE_QUOTE
HEXDIG               ::= [a-zA-Z0-9]
ANONYMOUS_VARIABLE   ::= UNDERLINE ID | UNDERLINE VARIABLE | UNDERLINE
DOT ::= "."
COMMA ::= ","
QUERY_MARK ::= "?"
COLON ::= ":"
SEMICOLON ::= ";"
PIPE ::= "|"
```

```
NAF ::= "not"
CONS ::= ":-"
WCONS ::= ":~"
PLUS ::= "+"
MINUS ::= "-"
TIMES ::= "*"
DIV ::= "/"
AT ::= "@"
PAREN_O ::= "("
PAREN_C ::= ")"
SQUARE_O ::= "["
SQUARE_C ::= "]"
CURLY_O ::= "{"
CURLY_C ::= "}"
EQUAL ::= "="
UNEQUAL ::= "<>" | "!="
LESS ::= "<"
GREATER ::= ">"
LESS_OR_EQ ::= "<="
GREATER_OR_EQ ::= ">="
AGGREGATE_COUNT ::= "#count"
AGGREGATE_MAX ::= "#max"
AGGREGATE_MIN ::= "#min"
AGGREGATE_SUM ::= "#sum"
FALSE ::= "#false"
TRUE ::= "#true"
COMMENT ::= "%" ([^*\n] [^\n]*)?
MULTI_LINE_COMMENT ::= "%*" ([^*] | "*" [^%])* "*%"
NL ::= [#x0A#x0D]
WS ::= [#x09#x20]
```

Value Evaluation Statements

Insight	The visualization facilitates answering questions about the data	The visualization exposes individual data cases and their attributes	S1
		The visualization facilitates perceiving relationships in the data like patterns & distributions of the variables	S2
		The visualization promotes exploring relationships between individual data cases as well as different groupings of data cases	S3
	The visualization provides a new or better understanding of the data	The visualization helps generate data-driven questions	S4
		The visualization helps identify unusual or unexpected, yet valid, data characteristics or values	S5
	The visualization provides opportunities for serendipitous discoveries	The visualization provides useful interactive capabilities to help investigate the data in multiple ways	S6
		The visualization shows multiple perspectives about the data	S7
		The visualization uses an effective representation of the data that shows related and partially related data cases	S8
Time	The visualization affords rapid parallel comprehension for efficient browsing	The visualization provides a meaningful spatial organization of the data	S9
		The visualization shows key characteristics of the data at a glance	S10
	The visualization provides mechanisms for quickly seeking specific information	The interface supports using different attributes of the data to reorganize the visualization's appearance	S11
		The visualization supports smooth transitions between different levels of detail in viewing the data	S12
		The visualization avoids complex commands and textual queries by providing direct interaction with the data representation	S13
Essence	The visualization provides a big picture perspective of the data	The visualization provides a comprehensive and accessible overview of the data	S14
		The visualization presents the data by providing a meaningful visual schema	S15
	The visualization provides an understanding of the data beyond individual data cases	The visualization facilitates generalizations and extrapolations of patterns and conclusions	S16
		The visualization helps understand how variables relate in order to accomplish different analytic tasks	S17
Confidence	The visualization helps avoid making incorrect inferences	The visualization uses meaningful and accurate visual encodings to represent the data	S18
		The visualization avoids using misleading representations	S19
	The visualization facilitates learning more broadly about the domain of the data	The visualization promotes understanding data domain characteristics beyond the individual data cases and attributes	S20
	The visualization helps understand data quality	If there were data issues like unexpected, duplicate, missing, or invalid data, the visualization would highlight those issues	S21

Table A1: Value evaluation statements - reprinted from Wall et al. [WAM+19]