Supporting Iterative Cohort Construction with Visual Temporal Queries

Josua Krause, Adam Perer, Harry Stavropoulos

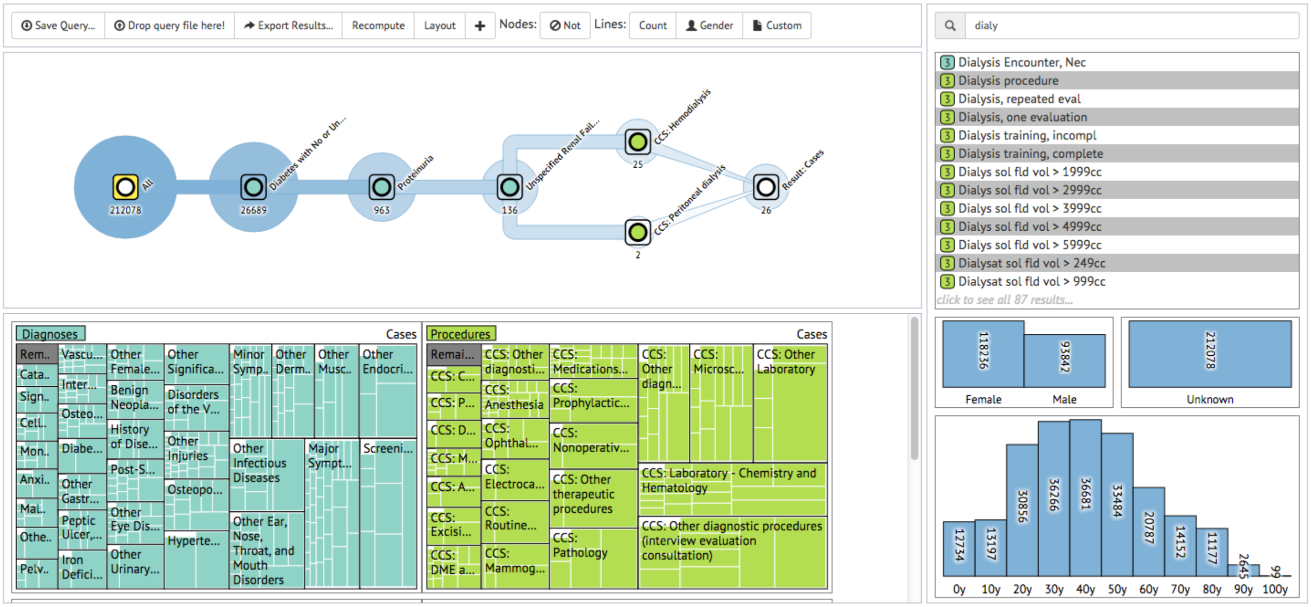


Fig. 1. As a part of a case study with a clinical researcher, *COQUITO* is utilized for medical cohort analysis. The researcher visually applies a series of temporal constraints to a patient population and generates a cohort to statistically analyze.

Abstract— Many researchers across diverse disciplines aim to analyze the behavior of cohorts whose behaviors are recorded in large event databases. However, extracting cohorts from databases is a difficult yet important step, often overlooked in many analytical solutions. This is especially true when researchers wish to restrict their cohorts to exhibit a particular temporal pattern of interest. In order to fill this gap, we designed *COQUITO*, a visual interface that assists users defining cohorts with temporal constraints. *COQUITO* was designed to be comprehensible to domain experts with no preknowledge of database queries and also to encourage exploration. We then demonstrate the utility of *COQUITO* via two case studies, involving medical and social media researchers.

Index Terms— Visual temporal queries, cohort definition, electronic medical records, information visualization.

1 Introduction

Researchers in many disciplines, such as medicine, social science, and business analytics, seek to understand the effects of various factors on cohorts, a population or group of individuals with common features. For instance, in medicine, researchers may want to understand if a cohort is at risk of developing a disease or health outcome, or social media researchers may wish to understand if a cohort will adopt a new social technology. In the era of Big Data, where electronic medical records and social data logs are commonplace, the opportunity to do retrospective cohort studies has never been greater. However, extracting cohorts with temporal constraints from databases is an important yet difficult step, often overlooked in many analytical solutions.

While there are many tools to assist researchers in filtering populations by attributes or facets (e.g. i2b2[1] and BTRIS[2] in medicine), many cohort studies require their subjects to exhibit a temporal pattern to qualify. Unfortunately, these traditional query tools often lack support for temporal queries.

Suppose medical researchers wish to understand the treatment effectiveness for patients suffering from a disease. They may need to define a cohort as a set of patients with a disease diagnosis A, followed by a medication prescription B and a performed procedure C happening within 3 days apart from each other, and within 7 days after A. Alternatively, the cohort should also contain other patients with a diagnosis A followed by a treatment D within 5 days. A visual representation of this query is illustrated in Fig. 2. Expressing such a temporal query is extremely complex with standard query languages, such as SQL. Recently, there have been several research projects designed to define temporal constraints on populations [13,29], but these approaches typically rely on filling out a set of forms, and do not give feedback about the results until all constraints have been defined.

- *Josua Krause is with NYU. E-mail: josua.krause@nyu.edu*
- *Adam Perer is with IBM T.J. Watson Research Center, E-mail: adam.perer@us.ibm.com*
- *Harry Stavropoulos is with IBM T.J. Watson Research Center, E-mail: hstavrop@us.ibm.com*

Manuscript received 31 Mar. 2015; accepted 1 Aug. 2015; date of publication xx Aug. 2015; date of current version 25 Oct. 2015. For information on obtaining reprints of this article, please send e-mail to: tvcg@computer.org.

[1] `http://www.i2b2.org`
[2] `http://www.btris.nih.gov`

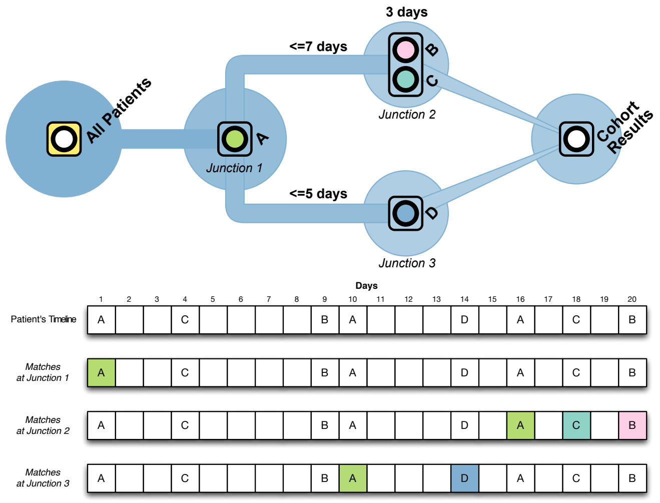


Fig. 2. An illustrative query that demonstates how temporal constraints work, beginning with the full population (the leftmost source junction) and the resulting cohort at the rightmost node. To demonstrate how partial results work, an example timeline of a single patient is presented. Junction 1 contains only patients with an event A in their time line. Junction 2 contains those that also have an event B and C at most 3 days apart from each other and within 7 days after the initial event A. Junction 3 contains patients with an event D within 5 days after the initial event A. The result junction contains all patients that satisfied either junction 2 or 3.

In order to fill this gap, we designed *COQUITO*, a visual interface that assists users in building COhort QUeries with an ITerative Overview for specifying temporal constraints on databases. *COQUITO* was designed to be accessible to domain experts with no pre-knowledge of database queries. But most importantly, it was designed to encourage exploration, where all types of data can be used in the query, intermediate results are shown so users can iteratively adapt where a query yields too few results or too many results, and users receive hints on what to include in a query using visual summaries of the databases contents.

Concretely, our contributions are:

a. A novel visual query UI that allow users to specify temporal constraints in a user-centric way. Interactions are done by drag-and-dropping patterns of interest, rather than relying on forms, modal dialog boxes, or an SQL-like language. Users can build queries using a hierarchical event dictionary, to easily specify event types at the level-of-detail relevant to them. Users receive continual, real-time feedback about the result set as new constraints are added and results are encoded within the visual query.

b. A novel Temporal Query Server that is optimized to support complex temporal queries on large databases with interactive feedback for real-time exploration.

c. Case studies, involving medicine and social media, that illustrate that *COQUITO* is an effective tool for domain experts defining temporal-based cohorts on real-world datasets.

Our paper begins with a description of the motivation for our tool, which is rooted in a medical informatics scenario. We then describe related work, following by a description of our system. After that we describe case studies of domain experts utilizing *COQUITO* in their research. Finally, we conclude with a discussion and our plans for future work.

2 Motivation: Medical Informatics and Predictive Modeling

The desire to create temporally-constrained cohorts is common to many domains. However, the design of *COQUITO* was motivated by our experience working with medical researchers using retrospective electronic medical records for predictive modeling so they can understand the risk of disease onset for patients. Medical institutions are increasingly collecting electronic medicals records (EMR) systems on their patients that combine information about diagnoses, procedures, medications, laboratory test results, and more. EMRs can be leveraged in predictive modeling, a common and important methodology used in healthcare research to personalize treatment guidelines [17] or detect diseases earlier before they progress [2].

Cohort construction, also known as cohort selection, is the first step of the predictive modeling pipeline [22, 30]. Without properly defined cohorts, the quality of the models and predictions will be impacted. Often, predictive models require the definition of multiple cohorts. For example, if researchers were interested in modeling onset of a disease, researchers will need to define a cohort with the disease (the cases) and a matched cohort that does not have the disease (controls). Then, these cohorts will be mined for predictive signals that distinguish the two cohorts.

However, defining these cohorts is often problematic. When talking to our medical colleagues, they described the painful scenario of how they recently defined a cohort. It involved medical doctors pontificating how certain diseases may be represented in EMR diagnosis codes, which are abstract as they are typically used for billing and not for recording diseases. After the doctors' suggestions, technologists would then design SQL queries to model the patients as described, and prepare reports for the medical doctors to review. Often, the resulting patient set would be too small or too large to do meaningful analysis, or the patients would have properties unforeseen and not valid for the cohort study they had in mind. In a recent project, the colleague described this process took several months and over a dozen iterations, which significantly delayed the ability to begin building predictive models. In the end, the cohort was defined using a sequence of one of 30 possible and different diagnosis codes, clinical encounters of 4 different types, complex temporal constraints determining when the patient truly had onset of a disease (e.g. if two diagnoses appear in the first year, and the third occurs within 18 months, use the date of the first diagnoses – otherwise use the second), age criteria constraints at time of diagnosis, and temporal constraints ensuring there is enough data surrounding these diagnoses to be useful for predictive modeling. In fact, the details for cohort definition ended up requiring a lengthy document to describe its complexity, and a series of Python and SQL scripts to extract the cohort from the database.

Only after this cohort was defined could the researchers begin focusing on what their goal was all along: building predictive models to predict the onset of a disease.

COQUITO was designed to alleviate this process of cohort construction, where the domain experts could be empowered to do query generation and exploration. Furthermore, the queries should be fast enough that it can be used interactively, particularly in exploratory research group discussions where cohort definitions are brainstormed.

3 Related Work

There are two main ways of supporting users to define a set of records, or a cohort, in temporal data: a *pattern recognition* approach where users begin with an overview of the data and then filter towards their desired result set, or a *pattern specification* approach where users specify a query. We review related work among these two approaches, as well examples of recent visual-based cohort exploration systems for medical records.

3.1 Pattern Recognition Approaches

The pattern recognition approach allows users to start with a view of records, and using these records, determine an interesting pattern, and expand the result set to find more records that fit the pattern. Overview-based visualizations also fall into this category, as users may not have any expectations to properties of the data, but upon finding interesting visual patterns, choose to filter on demand.

A visual way of finding repeating patterns in single records of continuous time data can be done by using a spiral view of the data as proposed by Weber *et al.* [38] and Carlis and Konstan [5]. Also for continuous time data, *TimeSearcher* (Hochheiser and Shneiderman [15]) introduced time boxes that allow for brushing multiple time spans and value ranges. With *TimeSearcher 2*, Buono *et al.* [4] amends this by introducing a similarity search feature. Pattern searching in the ISS solar panel motor data is done by Haigh *et al.* [14]. Users can select time windows that are being used for the pattern search and also can join multiple windows temporally or logically to make the search more specific. The joins are specified using a visual interface. Holz and Feiner [16] follow a semi-automatic approach called spatially relaxed selection in which a user sketches a pattern and the system finds similar sections in the data. *TimeBench* [33] provides a software library with data structures and algorithms for building pattern recognition capabilities for visual analytics tools.

Lin *et al.* [24] with *VizTree* uses a suffix / subsequence tree showing the frequency of patterns as edge weight to search discretized continuous time data. This enables the discovery of both common and unusual patterns. Vrotsou *et al.* [37] expands on this idea in *ActiviTree* by dynamically expanding the tree via user interaction while using the created path as query. *ActiviTree* is used on daily activity of persons data which consists of point event time lines. Demographics (i.e. gender) of the resulting population are shown.

In *Similan 2* the user can query EMRs using a record as reference (Wongsuphasawat *et al.* [41]) to find similar patients. Aligning, ranking, and filtering records eases this task. In *LifeFlow* (Wongsuphasawat *et al.* [40]) users get an aggregated view on EMRs from intensive care unit logs. This way common patterns about patient treatment can be detected. *EventFlow* (Monroe *et al.* [27]) is an extension of this which can handle even larger and noisier data sets. This is achieved by simplifying records and allowing for alignments on arbitrary points in time.

Finally, there are also numerous automated data mining techniques to detect patterns in single or multiple records or features (e.g. Keogh and Smith [20], Wu and Chen [42]). A recent approach, *Frequence* (Perer and Wang [32]), integrates an automatic frequent sequence mining approach with an interactive visualization.

3.2 Pattern Specification Approaches

An alternative approach for creating cohorts of records is to start by defining a specification and using matching elements as result set. This approach is often rooted in a prior assumption about the data, a hypothesis, or expert knowledge. In order to tailor the resulting record set to desired needs, the specification often needs to be iterated and altered slightly until a satisfying result is reached.

There are numerous extensions to standard query languages, like SQL, that ease the task of querying temporal data (e.g. T-SQL [26], Sequence Subset Operators by Dunn *et al.* [9], Tquel by Snodgrass [36]). However, formulating queries in those languages is still too complex for many domain experts.

Visual query languages ease this task by giving an easy to interpret visual representation of the current query. Chittaro and Combi [7] use a paint strip metaphor to show interval relations and query a patient history database. Different event types occupy their own row in the visual representation. However, this leads to harder to read queries since you have to find the row of a given type first. Also, logical operations cannot be represented in the query and need to be encoded separately. This is done by Combi and Oliboni [8] who add a separate logic composition area to apply logical operations to the queries. PAT-Expert [21] also supports a graphical representation of boolean query operators, but partial results are not shown in line with the view.

PatternFinder (Fails *et al.* [11]) lets users formulate queries on patient event histories with connected boxes. Each box represents an event constraint that is customizable by widgets in the box. The same applies for specifying the sequential constraints of the events. Results are shown in a different visualization showing the individual matches of the query per patient. A similar approach is *Query Marvel* (Jin and Szekely [18, 19]) which replaces the boxes with a comic strip metaphor and introduces negation, conjunction, and disjunction of events. As with our system their representation of queries focuses more on the relative order of events which makes it easier to formulate and read queries than with the paint strip metaphors mentioned above. However, both *PatternFinder* and *Query Marvel* allow only for a single sequence of events (i.e. no branching like in our tool) and neither show intermediate results nor provide hints on what constraint to add to the query.

Monroe *et al.* [28, 29] also introduced a visual query language for their point and interval data in *EventFlow*. The search queries are specified by creating event sequences using the visual encoding of its predecessor *LifeLines 2*. Queries can contain constraints including point events and intervals specifying presences and absences which also can overlap. Forms are used to specify constraints and results are shown in the aggregate view of *EventFlow*. Intermediate results are implicitly shown in the result view. However, since only patients matching the query are shown there is no way to detect when in the sequence of events patients got removed from the result. Therefore it is difficult to find out which part of the formulated query is responsible for e.g. the removal of the most patients. Similarly, DecisionFlow [13] supports the analysis of high-dimensional temporal event sequence data with interactive visualizations, but relies on non-iterative, form-based queries.

For non-temporal query systems, Elmqvist *et al.*'s [10] DataMeadow provides a graph based query system where connections between nodes represent the *flow* of the data. Nodes represent filters or set operations which represent the current state of the data, with only a limited amount of attributes visible per node. DataMeadow lacks an explicit way of suggesting what to do next and cannot express temporal relations, like *COQUITO*.

Another system, DataPlay by Abouzied *et al.* [1], lets users formulate nested queries that are represented in a tree structure. Users create queries by typing commands into an interface, aided by suggestions from both nodes in the tree and the dynamically generated result set. DataPlay does not allow for intermediate query results or time sequence queries.

Zhao *et al.* [44] shows results of faceted search results in a Venn diagram like way by positioning all points of the dataset. However, aside from duplicating points, there is no way to show intermediate results. This system also does not support large numbers of rows or temporal data.

Our work, *COQUITO*, provides a visual way for pattern specification. Additionally, it provides complementary visualizations to provide hints so users can also recognize interesting patterns to include in their queries. After an exhaustive literature review, *COQUITO* appears to be the first pattern specification system for temporal event sequences to show results in-line with the query, show intermediate results for subqueries, and provide hints for new query constraints.

3.3 Cohort Exploration in Electronic Medical Records

In addition to several of the above systems that were designed for medical records, there are other recent tools that allow users to visualize patient cohorts. A recent survey provides a thorough analysis of works on exploration and visualization through 2013 (Rind *et al.* [34]). More recent work includes *Outflow* (Wongsuphasawat and Gotz [39]) and its successors *CareFlow* (Perer and Gotz [31]) and *DecisionFlow* (Gotz and Stavropoulos [13]) which allow users to view the temporal trends of similarity-based cohorts. *CAVA* (Zhang *et al.* [43]) uses multiple visualization techniques, including bar charts, treemaps, and Outflow visualizations, to explore and analyze patient cohorts, integrated with analytics.

A different approach is utilized by Malik *et al.* [25] with the tool *CoCo*. *CoCo* compares patient cohorts of ICU data and identifies differentiating metrics and event sequences.

4 SYSTEM

In this section, the user interface of *COQUITO* is described in detail, starting with the query view, how to interpret the visual representation, and what interactions are possible. Next, other visual elements of the user interface, including the event search panel, the demographics view, and the event treemaps are described. Then, we explain how the interface can be integrated with external cohort analytics using web-based APIs (e.g. a predictive modeling pipeline). Last, the temporal query server is described to showcase how queries are computed efficiently. *COQUITO*'s visual UI is implemented using D3 [3]. The Temporal Query Server is implemented in Java and Apache Tomcat to efficiently query a DB2 database server.

4.1 User Interface

The main part of the user interface of *COQUITO* is the query view, which occupies the center of the UI. A query, as seen in Fig. 2, is divided into *junctions* which are connected with *routes*. The left-most junction, the source, represents the entire population. The right-most junction, the results, represent the members that satisfy all user-defined constraints. Users can add new junctions on the route between the source and the results, which filters the population by putting constraints on their timelines. The radius of a junction, as well as its color saturation, are log-scale proportional to the total count of members that satisfy the query up until the point. A numerical label, describing the actual count, is also placed below each junction for precise feedback to users.

There are numerous types of temporal constraints that users can impose on their cohorts. The most common constraint type are event existence constraints which require an event to appear in a member's timeline. Event types are categorized by their inherent hierarchies which allow users to specify a desired granularity. Each hierarchy receives a categorical color to represent it in each of the views. Event existence constraints can also be negated by applying a "NOT" operation from above the query view on to an existing constraint. Negated nodes are shown with a red outline. Furthermore, age constraints can also be invoked to restrict members to a certain age range when a given event occurs. In contrast to the previous constraints there are some features of members that are time independent, like gender or ethnicity. This enables users to filter members to contain, for example, only women.

Overall, the combination of junctions and constraints allow users to express all common set operations (such as "AND", "OR", and "NOT"). Query references (Section 4.1.4) further allow set operations on end results of queries.

To demonstrate how the visual query builder works, consider a scenario where a medical researcher wishes to query for a diagnosis of "Glaucoma" followed by both "Visual Field Examination" and "Eye Exam & Treatment" procedure events in any order. Initially, she will start with an empty query which returns the entire population (Fig. 3a). Then, she adds a "Glaucoma" constraint node onto the source junction (Fig. 3b). After that, she drags a "Visual Field Examination" constraint to follow the "Glaucoma" node. Constraint nodes can be created by dragging visual elements from the other views, which are described later. As users drag new constraints, the creation of a new junction is hinted in the UI as with a subtle gray junction appearing under the cursor (Fig. 3c). Then, after dropping the constraint, a new junction containing the event existence constraint is created. As the query is still running, the constraint node is colored with a stripe pattern to indicate it is still loading (Fig. 3d). After the query finishes a few moments later, the junction is sized and colored according to its new member population (Fig. 3e).

However, in the scenario described above, the researcher wishes for both "Visual Field Examination" and "Eye Exam & Treatment" procedures to take place. Therefore, she drags "Eye Exam & Treatment" into the same junction to make it a conjunctive constraint (Fig. 3f). As new constraints are dragged into existing junctions, users receive visual feedback of an animated expanding junction as they drag constraints onto the top of it.

Upon seeing the results and not being satisfied, she changes her mind and wants a cohort that either has a "Visual Field Examination" or an "Eye Exam & Treatment". She drags the "Eye Exam & Treatment" constraint from its current junction, and drags it to after the "Glaucoma" junction. This creates two routes to the results, which yields a larger result population (Fig. 3g), as members of all disjunct paths in the query are summed in this rightmost junction.

However, the researcher wishes to make sure the members in her cohort actually had the medical procedures shortly after their "Glaucoma" diagnoses. Therefore, she wishes to specify a time window in which events need to appear. She does this by clicking on the route between the junctions which opens an input box for specifying a time window of less than or equal to 5 days (Fig. 3h). Time windows on routes define when the first event of next junction need to happen, and this can be a closed or open interval. Time windows can also be specified for junctions themselves when they contain multiple constraints and users wish to have them all happen within a given time.

The researcher now wishes to compare how much member overlap there is between the "Visual Field Examination" junction and the "Eye Exam & Treatment" junction. To do this, she drags "Visual Field Examination" junction on top of the other junction, and she receives visual feedback in form of a yellow circle indicating how many members are in the intersection between both junctions (Fig. 3i). This interaction can be used, for example, to get information about overlaps between separate branches of a query.

For further analysis the researcher wants to find patients distinct from the current result but with a similar history. To do this, she creates a new query and drags the node of the result junction onto the newly created input junction. After that she performs a "NOT" operation on this reference node (more details in Section 4.1.4) and adds a junction containing a "Glaucoma" and one containing a "Visual Field Examination" constraint. Now her second query contains all patients that are not in the result of the first query but still had a "Glaucoma" diagnosis before a "Visual Field Examination" (Fig. 3j).

Some tasks may require balanced cohort sizes, however, there are currently different sizes of these result sets. In order to match patients from the query returning fewer results with patients from the larger query the user drags the node from the result junction to the junction with more results. This samples the result of the larger query to match the result of the smaller query (Fig. 3k).

Different queries use different base colors to be identifiable in the other views. For instance, if she wants to understand the difference between the cohorts, she can click on junctions which in turn become yellow to indicate selection. The demographics panel will populate with summary information, such as the gender, ethnicity, and age distribution of the junction. This information is shown for each query distinguished by their base color (Fig. 3k).

It is also possible to encode various attributes inline with the query. For instance, in Fig. 4, the gender attribute was applied to the routes. In this mode, the height of the blue routes represents the proportion of male members that satisfy each constraint, and the red routes match the proportion of females. It is also possible to visualize other domain-relevant attributes in this mode, such as health outcome (e.g. the proportion of patients that remained healthy or were hospitalized). However, this feature likely does not scale beyond attribues with a small number of values, such as binary attributes like gender or health outcome.

4.1.1 Event Search

COQUITO features a search functionality on the right side of the user interface, so that users can locate specific events of interest to add as constraints to the system (see Fig. 4). As users type the event name, the results are automatically filtered to show the matching events with each keystroke. Each event name is augmented with an icon to help users distinguish what type of event it is (e.g. medication versus diagnosis). Color indicates event type and corresponds to the same colors used in the query view and treemaps. The numeric label inside the icon refers to the level of the event within its hierarchy (i.e. a 1 refers to a top level, e.g. "Medication", whereas a 4 refers to a finer level, e.g. "Aspirin"). If users wish to browse different hierarchical levels of the event types shown, they can click the icon to see all ancestor elements, so users can quickly select less specific levels of interest. If a user wishes to use this event as a constraint in a query, they can drag the icon into the query view.

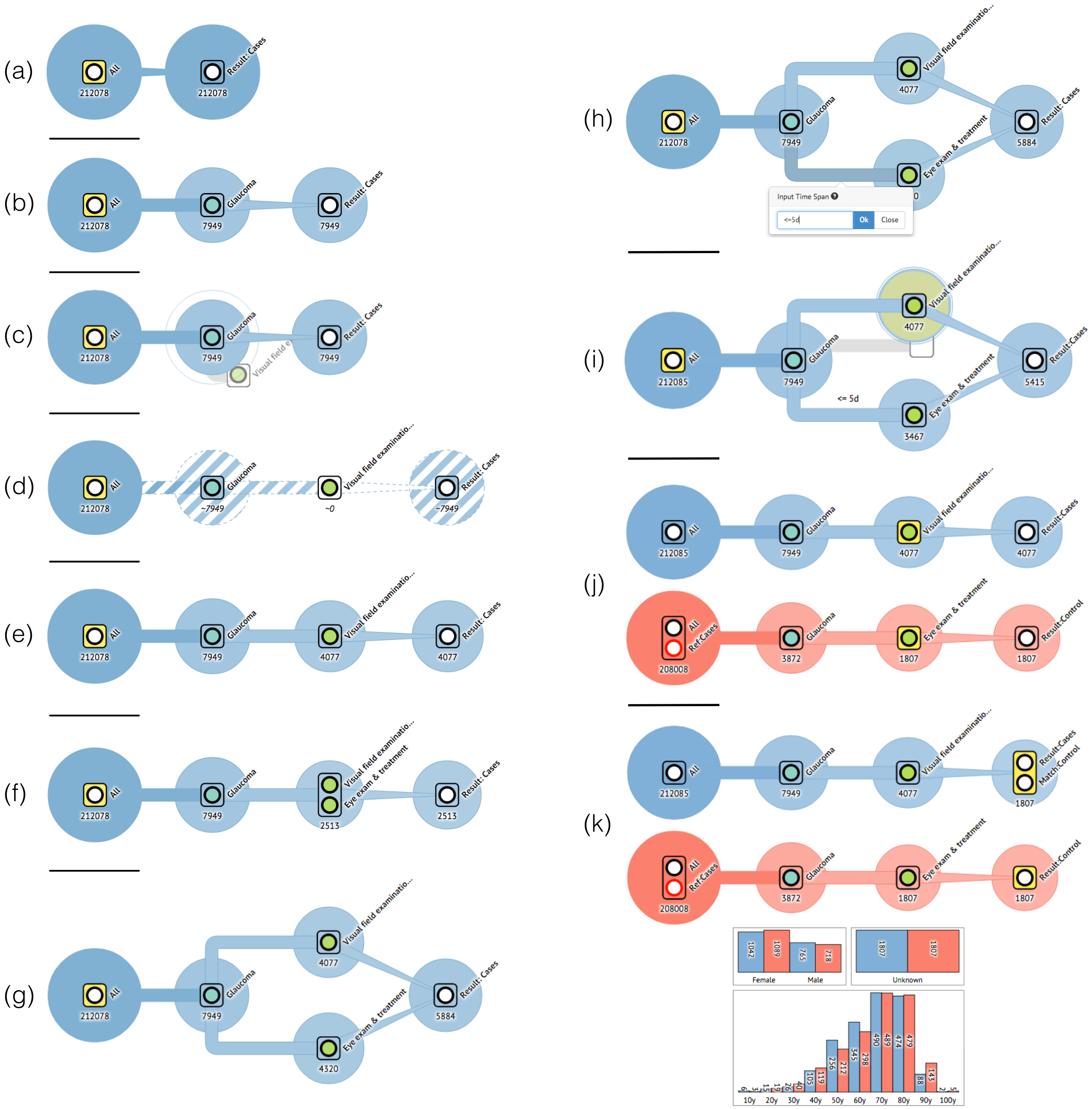


Fig. 3. This Fig. demonstrates how a query is built using *COQUITO*. Refer to the *User Interface* section (Section 4.1) for a description of each step.

4.1.2 Demographics

The bottom right of the UI provides a view of the general demographics of the cohorts, as shown in Fig. 5. Currently, *COQUITO* shows bar charts for gender, ethnicity, and age of the population in this view, by default, but can be expanded to show other variables. If users create multiple queries, each demographic chart has multiple colored bars for each of the corresponding queries (e.g. Fig. 3k). Initially, the bar charts will show demographics of the entire population, but when users click on a junction, the bar charts will update to show the demographics for members that match the junction's constraints. The demographics are also interactive, so users can drag values from the demographics view into the queries themselves. For instance, if users want to limit a query to females, they can drag the female bar chart into a junction of interest.

The age distribution bar chart is a special case, as the age of people may change over the progression of a query. If a user drags an age constraint to a junction, the junction will require a person to be within the specified age range to satisfy the constraint. Users can also later edit the age constraint values by clicking on it, and modifying the time constraint.

4.1.3 Treemap

In order to get an overview of which events are common within the population, treemaps are shown at the bottom part of the user interface. Each top level event type, like diagnoses or medications, has

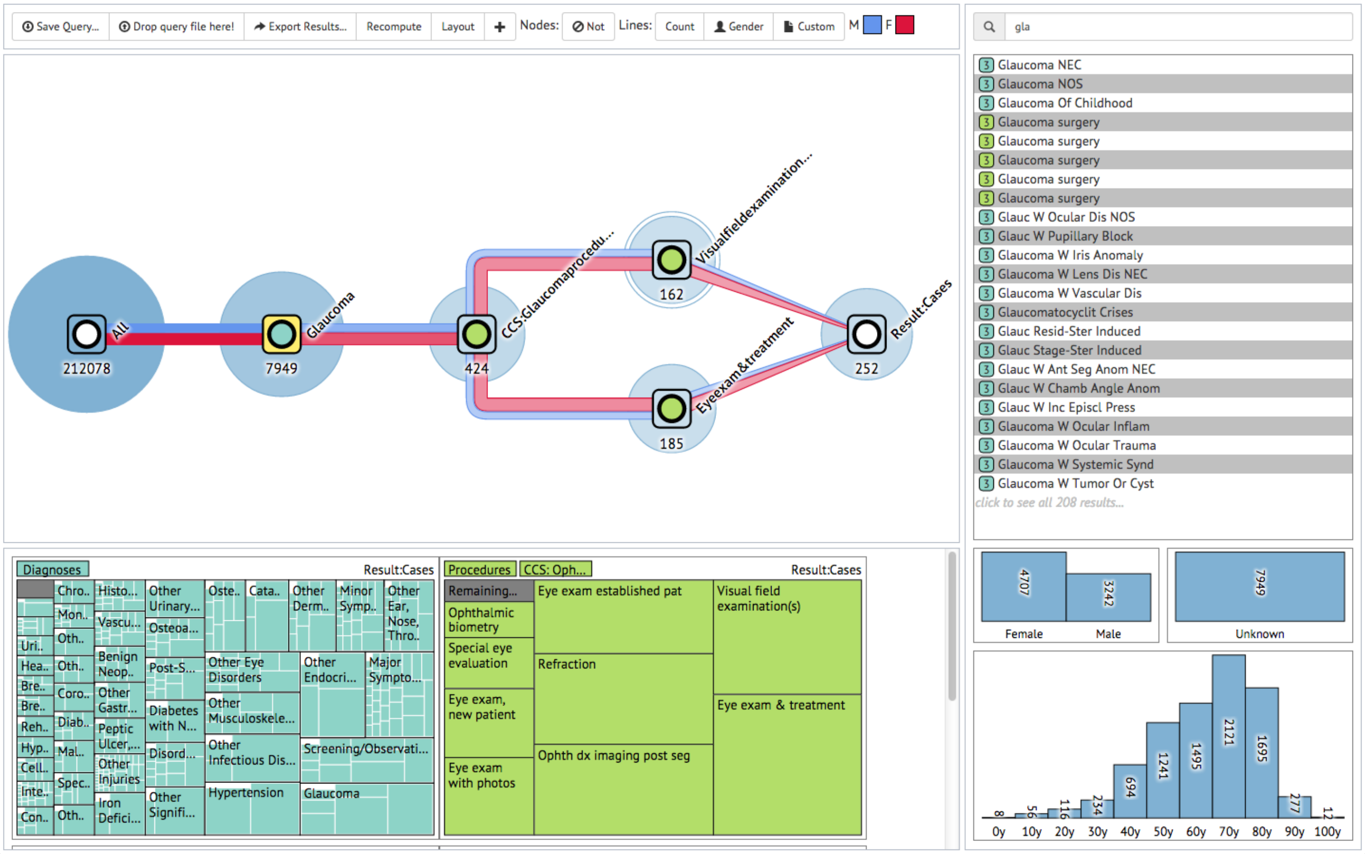


Fig. 4. An overview of the *COQUITO* user interface. The center features the visual temporal query to define a cohort. On the right, the search box is being used to locate additional event constraints. Below the query, treemaps are shown to show the event distribution after the selected junction ("Glaucoma" in yellow). On the bottom right, demographics of the selected junction are also shown, including gender, ethnicity, and age distributions.

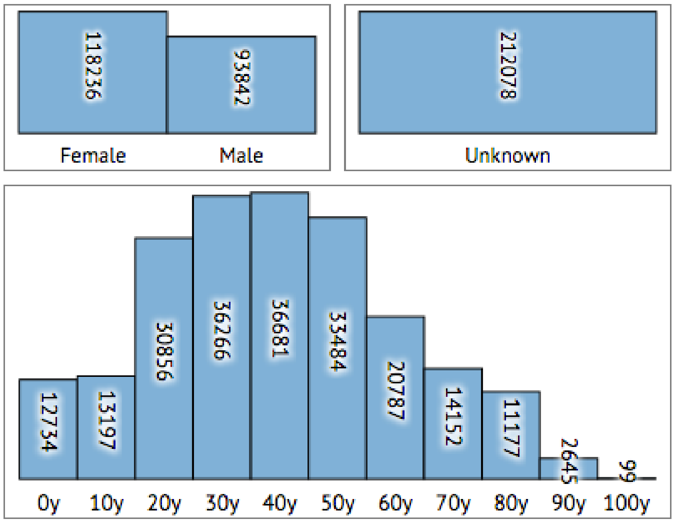
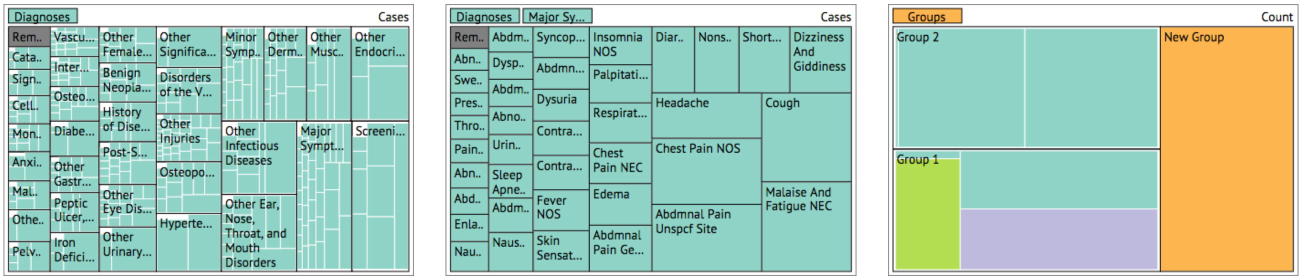


Fig. 5. Demographics of the whole population of a medical dataset. The top left bar chart shows the gender distribution of the complete data set and the top right shows the ethnicity distribution, which is unspecified in this dataset. The bottom bar chart shows the age distribution of the patients through the whole data set. The actual values are shown vertically within the bars as well as tool tip when hovering with the mouse.

its own treemap (Fig. 6a). Within each treemap, the events are organized hierarchically (such as grouping medications by their higher level classes). By default, and when the source junction is selected, the size of each node in the treemap reflects the number of members that have such an event. When users select a junction, the treemap is sized by the members that have such an event occur after the clicked junction. In this scenario, the treemaps act as a hint as to how new event constraints will effect the cohort results. To avoid making the treemap overwhelming for users, all events that are less than 10% as frequent as the most common event are merged into one gray node labeled as "Remaining" whose size reflects the sum of all merged nodes. Users can drill-down on a treemap by clicking on a node, which expands the subtree to the full size of the treemap (i.e. Fig. 6b shows a drill-down into the "Major Symptoms" subtree). Breadcrumbs at the top of the treemap show which part of the event type hierarchy is being visualized. The breadcrumbs can be clicked to navigate upwards in the hierarchy. All treemap nodes and breadcrumbs can also be dragged onto the query view to add new constraints.

Often, users may wish to group together multiple types of events, so that users can refer to them as a single meta-event, or group, in a query. A naïve approach to achieve this would be to force users to build parallel paths over and over again, which is time consuming. To overcome this limitation, *COQUITO* provides a way to define arbitrary disjunctive groups as a new group event type. A special, initially empty, additional treemap serves as target for group events (Fig. 6c). Dragging a node onto this treemap adds the given event type to a group or creates a new group of event types if it was dragged on a special "New Group" node in the treemap. The size of each group node is proportional to the number of members who satisfy the group's events. Users can then drag groups from this treemap onto a query to add a group constraint.



(a) (b) (c)

Fig. 6. Treemaps show the hierarchies of event types. The highest level of the “Diagnoses” hierarchy can be seen in (a) while (b) shows the next level for the subtype “Major Symptoms, Abnormalities”. The size of the elements reflect the total number of occurrences of a given type in the whole data set. (c) shows two user-created meta-event groups.

4.1.4 Referencing Query Results

For certain cohort construction tasks, users may need to be able to reference results of a query inside a different query. Many cohort analysis tasks require cohorts that have no overlap, such as when users need to define a case cohort (a population with a condition) and a control cohort (a population without that condition) to compare the differences. For instance, after defining an initial cohort, users may wish to create a new query that only examines data that is not in the results of this first cohort, but with additional constraints. *COQUITO* supports this by dragging the node of a query’s result junction onto an input junction of another query. The created reference node can also be negated with the “NOT” operation (indicated by a red outline of the node) from above the query view resulting in the complement of the referenced query as input for the given query (e.g. see Fig. 3j). Furthermore, by combining multiple reference nodes in the same junction, users can intersect multiple result queries or subtract patient sets (via negated reference nodes).

4.1.5 Matching

Cohort matching, or balancing, is an optional process of making the case and control cohorts comparable with respect to extraneous factors. For certain types of cohort studies, the cases and controls should be as similar as possible with regard to potentially important confounding factors. A variety of approaches exist to do this this, including techniques proposed by Rose and van der Laan [35] and Last [23]. *COQUITO* currently supports a naïve matching algorithm using random sampling to balance the size of the cohorts. However, the system is extensible to support more sophisticated techniques, such as logistic regression, by connecting to API services that do the matching calculation. Matching is performed by dragging a result node of a query on top of the result junction of the query that should match with the dragged query (see Fig. 3k).

4.1.6 Integration with external Cohort Analytics and Predictive Modeling

Users can also export result sets of the queries into file formats such as *CSV*, so that the cohorts are available to other applications, or for further analysis. In addition, there is functionality to support direct integration with existing APIs, e.g. sending the patient cohort to a cohort analysis back-end and displaying the results in the user interface.

As an example, *COQUITO* has been integrated with the PARAMO predictive modeling pipeline [30]. Users can use the “Deliver to Pipeline” button to send the cohorts defined visually by “Case” and “Control” queries to a pre-configured predictive model. After the model is created and evaluated, *COQUITO* displays the overall evaluation score (the AUC (Area Under Curve) score, a common measure of the predictive quality of models [12]). If users wish to inspect the model more closely, clicking on the AUC value will launch a separate window visualizing the model features using the INFUSE system [22] (see Fig. 7).

4.1.7 Save & Load Functionality

We encourage exploration and mitigate the impact of making mistakes while building a query by providing a history and save & load functionality in the user interface. Each change that is made to the query is added as point in the browser history to enable the use of the navigation buttons provided by the browser. In addition, queries can be saved as files and be loaded again at a later time.

4.2 Temporal Query Server

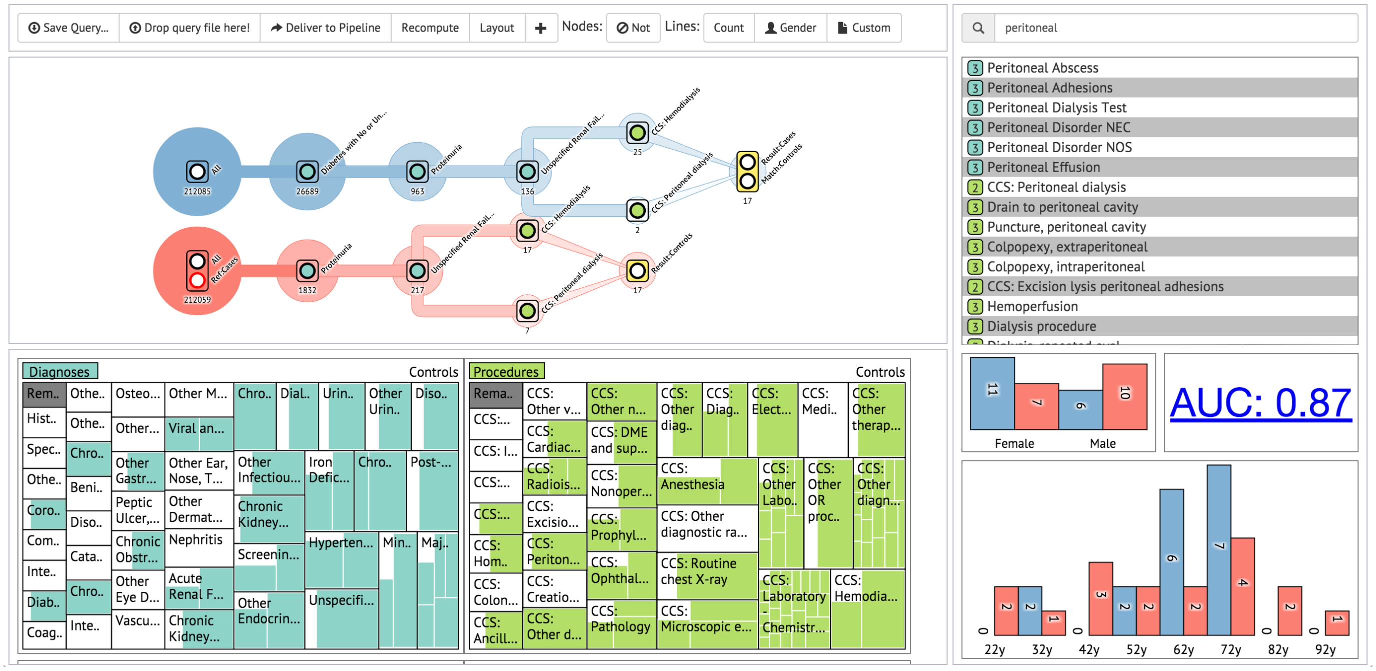
Users may wish to extract cohorts from databases with millions of patients, each with thousands of unique events, which are among tens of thousands of distinct event types. Querying those databases might take a very long time, considering users not only want the end result of the query but also each member (and their corresponding timestamps of the constraint event) within each junction in the query. The same holds true for the computation of the event hierarchies for the treemaps, which takes a set of members and a time stamp as input. While a query is in progress, the affected junctions and links in the query view are shown with stripe patterns as seen in Fig. 3d. Despite these demands, *COQUITO* employs various techniques to ensure the server’s response is interactive.

4.2.1 Incremental Changes

Often, during exploration, users will modify a query with incremental changes that affect only one path of the query, so there is rarely the need for the server to compute the full query. When a junction is modified, only the results for the junction itself and its descendants are recomputed. Furthermore, users are not forced to wait for a query to finish before they continue interacting with the UI, so they can continue to add or modify existing constraints. If other parts of the query are changed before a previous request returns, the new set of junctions to recompute is joined with the previous set. Responses from the server are sorted by the time of the request so that no result of older requests can overwrite more recent results. Since the server often does not compute the full query, but instead returns the population for every junction, the computation of the overall result of the query is done on the client side by creating the set union of the end points of all query paths.

4.2.2 Caching

An additional way of speeding up server responses is achieved by caching the most recent and most common queries. A LRU (Least Recently Used) cache with the ability of perpetually caching special requests is used for query, treemap, and demographics requests. However, matching queries is not necessarily trivial. A hash value is computed for every query tree in order to quickly reject unequal query trees. If the hash is the same, a tree matching algorithm that respects arbitrary ordered children and constraint nodes is used to verify that both trees are the same and the cached response can be used.



3 Peritoneal Abscess
3 Peritoneal Adhesions
3 Peritoneal Dialysis Test
3 Peritoneal Disorder NEC
3 Peritoneal Disorder NOS
3 Peritoneal Effusion
2 CCS: Peritoneal dialysis
3 Drain to peritoneal cavity
3 Puncture, peritoneal cavity
3 Colpopexy, extraperitoneal
3 Colpopexy, intraperitoneal
2 CCS: Excision lysis peritoneal adhesions
3 Hemoperfusion
3 Dialysis procedure

Fig. 7. After generating a cohort of patients with Diabetes in Fig. 1, the medical researcher creates a corresponding cohort of control patients with Diabetes by using *COQUITO*'s referencing and matching features.

4.2.3 Query Execution

A naïve translation of a query built with *COQUITO* to SQL would require nesting queries, which is quite complicated for query builders. For example, the query would first filter for all events after the first constraint, then on this sub-table filter for all events after the second constraint, and so on. Even this complexity does not take into account branching and specifying further temporal restrictions that *COQUITO* supports. As the sequence of events in time show up as multiple rows in the database, it is not possible to formulate simple queries without nesting and examining sub-tables. Formulating such a query requires an analyst to think of requirements from a database point of view and not from a point of view that expresses the natural way of brainstorming temporal queries. We believe this limits analysts to quickly formulate hypotheses or explore data set freely.

In order to avoid some of this complexity, our Temporal Query Server actually computes queries in two stages. First, the row based representation of point events in the database is converted into actual timelines while simultaneously filtering only for event types relevant to the query. Then, the server recursively searches each timeline for matches of the users' specification.

This process is illustrated in Fig. 2, where the server attempts to see if the patient matches the cohort query. The patient has an A event, so Junction 1 is satisfied. However, it needs to find a B and C event within 7 days in order to satisfy Junction 2. The A event found in Junction 1 is not sufficient, so it moves forward to the other A events to test. It finally succeeds when it finds the A event on Day 16. A similar process is used to see if the patient satisfies Junction 3's constraint. This example also showcases that the same person can appear in multiple nodes and in different branches of the tree, as shown by the patient who matches both Junction 2 and Junction 3.

In order to build the treemaps, event hierarchies need to be queried. If the user selects a junction the treemap reflects the event hierarchy only for events that happen after the events matched by the junction. Those events can happen at different times for each member. Therefore, an on-demand table is created filtering events that happen after the timestamp corresponding to each member. This table is then aggregated on each level of the event hierarchy. Events occurring multiple times in the timeline of a member are only counted once.

5 Case Studies

In order to show the utility of *COQUITO*, we describe two case studies of how cohorts were constructed from real-world datasets using our tool. As the nature of cohort construction is exploratory in nature, real user tasks do not map well to tasks measuring time or errors in a traditional controlled study. Instead, we showcase *COQUITO* on two datasets with real users to demonstrate it works in multiple domains on big, real-world datasets.

5.1 Medical Cohort Analysis

Our first case study involves a clinical researcher interested in extracting case and control cohorts from a longitudinal database of electronic medical records for use in predictive modeling. The researcher is an MD with a background in internal medicine. His database features over 200,000 patients from a major healthcare provider in the United States. The database contains over 5,000 unique medications, 11,000 unique procedures, and 14,000 unique diagnoses, and the researcher is interested in building cohorts based on these medical features. His larger goal is to better understand if the onset of Diabetes, a chronic diseases with high blood sugar levels that often lead to serious health complications, can be predicted from this type of data.

Attribute-based filtering is not sufficient for this use case, as he needs to build cohorts where the temporal order of treatments, procedures and diagnoses is important. However, these queries are notoriously difficult to write, so he typically requires a team of technologists and database experts to write the SQL to define his cohorts based on his clinical expectations. However, since clinical hypotheses do not always map well to EMR features, the results are often too restrictive or too general, so he often has to do several iterations of queries, which when relying on others, can drag on for weeks or even months. The clinical researcher recalled a recent project where the cohort query definition took a few months time and about a dozen iterations.

As the researcher only recently received access to the healthcare provider's database, he is unfamiliar with the contents and if there would be enough patients to test his hypotheses. His first investigation was set to involve patients with Diabetes and related conditions. After searching for and dragging the appropriate Diabetes diagnosis code to the query view, he realized that about 10% of the patient population had such a diagnosis, and he was off to a good start. He was particularly interested in Diabetes patients that have also been diagnosed with Proteinuria, a sign of renal (kidney) damage, common to Diabetes patients. After dragging this diagnosis to the query, there were 963 patients left, which he considered a reasonable cohort size to do his analysis. However, only some of these would go on to have renal failure (136). Of those, some would receive Hemodialysis (an ambulatory treatment), and others would receive Peritoneal dialysis (a treatment that can be done at home). He wanted to consider patients with one or both types of treaments, so he used a branch junction to get his final result set of 26 patients – and used this as the query to define his Case cohort. This query is shown in Fig. 1.

Curious about how predictive the onset of diabetes is in this complex patients, he created a new Control cohort query by referencing and negating the results of this original case query, so this new cohort would not have any of the same patients. Since he wanted to compare the Case patients to other patients that also had renal failure, he dragged the Proteinuria node to this query too, followed by a Renal Failure node. Similarly, he followed this with a branch of Peritoneal and Hemodialysis treatments, which resulted in 17 control patients. Since this number was smaller than the cases defined above, he dragged the Control results node to the Cases results node, shown in Fig. 7. Now that these cohorts were defined, he delivered them to his predictive modeling pipeline in PARAMO. Later, after the pipeline executed, the researcher was able to get the results of the prediction inside *COQUITO* (an AUC score of 0.87). He was quickly able to determine that there were indeed predictive features embedded in the EMRs of these diabetic patients.

The clinical researcher was excited by the speed of being able to determine the efficacy of the database, as well as also being in control of the cohort construction. He explored a variety of other diabetic sub-cohorts, built a variety of useful predictive models, and plans to continue using *COQUITO* as a part of his regular workflow.

5.2 Social Media Cohort Analysis

Our second case study involves a social media researcher interested in defining cohorts of people who exhibit interesting temporal patterns of visiting locations. Such data is available from Location Sharing Services, where users choose to broadcast the various places they visit. Marketing researchers are interested in better understanding if data from Location Sharing Services can give businesses and consumers improved experiences, by better understanding customers' habits and desires.

One such service is Foursquare, which allows its users to check-in at different venues to save and share their locations to friends. According to Foursquare[3], there are over 55 million users and over 6 billion check-ins with Foursquare, as of March 2015. A corpus of a subset of Foursquare data was made publicly available [6], which contains over 200,000 users and over 22 million check-ins. This corpus was further filtered to users in New York City (NYC) in [32], where the database consists of 17,739 users with 419,023 check-ins in 17,182 unique venues over 11 months. In this NYC-centric database, each unique venue was augmented with Foursquare's hierarchy of 9 top-level categories and 289 sub-level categories to create three hierarchical levels-of-data on which to query cohorts.

For her analysis, the researcher desired to understand the differences in demographics and social media popularity of two different populations, so she used *COQUITO* to create two parallel queries, shown in 8. For example, the researcher was interested in the trajectory of different Foursquare users who begin their day at coffee shops. Some, as shown in the blue query, move on to farmer's markets, and then cocktail bars. Others may go to fast food restaurants, followed by dive bars. The social media researcher can easily compare how many people fall into both queries by dragging the red query's results junction over the blue query's result junction – and seeing the overlap. After determining the cohorts were disparate populations, she exported the member sets generated by *COQUITO*, as CSVs, to her usual analytical tools and computed statistics about these populations.

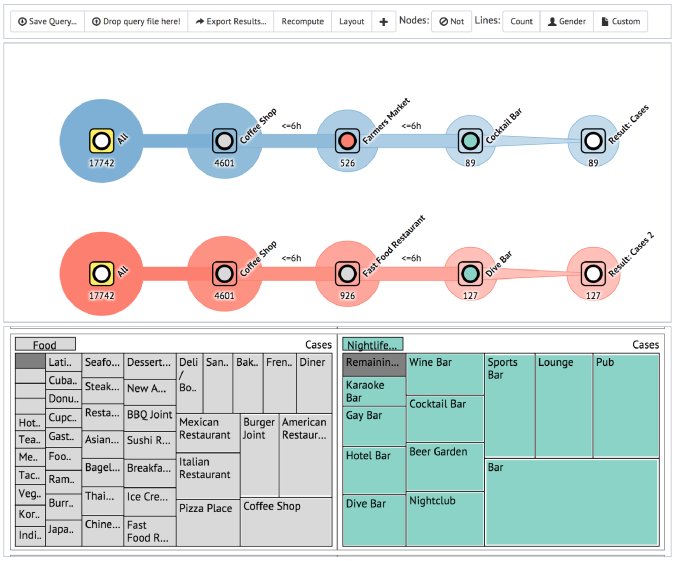


Fig. 8. As a part of a case study with a social media researcher, *COQUITO* is utilized for analysis of Foursquare users' "check-ins". In this example, the researcher is able to create two parallel queries to generate two complementary cohorts for analysis based on their temporal patterns.

An example of the resulting insights from this analysis included that the cohort that ended up at cocktail bars instead of dive bars had 36% more Twitter followers, suggesting this might be a more valuable population to market to, if the businesses goal was to drive more social media traffic. This is despite the fact that this cohort actually tweeted about 10% less than the dive bar cohort. *COQUITO* allowed the researcher to segment the population in more meaningful ways than her traditional analysis tools supported, and this is just one example of the types of insights that this iterative exploration afforded.

6 Conclusion and Future Work

We presented *COQUITO*, a visual query tool to interactively create cohort populations with temporal constraints. By providing iterative feedback during query creation, the system encourages exploration of the underlying data-set with complementary visualizations of bar charts showing demographic properties and treemaps showing the distribution of events. Visible elements from each of these views can be used to expand a query, even when previous queries are in-progress. In order to support rapid, interactive feedback, query execution is optimized by *COQUITO*'s temporal query server. Furthermore, the usefulness of the tool was demonstrated with case studies of domain experts using the tool on real-world datasets.

However, there remains future work to understand the effects of allowing users to create visual temporal queries for cohorts. We plan additional usability studies to ensure the visual query language and interactions are comprehensible for a diverse set of domain experts. Although our case studies provide anecdotal evidence that users are able to master *COQUITO*'s visual interface with less than 15 minutes of training, formal usability studies should also be conducted in order to quantify the effectiveness of the design of all advanced features. Furthermore, a request by many of the domain experts that have used the tool is to include standard statistical tests within *COQUITO*. This would allow them to make advanced decisions about their cohorts without having to first export the cohorts to their traditional tools. Finally, currently the tool only supports queries relating point events. An interesting challenge would be to expand the visual query language to also support interval events.

[3] http://www.foursquare.com/about

References

[1] A. Abouzied, J. Hellerstein, and A. Silberschatz. Dataplay: Interactive tweaking and example-driven correction of graphical database queries. In *ACM User Interface Software and Technology*, pages 207–218, 2012.

[2] R. Bellazzi and B. Zupan. Predictive Data Mining in Clinical Medicine: Current Issues and Guidelines. *International Journal of Medical Informatics*, 77(2):81–97, 2008.

[3] M. Bostock, V. Ogievetsky, and J. Heer. D3 data-driven documents. *IEEE Transactions on Visualization and Computer Graphics*, 17(12):2301–2309, 2011.

[4] P. Buono, A. Aris, C. Plaisant, A. Khella, and B. Shneiderman. Interactive pattern search in time series. *SPIE Visualization and Data Analysis*, 2005.

[5] J. V. Carlis and J. A. Konstan. Interactive Visualization of Serial Periodic Data. In *ACM User Interface Software and Technology*, pages 29–38, 1998.

[6] Z. Cheng, J. Caverlee, K. Lee, and D. Z. Sui. Exploring Millions of Footprints in Location Sharing Services. *AAAI Weblogs and Social Media*, 2011.

[7] L. Chittaro and C. Combi. Visualizing Queries on Databases of Temporal Histories: New Metaphors and Their Evaluation. *Data and Knowledge Engineering*, 44(2):239–264, 2003.

[8] C. Combi and B. Oliboni. Visually Defining and Querying Consistent Multi-granular Clinical Temporal Abstractions. *Artificial Intelligence in Medicine*, 54(2):75–101, 2012.

[9] J. Dunn, S. Davey, A. Descour, and R. T. Snodgrass. Sequenced Subset Operators: Definition and Implementation. In *Data Engineering*, pages 81–92, 2002.

[10] N. Elmqvist, J. Stasko, and P. Tsigas. Datameadow: A visual canvas for analysis of large-scale multivariate data. *Information Visualization*, 7(1):18–33, 2008.

[11] J. A. Fails, A. Karlson, L. Shahamat, and B. Shneiderman. A Visual Interface for Multivariate Temporal Data: Finding Patterns of Events across Multiple Histories. In *IEEE Visual Analytics Science And Technology*, pages 167–174, Oct. 2006.

[12] T. Fawcett. An Introduction to ROC Analysis. *Pattern Recognition Letters*, 27(8):861–874, 2006.

[13] D. Gotz and H. Stavropoulos. DecisionFlow: Visual Analytics for High-Dimensional Temporal Event Sequence Data. *IEEE Transactions on Visualization and Computer Graphics*, 2626, 2014.

[14] K. Z. Haigh, W. Foslien, and V. Guralnik. Visual Query Language : Finding Patterns in and Relationships among Time Series Data. *Workshop on Mining Scientific and Engineering Datasets*, 2004.

[15] H. Hochheiser and B. Shneiderman. Dynamic Query Tools for Time Series Data Sets: Timebox Widgets for Interactive Exploration. *Information Visualization*, 3(1):1–18, 2004.

[16] C. Holz and S. Feiner. Relaxed selection techniques for querying time-series graphs. In *ACM User Interface Software and Technology*, pages 213–222, 2009.

[17] P. B. Jensen, L. J. Jensen, and S. Brunak. Mining Electronic Health Records: Towards Better Research Applications and Clinical Care. *Nature Reviews Genetics*, 13(6):395–405, 2012.

[18] J. Jin and P. Szekely. QueryMarvel: A Visual Query Language for Temporal Patterns using Comic Strips. In *IEEE Visual Languages and Human-Centric-Computing*, pages 207–214, 2009.

[19] J. Jin and P. Szekely. Interactive Querying of Temporal Data Using a Comic Strip Metaphor. In *IEEE Visual Analytics Science and Technology (VAST)*, pages 163–170, Oct. 2010.

[20] E. Keogh and P. Smyth. A Probabilistic Approach to Fast Pattern Matching in Time Series Databases. In *ACM Knowledge Discovery and Data Mining*, pages 20–24, 1997.

[21] S. Koch, H. Bosch, M. Giereth, and T. Ertl. Iterative integration of visual insights during scalable patent search and analysis. *IEEE Transactions on Visualization and Computer Graphics*, 17(5):557–569, May 2011.

[22] J. Krause, A. Perer, and E. Bertini. Infuse: Interactive feature selection for predictive modeling of high dimensional data. *IEEE Transactions on Visualization and Computer Graphics*, 20(12):1614–1623, Dec 2014.

[23] J. M. Last. *A Dictionary of Epidemiology*. Oxford University Press, 4th edition, 2001.

[24] J. Lin, E. Keogh, S. Lonardi, J. P. Lankford, and D. M. Nystrom. Visually Mining and Monitoring Massive Time Series. In *ACM Knowledge Discovery and Data Mining*, pages 460–469, 2004.

[25] S. Malik, F. Du, M. Monroe, E. Onukwugha, C. Plaisant, and B. Shneiderman. Cohort comparison of event sequences with balanced integration of visual analytics and statistics. In *ACM Intelligent User Interfaces*, pages 38–49, 2015.

[26] Microsoft and Sybase. T-SQL. `http://www.tsql.info/`.

[27] M. Monroe, R. Lan, H. Lee, C. Plaisant, and B. Shneiderman. Temporal Event Sequence Simplification. *IEEE Transactions on Visualization and Computer Graphics*, 19, 2013.

[28] M. Monroe, R. Lan, J. Morales del Olmo, B. Shneiderman, C. Plaisant, and J. Millstein. The Challenges of Specifying Intervals and Absences in Temporal Queries: A Graphical Language Approach. In *ACM Human Factors in Computing Systems*, pages 2349–2358, 2013.

[29] M. Monroe, K. Wongsuphasawat, C. Plaisant, B. Shneiderman, J. Millstein, and S. Gold. Exploring Point and Interval Event Patterns : Display Methods and Interactive Visual Query. In *HCIL Technical Report*, pages 1–10, 2012.

[30] K. Ng, A. Ghoting, S. R. Steinhubl, W. F. Stewart, B. Malin, and J. Sun. PARAMO: A PARAllel predictive MOdeling platform for healthcare analytic research using electronic health records. *Journal of Biomedical Informatics*, 2013.

[31] A. Perer and D. Gotz. Data-driven Exploration of Care Plans for Patients. In *ACM Human Factors in Computing Systems*, pages 439–444, 2013.

[32] A. Perer and F. Wang. Frequence: Interactive Mining and Visualization of Temporal Frequent Event Sequences. In *ACM Intelligent User Interfaces*, pages 153–162, 2014.

[33] A. Rind, T. Lammarsch, W. Aigner, B. Alsallakh, and S. Miksch. Timebench: A data model and software library for visual analytics of time-oriented data. *IEEE Transactions on Visualization and Computer Graphics*, 19(12):2247–2256, 2013.

[34] A. Rind, T. D. Wang, W. Aigner, S. Miksch, K. Wongsuphasawat, C. Plaisant, and B. Shneiderman. Interactive Information Visualization to Explore and Query Electronic Health Records. *Foundations and Trends in Human-Computer Interaction*, 3(3):207–298, 2013.

[35] S. Rose and M. J. van der Laan. Why match? investigating matched case-control study designs with causal effect estimation. *Journal of Biostatistics*, 2009.

[36] R. Snodgrass. The Temporal Query Language TQuel. *ACM Transactions on Database Systems*, 12(2):247–298, 1987.

[37] K. Vrotsou, J. Johansson, and M. Cooper. ActiviTree: Interactive Visual Exploration of Sequences in Event-Based Data Using Graph Similarity. *IEEE Transactions on Visualization and Computer Graphics*, 15(6):945–952, 2009.

[38] M. Weber, M. Alexa, and W. Müller. Visualizing Time-Series on Spirals. In *IEEE Information Visualization*, 2001.

[39] K. Wongsuphasawat and D. Gotz. Exploring Flow , Factors , and Outcomes of Temporal Event Sequences with the Outflow Visualization. *IEEE Transactions on Visualization and Computer Graphics*, 18(12):2659–2668, Dec. 2012.

[40] K. Wongsuphasawat, J. A. Guerra Gómez, C. Plaisant, T. D. Wang, M. Taieb-Maimon, and B. Shneiderman. LifeFlow: Visualizing an Overview of Event Sequences. In *ACM Human Factors in Computing Systems*, pages 1747–1756, 2011.

[41] K. Wongsuphasawat, C. Plaisant, M. Taieb-Maimon, and B. Shneiderman. Querying Event Sequences by Exact Match or Similarity Search: Design and Empirical Evaluation. *Interacting with Computers*, 24(2):55–68, 2012.

[42] S.-y. Wu and Y.-l. Chen. Mining Nonambiguous Temporal Patterns for Interval-Based Events. *IEEE Transactions on Knowledge and Data Engineering*, 19(6):742–758, 2007.

[43] Z. Zhang, D. Gotz, and A. Perer. Iterative cohort analysis and exploration. *Information Visualization*, 2014.

[44] J. Zhao, C. Collins, F. Chevalier, and R. Balakrishnan. Interactive exploration of implicit and explicit relations in faceted datasets. *IEEE Transactions on Visualization and Computer Graphics*, 19(12):2080–2089, 2013.