

Inspecting Data Using Natural Language Queries

Franscesca Bacci, Federico Maria Cau, and Lucio Davide Spano(✉)

Department of Mathematics and Computer Science,
University of Cagliari, Cagliari, Italy
davide.spano@unica.it

Abstract. In this paper, we discuss a simple architecture for supporting the inspection of a generic dataset using natural language queries. We show how to integrate modern Artificial Intelligence libraries in the system and how to derive chart visualization out of the user's intent. The result is a lightweight architecture for supporting such natural language queries in web-based visualization tools. Finally, we report on the user evaluation of the interface, showing a good acceptance and effectiveness of the proposed approach.

Keywords: Data visualization · Natural language interaction · Artificial Intelligence · Human computer interaction

1 Introduction

The interfaces for Information Visualization (InfoVis) usually exploit a direct manipulation paradigm for selecting, filtering and inspecting data. The graphical modality supports the trial and error and the user learns and drives conclusions out the data during the inspection process. The paradigm applies also to the visualization *construction*, supporting the user in selecting which attributes to visualize and their layout. Such operations in the graphical modality are quite tedious and repetitive, so the research focused on exploiting Natural Language (NL) queries for building visualizations. Requests in NL are quick to express and they proved quite easy to create for simple queries [6]. Different results are available in the literature (see Sect. 2), but they all require complex NL analysis techniques whose application effort often exceeds the benefits in most visualization tools. In this paper, we propose a simple architecture for supporting NL queries in visualization tools exploiting a standard chatbot API. Reducing the development complexity we aim at a widespread inclusion of NL queries in InfoVis tools.

The work has been founded by the AI4fit project (POR-FSER Sardegna 2014–2020, CUP: F25D17000200008).

© Springer Nature Switzerland AG 2020
O. Gervasi et al. (Eds.): ICCSA 2020, LNCS 12254, pp. 771–782, 2020.
https://doi.org/10.1007/978-3-030-58817-5_55

2 Related Work

The quest for NL queries on databases has a history [6], difficult to summarise entirely here. We will focus on the work that most influenced our system design. Eviza [8] proposes a solution for enabling users to interact with existing visualizations using NL queries. Through open-ended questions, the user enhances the current visualization, tailoring it for specific purposes. The solution exploits a probabilistic grammar designed for the tool and finite state machines for driving the conversation. Evizeon [4] further develops the abilities of the previous system introducing multimodal conversations with the user. The system resolves the language ambiguities in the user's query through further questions and/or picking values in the current visualization. Once the system reaches a sufficient comprehension of the current request, creates or updates the visualization.

Recently, Setlur et al. [9] discussed a system for resolving partial utterances through syntactic and semantic constraints. They construct a heuristic for creating a manageable solution space and apply logical ranking for selecting the best candidate. Such inference allows the user to express the query in a much more natural way, including ambiguities and ellipsis that are managed by the system.

FlowSense [11] is a natural language interface that utilizes the latest natural language processing techniques to assist dataflow diagram construction. The system employs a semantic parser that expands the variables in the grammar to match the input query, exploiting special utterance tagging and placeholders to support progressive construction of dataflow diagrams that deal with various types of datasets.

Siwei Fu et al. propose Quda [3], a dataset which helps in the design of visualization-oriented natural language interfaces (V-NLIs). In this work, they present the design and implementation of a V-NLI prototype, called FreeNLI, that uses a language parser to process the query combined with a pool of design rules to provide useful charts and tables to the end-user.

Saktheeswaran et al. [7] conducted a qualitative user study in the context of a network visualization tool using a modified version of the Orko system [10], enhancing the multimodal aspect of the interface and types of interaction.

The system we propose leverages on different ideas proposed in this field. In opposition to more complex solutions, it exploits a general-purpose chatbot library for the natural language interpretation and relies on a partially automatic training, which simplifies the adoption of NL for performing queries. Besides, it splits complex requests into sub-queries, allowing the user to iteratively modify and delete them for obtaining the desired result. Finally, we combine the graphical and the NL modality for managing the data filtering.

3 System Architecture

In this section, we discuss the overall system architecture, describing the different components that cooperate for producing the chart visualization starting from an NL query. Figure 1 depicts the high-level solution components. The user, through

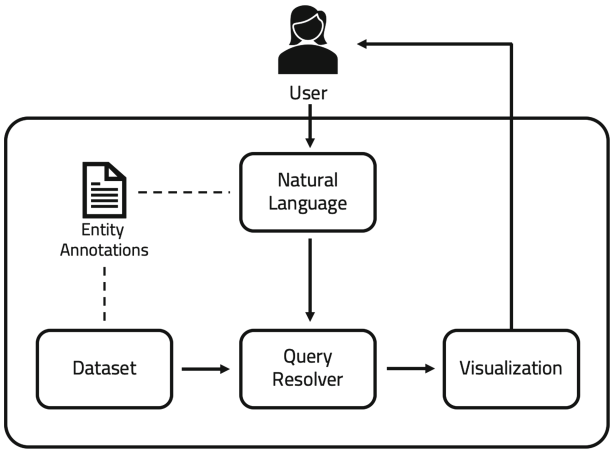

Fig. 1. Overall system architecture

a web user interface, inserts the query in a text field. The *Natural Language* component analyses the text for interpreting the user's intent. It takes as input the query and returns an object describing the entities involved and how the user wants to visualize them (the intent). This component relies on an Artificial Intelligence (AI) library for the natural language processing, Wit.ai[1]. The library requires training for understanding properly the specific queries in the current domain. In our case, the training has two levels of abstraction. At the higher level, we have the samples teaching the AI to recognize the intents for asking a specific visualization, such as *showing*, *ordering*, *correlating* etc. The wording used for requesting the visualization does not depend on the current domain or dataset, so they can be reused for different applications.

To build a proper visualization, the AI must recognize also the entities involved. Their names strongly depend on the considered dataset. Since seldom an entity identifier in a dataset is the name people would use for addressing it, we included an *Entity Annotation* file that maps the entities to one or more friendly names. It includes also other useful metadata, such as the abstract data type associated with each attribute (e.g., date, categorical, ordinal or quantitative) or human-readable labels for their values. We use the annotation file for generating some annotated queries in natural language for training the AI library on the four templates we introduce in Sect. 4. In this way, we have auto-generated samples exploiting the entities involved in a particular dataset, without requesting developers to manually perform the training. However, they can still add fine-tuned NL queries to the generated set. For instance, Wit.ai notifies developers when it is not able to correctly understand a user's sentence. In such a case, developers will need to enter the label manually.

[1] https://wit.ai/.

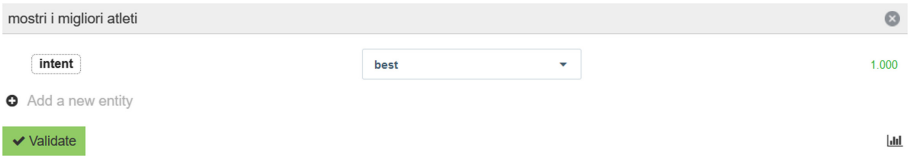

Fig. 2. Wit.ai training phase

At runtime, when the *Natural Language* component correctly identified the user's intent and the involved entities, the *Query Resolver* component reads, filters and sorts the required data from the *Dataset*, preparing them for the *Visualization* component, which generates the graphic rendering according to the user's intent. The current implementation exploits *Python* libraries for the data manipulation and D3.js[2] for the visualization.

3.1 Wit.ai Training

We will now see an example of how we can train Wit.ai to recognize the meaning of a sentence. Suppose we want to teach the Wit.ai engine how to interpret the clause "*Show me the best athletes*": we will insert the latter in the *Understanding* tab of the engine making sure that it learns to associate the word "*best*" as a tag. Once this new behavior is learned, we will insert once more the sentence seen above and Wit.ai will return an intent with value "*best*" as shown in Fig. 2.

After the completion of the training step, we will query the tool through the Django framework via Python typing the previous sentence in the search bar. As a response, Wit.ai will return a JSON object whose structure is shown in Fig. 3.

On the Python side, we will check the *value* field of the intent object to understand what type of query to perform on the dataset. In this case, the behaviour of the Python code will create a query that will filter the dataset by selecting the athletes who have the highest average score and the corresponding unique code, ordering them in descending order. With the values obtained, we will draw through D3 the appropriate chart for the type of query, which in this case is the bar chart: we will insert the athlete code in the ordinates and the average of the marks of each in the abscissas, as shown in Fig. 4. Meanwhile, taking as an example the statement "*order the athletes' ratings*", the training procedure will be slightly different: we will set the word "order" as intent and the word "*ratings*" as an entity to get the athletes ratings. So again we have only one intent in the JSON object and also an entity to check, with the resulting visualization shown in Fig. 6.

In general, the intent is controlled first and then the various entities in order.

[2] https://d3js.org/.

```
{
  '_text': 'mostra i migliori atleti',
  'outcomes': [
    {
      'confidence': None,
      'intent': 'default_intent',
      '_text': 'mostra i migliori atleti',
      'entities': {
        'intent': [
          {
            'confidence': 0.99984855184298,
            'value': 'best'
          }
        ]
      }
    }
  ]
}
```


Fig. 3. Resulting JSON file from the query.



Fig. 4. Showing the best athletes ordered by ranking

4 Querying Datasets in Natural Language

In this section, we discuss different sample queries and how the proposed tool manages it. We use a running workout dataset, which contains different training sessions by both professional and amateur running athletes [2]. Each session includes the athlete's identifier, his/her age, the session date, the calories spent, the pace, the beats per minute and a 1 to 5 performance rating. A possible usage scenario for the following examples is that of a coach that inspects the athlete's performances for providing guidance.

At the beginning of the interaction, the interface shows a simple search bar waiting for the user's query. When the user confirms his/her input, the tool displays the generated visualization. Besides, it includes a button for showing and hiding the list of the data attributes in the dataset. The search bar is provided with an autocomplete function, which shows suggestions for the current query,

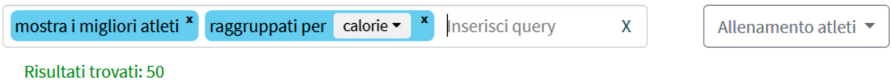

Fig. 5. Subqueries visualization

according to the templates used for training the AI component. As usual, the tool displays the autocompletion in a drop-down menu under the search bar. In addition, the tool splits the user's input into subqueries, i.e. parts of the same query that the user can interactively modify fir reaching the desired visualization. Subqueries separate the main user's intent (such as e.g., showing the athletes' rating) from aggregation and/or ordering functions. Figure 5 shows how the tool displays subqueries, using boxes grouping the input string. Each one has a button for removing the subquery and updating the visualization accordingly. In addition, the values of categorical attributes are displayed as drop-down menus for rapidly changing the ordering or grouping criterion.

4.1 Single Queries

The simplest query supported in the tool is a request for visualizing an entity in the dataset. The user requests such visualizations through a *show* intent, i.e. using the verb show or a synonym for expressing the command (e.g., display, draw etc.). Single queries may also include a request for ordering the values, both in the intent (*order* or synonyms) or as a subquery. A sample natural language query of a simple query is the "*Show me the ratings*" for displaying the rating of each workout in the dataset. The user can ask for ordering in two different ways: directly on the intent (e.g., *Order the athletes' ratings*) or through a subquery (e.g., *Show me the ratings in order*). In both cases, the tool will show the bar chart in Fig. 6, which is the default chart for numeric attributes. Categorical ones use the same chart, but they report the frequency distribution.

The visualization shows different interactive widgets that help the user in refining the initial query. On the top-left corner, the interface shows a button for changing the ordering (ascending or descending) and the orientation of the bar chart (vertical or horizontal). At the bottom, it shows a slider for filtering the bars according to the attribute value (in this case, the performance rating).

4.2 Grouping Queries

The second category of queries allows the user to aggregate the entries in the dataset according to the value of one attribute. Grouping queries work on categories, numerical and date attributes. In the latter case, the system requires the size of the discretization bucket, whose default are included in the annotation file.

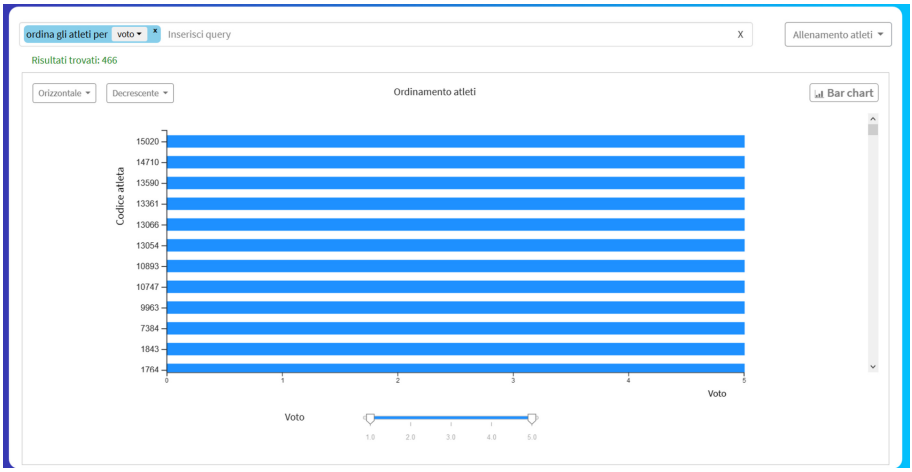

Fig. 6. A sample single query: ordering the athlete's ranking

There are two visualization templates for the grouping queries. The first one works for categorical or numerical attributes and allows the user to visualize the relative occurrence of a given category or the percentage of samples contained into a given bucket. A sample natural language query corresponding to this template is "*Group workouts by calories*". The *group* intent and its synonyms request the aggregation, while the name of the involved entity bounds the considered query to the first template since calories are a numeric attribute in our dataset. Figure 7 shows the resulting pie chart, using buckets of 300 calories.

The second template for the grouping queries exploits a date (or time) attribute for the aggregation. In this case, the resulting visualization shows the trend of a numerical attribute over time. The user may set the interval directly in the query, or the system may use the default in the annotation file. A sample natural language query for this template is "Show me the login trend by month" or "Group the login by month". In this case, the user specifies an aggregation or a generic intent, but we recognize the request relying on the time unit (a month in our example, but it may be a day, week, year etc.). Figure 8 shows the resulting visualization for the sample query. The system uses a line plot for better displaying the overall trend. Besides, below the graph, the system displays the fields for filtering the plot by date.

4.3 Correlation Queries

Another category of queries available in the system shows the correlation between two attributes. As indicated by the name, a correlation query shows a possible relation between the two attributes through a scatterplot. A sample natural language query for this template is "Show me the workout distribution on calories and duration". In this case, the user specifies either an ordering or a generic

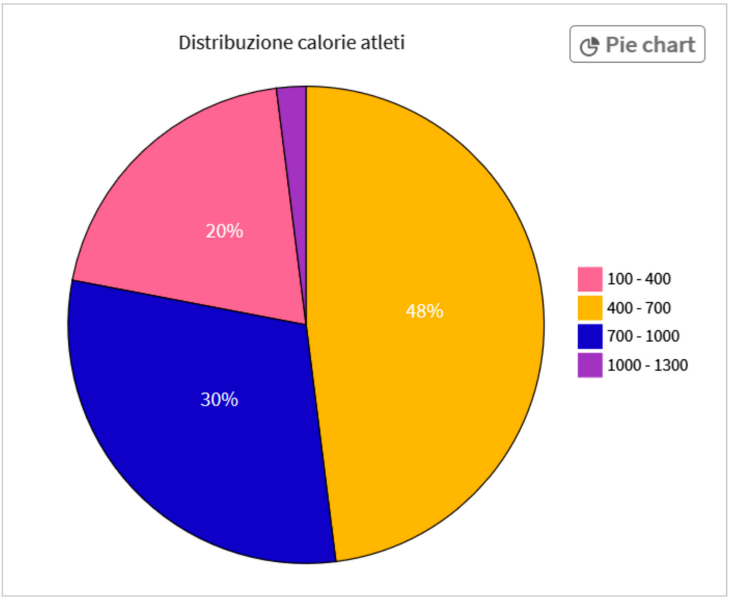

Fig. 7. Sample grouping query on a numeric attribute



Fig. 8. Sample grouping query showing a trend over time

intent, but we recognize the correlation request relying on the specification of two different attributes for ordering the entries. Figure 9 shows the resulting plot.

4.4 Composite Queries

The tool supports the composition of the different query templates we discussed in the previous sections, resulting in multiple visualizations over the same data

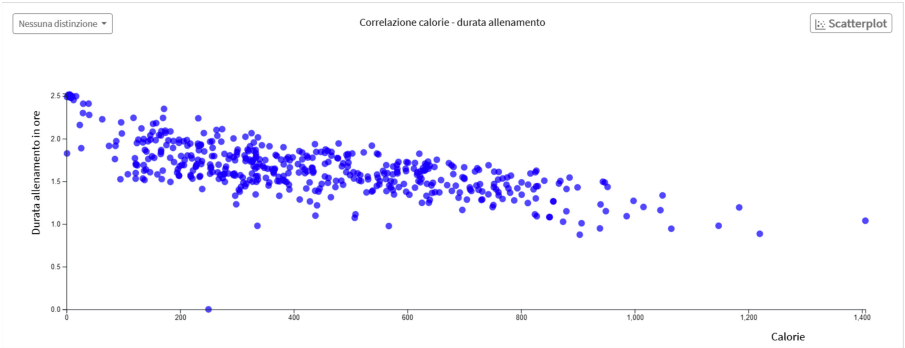

Fig. 9. Sample correlation query

attributes. In particular, the system supports the composition of up to three visualizations, requesting filtering, grouping and aggregation at the same time. A sample composite query is "Show me the duration of the workouts by athletes between 20 and 55 years old, grouped by calories and with their login trend by month". The resulting visualization is available in Fig. 10. It shows three different blocks: on the left, we have a column for the bar chart showing the workout duration and the line chart shows the login trend, while on the right we have the pie chart showing the grouping by calories. All the displayed entries satisfy the filtering condition on the athlete's age. The user can specify different composite queries combining only two intents (e.g., grouping and trend aggregation) or using different attributes.

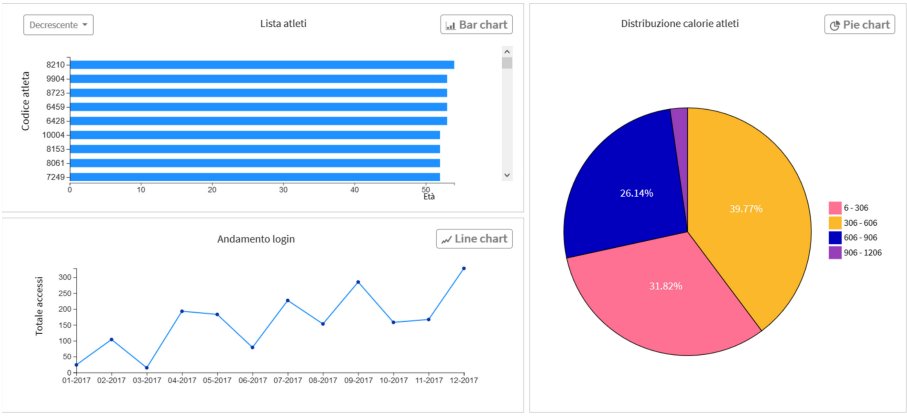

Fig. 10. Sample composite query

5 Evaluation

We evaluated the usability and the effectiveness of the proposed system through a user test, which included different data searching tasks on the athlete workout dataset. At the beginning of the session, we asked the participants to read a document explaining the test purposes, the organization of the dataset and an overall view of the requested tasks. After that, we asked each participant to fill out a demographic questionnaire and to complete the following five tasks through the application:

1. Sort the athlete workouts by ranking and filter out those lower than 2;
2. Inspect the relation between the calories spent and the duration of the workouts;
3. Find the best athletes and group them by calories spent;
4. Show the login trend in the last 12 months;
5. Filter the workouts of the athletes between 20 and 55 years old, group them by calories and analyse their login trend.

After each task, participants were requested to fill the Subjective Mental Effort Questionnaire (SMEQ) [12] for measuring the user's cognitive effort and the After-Scenario Questionnaire (ASQ) [5] for evaluating the ease of use and the supporting information. After completing the test, we requested the participants to fill out the System Usability Scale (SUS) [1] questionnaire for evaluating the overall usability, complemented by a set of questions on peculiar aspects of the application.

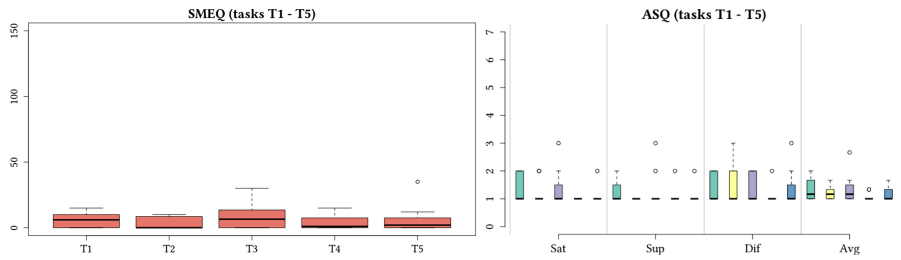

Fig. 11. User test results including the SMEQ [12] (left part) and ASQ [5] (right part) questionnaires.

Twelve people participated in the test. They had different education levels: 8 had a High School Degree, 3 a Bachelor, 3 a Master and one a PhD. They had a good experience with office applications ($\bar{x} = 5.5, s = 0.79$) in a 1 to 7 Likert scale) and they spend about 7 h ($\bar{x} = 7.2, s = 3,1$) per day working or entraining themselves using the web.

They were all able to complete all the tasks. The ANOVA analysis for repeated measures on the SMEQ data shows that we do not have a significantly

higher complexity ($p = .39$, $F(11, 44) = 1.05$, see Fig. 11 left part). Therefore, the cognitive effort requested to the users was comparable, but we noticed a slightly lower value for the tasks requesting to visualise only one chart (T1: $\bar{x} = 5.6$, $s = 5.3$, T2: $\bar{x} = 3.2$, $s = 4.6$) if compared against those requesting two (T3: $\bar{x} = 7.9$, $s = 9.3$) or three (T5: $\bar{x} = 5.9$, $s = 10.1$). All tasks require a cognitive effort below the *Not very hard to do* label in [12].

The ASQ questionnaire instead measures different perceived dimensions for each task: the satisfaction (*Sat*), the provided support (*Sup*), the difference between the time expected and spent (*Dif*), using a 1 to 7 Likert scale where 1 is positive and 7 is the negative end. All tasks received ratings between 1 and 2 and there was no significant difference among them (Fig. 11 right part).

The SUS post-test questionnaire [1] results show that the tool usability is very good ($\bar{x} = 93.8$, $s = 4.1$ in a 1–100 scale). In the post-test comments, the user highlighted some minor usability issues regarding the query suggestion feature when used through the keyboard. Overall, the users considered the approach very useful, especially for browsing online dataset when graphical elaboration is not available.

6 Conclusion and Future Work

In this paper, we discussed a simple solution for supporting natural language queries on a generic dataset. The approach exploits existing Artificial Intelligence libraries for natural language analysis and an annotation file for training query recognition. At runtime, a Python module executes the interpreted query while the visualization relies on a JavaScript library. We summarised the different types of queries supported in our system, describing how it extracts and presents the results. The user test results show that despite the simplicity of the approach, the usability results are very good, so we are positive that the integration of natural language queries will be more and more available in data exploration tools.

The proposed solution may be further developed in the future, adding a higher number of graphs. In particular, the tool is lacking support for geographical data (e.g., choropleth maps), which are both extremely useful and quite widespread on the web. In addition, we are working for going beyond the "one-shot" query interpretation model, where the user provides all his/her input at once and the system works on a single string. The next version of the tool will rely on a conversational interface, where the system will ask questions to the user for understanding which kind of visualization suits better the task, guiding him/her through the different option available. Considered the evolution of general-purpose conversational agents based on modern AI libraries, the technology is mature enough for studying its application in the information visualization field.

References

1. Brooke, J., et al.: SUS-a quick and dirty usability scale. Usability Eval. Ind. **189**(194), 4–7 (1996)
2. Cau, F.M., Mancosu, M.S., Mulas, F., Pilloni, P., Spano, L.D.: An intelligent interface for supporting coaches in providing running feedback. In: Proceedings of the 13th Biannual Conference of the Italian SIGCHI Chapter: Designing the Next Interaction, pp. 1–5 (2019)
3. Fu, S., Xiong, K., Ge, X., Tang, S., Chen, W., Wu, Y.: Quda: natural language queries for visual data analytics (2020)
4. Hoque, E., Setlur, V., Tory, M., Dykeman, I.: Applying pragmatics principles for interaction with visual analytics. IEEE Trans. Vis. Comput. Graph. **24**(1), 309–318 (2017)
5. Lewis, J.R.: Psychometric evaluation of an after-scenario questionnaire for computer usability studies: the ASQ. ACM Sigchi Bull. **23**(1), 78–81 (1991)
6. Popescu, A.M., Etzioni, O., Kautz, H.: Towards a theory of natural language interfaces to databases. In: Proceedings of the 8th International Conference on Intelligent User Interfaces, pp. 149–157 (2003)
7. Saktheeswaran, A., Srinivasan, A., Stasko, J.: Touch? speech? or touch and speech? investigating multimodal interaction for visual network exploration and analysis. IEEE Trans. Vis. Comput. Graph. **26**(6), 2168–2179 (2020). https://doi.org/10.1109/tvcg.2020.2970512, https://doi.org/10.1109/TVCG.2020.2970512
8. Setlur, V., Battersby, S.E., Tory, M., Gossweiler, R., Chang, A.X.: Eviza: a natural language interface for visual analysis. In: Proceedings of the 29th Annual Symposium on User Interface Software and Technology, pp. 365–377 (2016)
9. Setlur, V., Tory, M., Djalali, A.: Inferencing underspecified natural language utterances in visual analysis. In: Proceedings of CH 2019, Italy, pp. 40–51 (2019)
10. Srinivasan, A., Stasko, J.: Orko: facilitating multimodal interaction for visual exploration and analysis of networks. IEEE Trans. Vis. Comput. Graph. **24**(1), 511–521 (2018)
11. Yu, B., Silva, C.T.: Flowsense: a natural language interface for visual data exploration within a dataflow system. IEEE Trans. Vis. Comput. Graph. **26**(1), 1–11 (2020). https://doi.org/10.1109/tvcg.2019.2934668, https://doi.org/10.1109/TVCG.2019.2934668
12. Zijlstra, F.R.H., Van Doorn, L.: The construction of a scale to measure subjective effort. Delft, Netherlands, vol. 43, pp. 124–139 (1985)