Releviz: Visualizing similar search results in two dimensions

David Eitan Poll
University of California, Berkeley
depoll@berkeley.edu

Jeff Bowman
University of California, Berkeley
jeffbowman@berkeley.edu

Abstract

We present Releviz, a system that provides a two-dimensional interface for selecting and displaying multivariate data records. Releviz describes a two-component system: A standardized constraint language for expressing search parameters specific to a particular data domain, and for evaluating those search parameters; and a two-dimensional display field of the search results, in linear or radial orientation, in which relevance maps to one of the major coordinate axes and a grouping function maps to the other. We provide two implemented grouping functions, weighted average, and a constraint-based clustering algorithm. We document and provide an internal evaluation of this system, including its sample implementation.

Keywords: Search, filtering, relevance, constraint

CCS: H.5.2: User Interface

1 Introduction

Modern user-centric applications, especially those on the internet, allow users to sift through massive amounts of data to find information useful to them. From search engines (Google) to real estate listings (Zillow), data filtering and searching are extremely common tasks for information workers of all sorts.

When using search engines with highly domain-specific data, such as Zillow's "Home Postings" search engine (See Figure 1) or PCWorld.com's laptop review browser (See Figure 2), users are subjected to a common pattern: given the large set of results, a user must filter down the results by adding constraints to their query. The user then is subjected to an iterative process of more tightly constraining a query until either their desired results are located or they have over-constrained the data, leaving him with too few results, and forcing him to relax his constraints in order to find results that closely match his criteria.

This iterative filtering process that users must perform when searching large data sets is time-consuming and impractical for users. Much of this stems from the fact that users rarely think in terms of hard constraints and filters, but rather in terms of "desires." As a result, the constraints they set are often more representative of their "ideal" results rather than their strict requirements. In fact, especially with data domains where perfect matches are unlikely, users are more often interested in "near-misses" to their queries. These "near-misses" are results that

Figure 1: Zillow.com's "Home Postings" search engine. Note the filters on the left that allow users to constrain their queries based on parameters such as price, number of bedrooms, and square footage in search of a home that meets their needs.

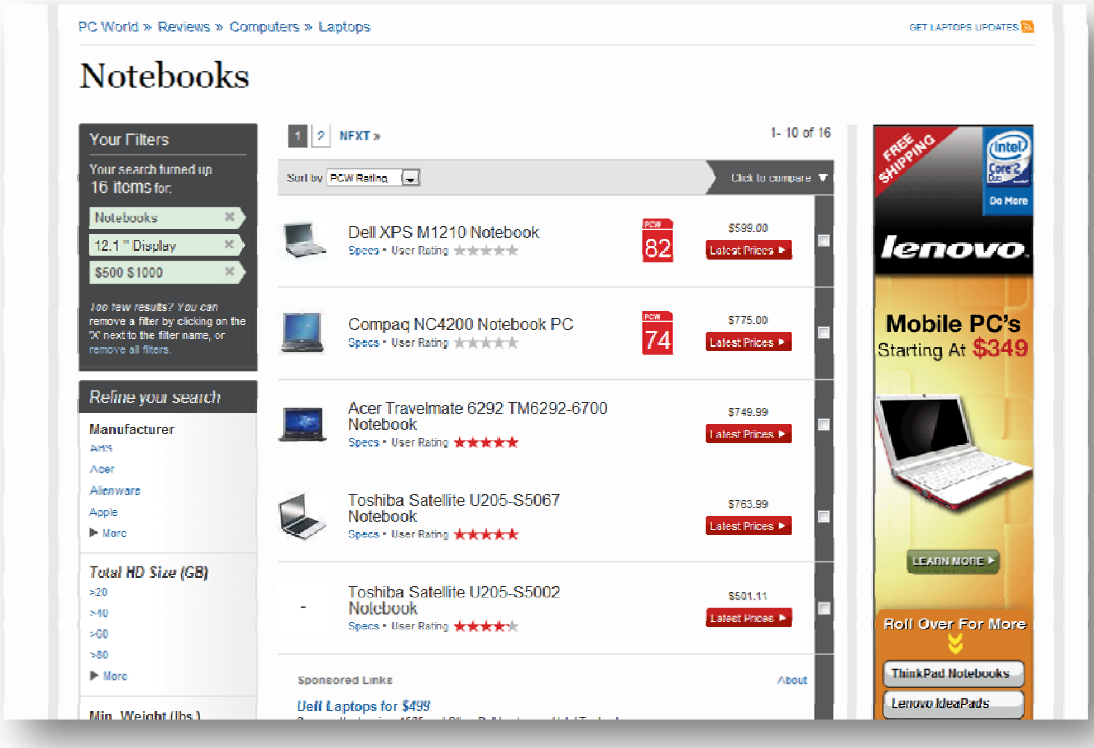

Figure 2: PCWorld.com's notebook review browser. Note that users can "refine their searches" based on a set of criteria. Selecting a criterion filters out laptops that do not meet the criteria. In this case, the user has filtered down to laptops between $500 and $1,000, but any laptop that costs $1,001 will be excluded from this list.

either meet most of their constraints or those that nearly meet a particular constraint. For example, a prospective home-buyer might constrain a search for home to a maximum price of $100,000, but if a home showed up on the market that otherwise met their needs and was listed for $101,000, the home-buyer might still be interested in that home.

In these types of situations, users are far less interested in strictly constraining a query to filter down results, but are instead interested in retrieving the most relevant results based upon their queries. In this paper, we propose Releviz, a pair of visualizations that helps users discover the "near-misses" based upon relevance for their queries, and that can be applied across varied data domains with multiple query constraints. First, we will explore works related to this problem. Next, we will explore the approaches we have taken in developing these visualizations. We will then present Releviz and discuss its merits and weaknesses. Finally, we will suggest future work within the scope of Releviz.

2 Related work

We build existing approaches, combining the results of constraint-based mapping, dynamic queries with direct manipulation, and 2D graphical representation to search results. The original interface to dynamic query, the Dynamic Homefinder [Williamson and Shneiderman 1992], provides a prototypical search function by which constraints are controlled and instant updates appear on a geographic map display. Another early example of constraint and relevance search is the Dynamic FilmFinder [Ahlberg and Shneiderman 1994], that seeks to generalize results display in a 2D grid. We work extensively from this principle, and from the principles of direct manipulation and incremental refinement that the system embodies.

In addition, the Attribute Explorer [Smith 2001] introduces a multiple-histogram and data-brushing approach to the problem of selection across multiple constraints, and the example produced with the Attribute Explorer also lists perfect matches along with records that fail one or more constraints; however, the output of this constraint matching interfaces is merely an ordered list, and does not allow for graphical output and relevance comparison.

Finally, Mann and Reiterer [1999] offer a system for visualizing text search results through a scatter plot with relevance as one of the core axes; however, this system provides for text search results instead of constraint satisfaction, and the scatterplot provided has no accounting for clustering algorithms of any form, instead suggesting individual keywords on each axis.

3 Methods

We approached the problem of visualizing near-misses to a query from two angles we considered to be important for users trying to browse filtered data. First, we explored constraint satisfaction in order to assign a relevance to each of the data points in the given data set. Next, we considered how near-misses might most appropriately be grouped so that users can see why data points are failing to meet their constraints.

Our methods assume that we have been given a "domain data set," which describes both the data points in the domain and the set of constraints that can be placed upon that data. With these two sets of information (constraints and data points), we can derive a general means for visualizing near-misses for queries over the data.

3.1 Constraint satisfaction

We define a constraint as an arbitrary weighted fitness function over a data point. As such, constraints consist of two main components:

1. Weight – an arbitrary real-number value that will be used in conjunction with other constraints.
2. Fitness function – an arbitrary function that returns a value between 0 and 1 indicating how well the constraint has been met by the data point the function takes as a parameter., where 1 indicates that the data point meets the constraint.

We have also identified two main types of constraints:

1. Discrete – constraints that have either been met or violated by a particular data point. Discrete constraints are characterized by piecewise fitness functions, usually only returning a 0 or 1. Examples of discrete constraints for searches for a laptop might include "has a webcam" or "has a solid-state disk."
2. Progressive – constraints that may be partially met by "close" values. Progressive constraints give us more information about how close a data point is to meeting the constraint. Often, progressive constraints have fitness functions that return 1 for all perfect matches, then decay in value as the data points drift further away from the criteria of the constraint. Examples of progressive constraints for home postings searches might include "distance from a place" or "price range."

Traditionally, constraints would be used to strictly filter the data set, where only data points that meet all constraints (with a fitness of 1) are included in the result set. For these types of queries, weight has no bearing.

We attempt to use these constraints to derive a scalar "relevance" value for each data point. We define relevance to be a weighted average of the fitness of the data point over all constraints:

$$rel(data) = \frac{\sum_{c \in constraints} fitness(c, data) \times weight(c))}{\sum_{c \in constraints} weight(c)}$$

This relevance calculation gives us a scalar between 0 and 1 indicating the overall fitness of a data point to a query. By mapping this value to a visual variable, we can indicate how well a particular data point matches the query.

3.2 Near-miss grouping

In addition to visualizing the relevance of a data point to a user's query, we attempt to visualize the reasons for which a particular data point received a less-than-perfect relevance score. We derive this grouping from the constraints that a particular data point failed to completely satisfy. To completely map these failures, we would require an arbitrary number of dimensions, since each constraint provides another dimension of data as to why the data point lost relevance. It is always a struggle to try to present something in two dimensions that would more naturally be plotted in n-dimensional space, so we made a few attempts to approximate the effect in a linear fashion. We have used two different methods in our visualizations to solve this:

3.2.1 Averaging

Our "averaging" approach attempts to linearize the constraint violations by using a weighted average of the constraints that were violated. We begin by distributing the constraints evenly along a number line. If only one constraint has been violated by a particular data point, the "grouping value" is simply assigned to the point on the number line that corresponds with the broken constraint.

With two constraints violated, we start at the point on the number line corresponding to the constraint for which the fitness was lowest, then shift the point toward the next violated constraint based upon how badly it was violated by the data point. Thus, if two constraints were equally violated by a data point, the grouping value would be exactly half-way between them.

We continue this process for three, four, up to n different constraints, resulting in an "average" (loosely defined) of the constraints that contributed to a decrease in relevance for a data point.

For some visualizations (in our case, the "radial" visualization), it is more useful to treat the number line as a circle, where the "halfway point" always lies on the minor arc between the constraints being averaged.

Whether we are working with a circular mapping or a linear one, the end-effect of this calculation is that the grouping value starts by mapping to the constraint that contributes the least to the data point's relevance (has the lowest fitness), then gets "nudged"

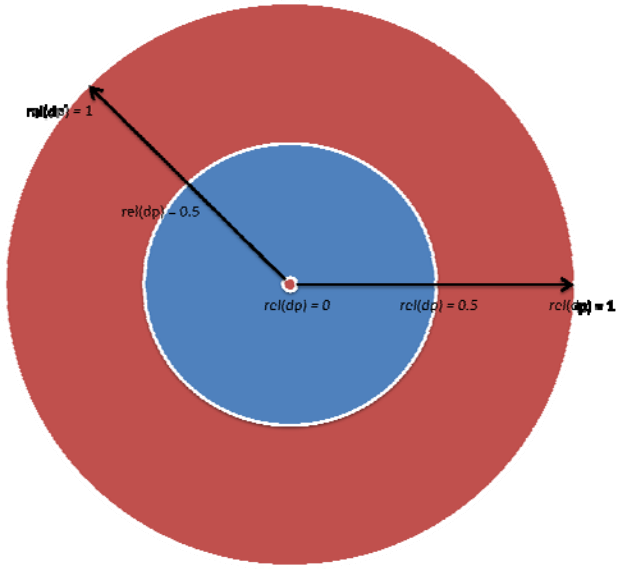

Figure 3: The "dartboard" of relevance used by the radial visual mapping. Note that two data points that have the same radius from the center of the circle have equal relevance. Thus, the most relevant data points will be the most central ones.

toward each of the remaining constraints based upon their contribution (or lack thereof) to the data point's relevance.

3.2.2 Clustering

Our "clustering" approach assigns a single, discrete range for each violated constraint. These constraints necessarily overlap, creating a 'map' of divided regions in which each region is associated with zero or more constraints. In this way, we can visually represent each constraint as an undivided region of the graph, and the constraints can logically combine in their overlaps.

We define "region fitness" to be the sum of the products of fitness and weight for each constraint represented by the region, to express how well a data point matches this subset of constraints:

$s =$ the subset of constraints represented by a region
$s \subseteq constraints$
$\text{region-fitness}(s, data) = \sum_{c \in s} fitness(c, data) \times weight(c)$

We implement this range satisfaction problem by building a list of regions, initially set to a single region with no representative constraints. In this way, we can "add" a constraint-satisfying region by specifying both the start and end regions with which to overlap, and then dividing the region twice (see Figure 3).

To build the map and determine overlap, we do the following process, using the entire set of data or a small representative sample as necessary:

1. Select the "best" constraint—the one with the largest sum of fitness values given the data, multiplied by the weight.
2. Test each possible start and end region to see which best fits the data.
3. Make that selection permanent (splitting the start region and end region as in Figure 3) and loop until all violated constraints have been assigned a location on the map.

We assign each data point to the region that has the highest fitness. In the event of equal region-fitness, we assign the data-point to the region with the lowest sum of weights of the constraints it represents, ensuring that (in case of a tie) each data point is assigned to the least-specific region that can contain it; otherwise, data points would accumulate in a random section, sometimes leading to improper implications about the data.

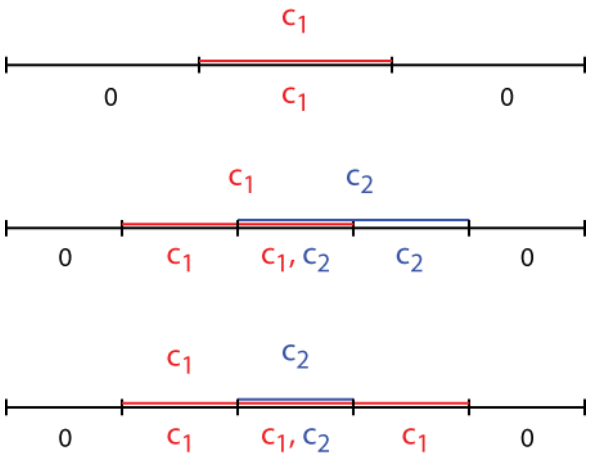

Figure 4: The region-splitting algorithm. The first line shows the initial map with one region placed, representing constraint c_1. The second shows the five regions resulting from some placing c_2 with some overlap, and the third shows five regions resulting from placing c_2 fully overlapping c_1.

Finally, we remove regions that have zero members, and plot the data points on the graph. In our implementation, we also added lines to further distinguish between the data points, and provided an equal-spacing algorithm that gives each region an area proportional to the number of data points accumulated in it, and prevents data points from overlapping. In the current implementation, X-location within a region is random, but related data points are located in the same region.

3.3 Visual Mappings

We looked at two different ways of mapping relevance to visual variables. The first uses a linear mapping of relevance to portray the fitness of data points. The second uses a radial mapping of relevance to portray the fitness of data points. These approaches are described here:

3.3.1 Linear

We attempt a "Linear" visual mapping. This mapping matches the existing model in which search engines return results. A large box at the top contains data points that match all constraints, to avoid a singularity; this top box does not have a well-defined X and Y coordinate, but instead is displayed randomly in order to assist in estimating the number of returned results.

The y-coordinate indicates relevance, as calculated before: as the data points are less and less relevant, they fall further down on the chart.

Our x-coordinate mapping provides two separate functions: To distinguish data points from one another, and to create a spatial grouping for results that are "related". In the linear visual mapping we implemented, we avoided the problem of redundant encoding by using the clustering algorithm above; using weighted averages in the linear function is not advisable, as a lack of "wrap-

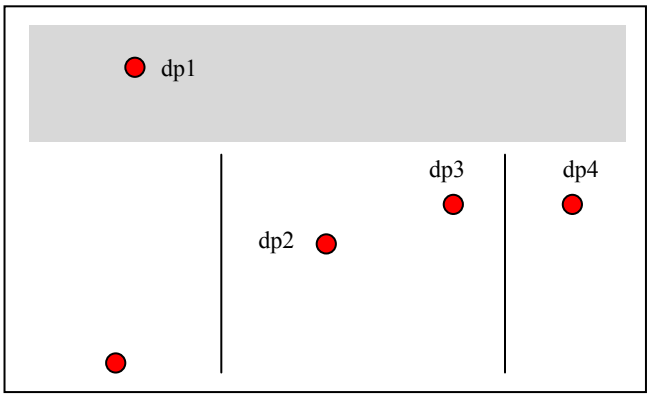

Figure 5: The linear visual mapping, displaying the clustering system. Point dp1 matches all constraints, and is displayed in the top box. Point dp2 and dp3 are grouped because they violate similar constraints, but dp3 and dp4 are the same distance from the top gray box because they are equally "relevant".

around" ensures that most weighted averages will tend to the center of the graph.

Finally, our implementation provided for a slight random color jitter in order to further distinguish data points during animation and between state transitions. In our current implementation, this is a random value assigned when the data set is loaded.

3.3.2 Radial

We attempt a "Radial" visual mapping. This mapping can be envisioned as a dart board, where the bullseye represents data points with 100% relevance. As data points move out from the center, they are less and less relevant. In Figure 3, we demonstrate the mapping of relevance to the radius of the data point's plotted location within a circle.

Specifically, for a circle of radius 1, relevance is mapped as follows:

$$radius(dp) = 1 - rel(dp)$$

To incorporate near-miss grouping, we use our averaging approach to derive a scalar that can be mapped to the angle (θ) of the data point in this visualization. We will give each constraint an equal share of the angle, then use averaging to determine where a data point should fall. In Figure 6, we see how data points with lowered relevance are mapped onto an angle in the radial visualization.

It is certainly the case that the error near-miss grouping value that is mapped to the angle may not be unique. As a result, it is impossible to say definitively that a point with a low relevance was caused by a violation of a particular constraint simply by looking at its mapping to an angle. However, "near-misses" occur most often when the relevance is high, and it is more important that the causes for such points' lowered relevance be clear. As the number of violated constraints decreases, the combination of averaging with a relevance becomes less ambiguous, making constraint violations easier to interpret. In the example in Figure 6, dp2 can be reliably said to have violated constraint c2 because it lies close to the center and in the middle of the angle range given to c2.

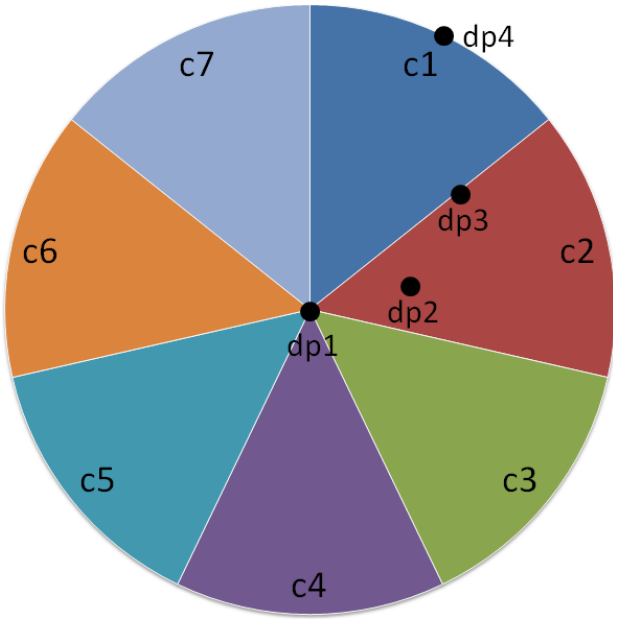

Figure 6: Demonstrates how data points are mapped based upon the constraints they violated. Here, dp1 has violated no constraints, so its angle component is undefined. Dp2 has violated only the c2 constraint. Dp3 has violated both the c1 and c2 constraints. Dp4 has either violated c2 and c7, c1 severely, or c2, c1, and c7 in equal amounts. These error grouping scores are not unique, but usually items with the same error score and radius pairing will have violated similar constraints.

3.4 Animation

Animation plays an important role in making Releviz effective. Our approach uses animation primarily for the purposes of helping users understand how changes to their constraints affect the relevance of the data points they are browsing. As users adjust constraints, they will be able to see the data points moving to their new locations in the visualization in real time, giving them valuable feedback on how the constraint impacts their query results.

Changes to constraints come in to forms, each of which may be animated differently:

- Discrete – these changes to constraints usually involve changing discrete values, such as the allowed categories of laptops or types of homes in a query. In this case, users would rarely make continuous gestures, where they would expect to see the visualization changing in real-time with their adjustments. Animation of discrete changes to constraints are handled by smoothly transitioning (accelerating and decelerating) each data point from its original position to its new one.
- Continuous – continuous changes to a constraint are changes where the user will expect to see changes to the visualization during their gestures, such as sliding a slider. Such changes must occur rapidly, and accelerating and decelerating data points during such animations keeps the visualization from reflecting the user's changes in real-time. Thus, user gestures to change continuous constraint parameters are animated by rapidly updating the visualization to always reflect the user's most recent input. Large changes that would disorient the user are treated as discrete, and are thus fully animated.

4 Results

Both the linear and radial visualizations were implemented using Adobe Flex 3.0 (Adobe, Inc.) and Flare (Flare | Data Visualization for the Web), a visualization toolkit for flash-based user interfaces.

We designed a common data set object model in ActionScript for use with Releviz, and which both visualizations (linear and radial) would be able to consume. We then implemented two different data sets that would allow our visualizations to operate on real-world data, and provided a reasonable set of constraints for each that users would be able to adjust. The first kept a local copy of laptop review data from PCWorld.com. The second used Zillow's Postings API to fetch data about homes for sale in a particular

county. Each of these thus represented real data for which filtering search engines were already in existence.

We also provided a common interface whereby constraints can provide UI, allowing the same data set to be reused with multiple visualizations.

Both visualizations are interactive, allowing users to see changes to their query reflected in real time through animation and providing some details about the individual data points to users in the form of tooltips or click handlers.

4.1 Linear

The linear visualization was an implementation of the linear mapping described above; the implementation tested uses the clustering grouping function also described in this paper. Figure 7 shows the final product in action.

4.1.1 Analysis

In this implementation of the user interface, the linear interface demonstrates a user interface consistent with the specification we described, but with significant room for further refinement.

Weaknesses:

- In the current implementation, regions are not labeled, making it difficult to discern how the search results are grouped.
- Also, in the current implementation, it is difficult to determine which series of regions represent which constraint. It would be a simple extension to allow a user to select a constraint and see a color or label marking each region representing that constraint.
- There is currently no way to graphically see how much a given constraint was violated. A good candidate for this display would be to provide a quantitative encoding (e.g. color saturation or size) for each data point when the chosen constraint is selected or highlighted.
- Animation does not perform as strongly as in the radial grouping, because the horizontal (grouping) encoding cannot wrap around the way the angular grouping encoding does in the radial graph.
- Within each group, there is no horizontal ordering, making the lines or other region-distinguishing marks important for the interpretation of the graph. In the future, X-ordering within regions may encode more details about related data points.

Strengths:

- Unlike the radial implementation, the linear implementation gives the same amount of horizontal space to data points that are very near misses, due to the center point of the circle representing the perfect constraint matches.
- The current implementation provides space for many perfect matches, unlike the current implementation of the radial graph. This, however, is a byproduct of these particular implementations; the radial graph is capable of reserving a threshold radius, and by reducing the size of the gray box, the implementation of linear gains this same "singularity" problem.
- The current implementation provides a unique grouping; groups that appear in the same region are absolutely similar, as opposed to the "false encodings" in the radial visualization.
- In the current implementation, data points are expressly prohibited from overlapping.
- The current visualization matches existing web search engines such as Google, where the most relevant results are listed at the top of the page. This should reduce the learning time involved in understanding the input.

4.2 Radial

The radial visualization was an implementation of our radial approach as described above. It uses a pie chart to visualize the constraint boundaries and a circular layout for plotting the data points. Figure 8 shows the radial Releviz visualization in action.

The radial visualization's animations occur over polar coordinates, so data points do more than simply move to the correct location. Rather, they follow an appropriate spiral to reach the correct radius and angle. This allows users to see how the radii and angles would have changed had each step between their constraint changes been made as well. This form of animation also helps users locate similar results, which will tend to move together as constraints change. This effect is immediately perceptible during animation using the radial visualization.

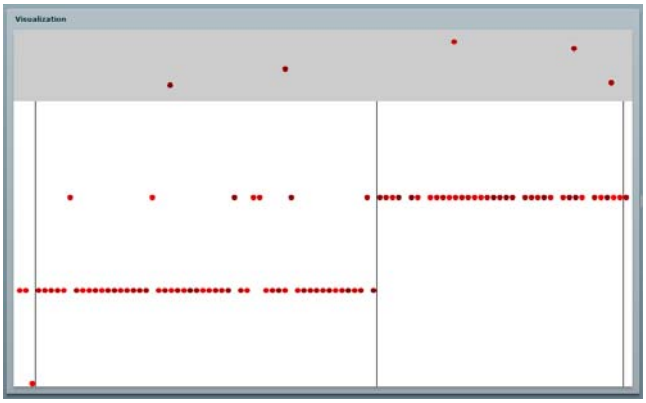

Figure 7: The linear visualization, running using the laptop data. Note the gray box at the top for matching constraints, and the vertical lines separating regions. The lines of data points are due to exclusive use of "discrete" constraint functions.

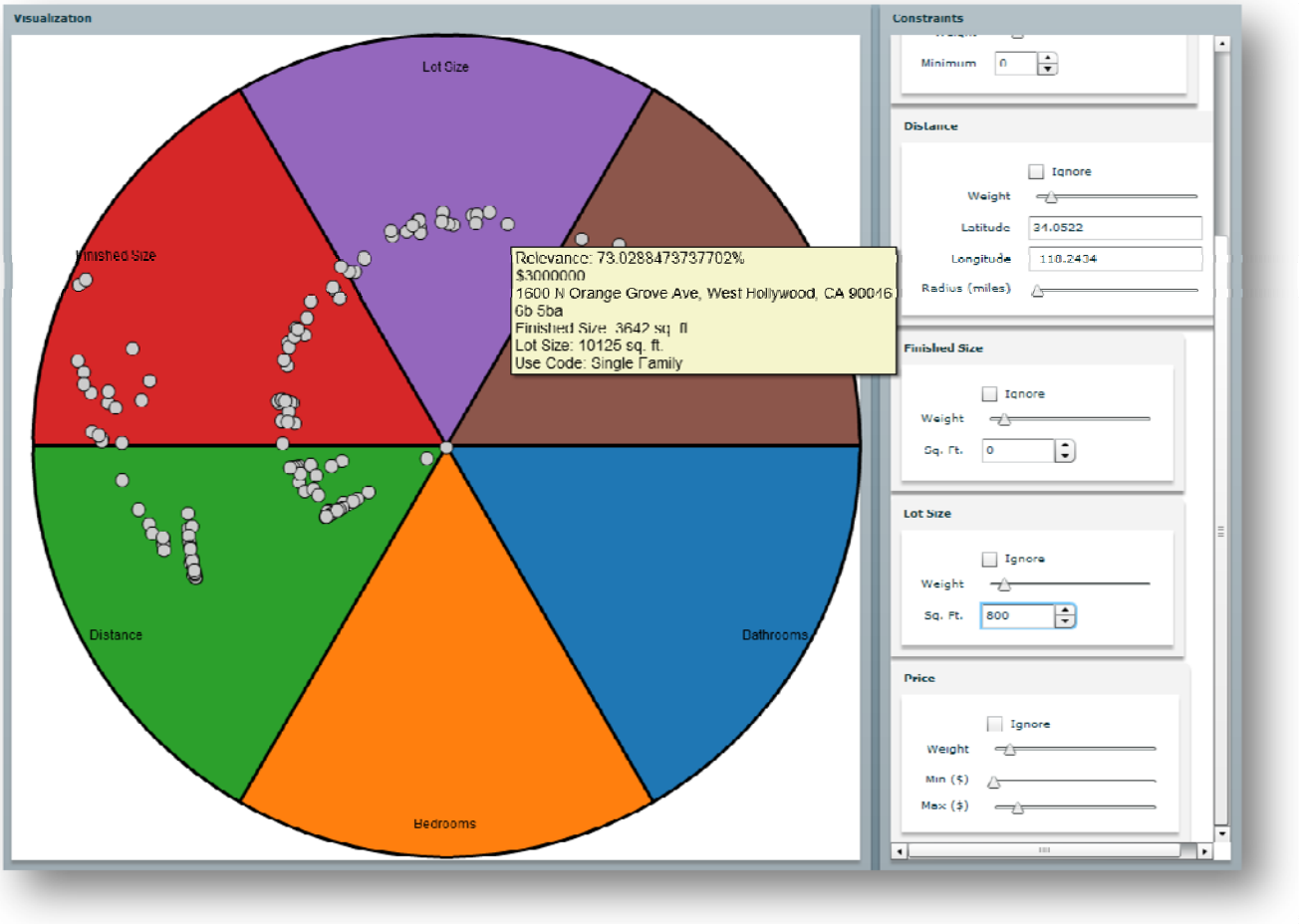

Figure 8: The Radial Releviz visualization in action, operating on live Zillow home postings data. Here, we can see that there are some perfect matches and a number of results that violated the distance, lot size, and price constraints. Also shown is the tooltip for one of the data points, where the calculated relevance is shown alongside the details for that data point.

4.2.1 Analysis

The radial visualization is by no means flawless, and there is certainly room for improvement. It has a number of strengths and weaknesses when visualizing near misses:

Weaknesses:

- Misleading segmentation of constraints – the radial visualization as it is currently implemented uses what is essentially a pie chart to draw the division of the circle into the constraints for the data set. Unfortunately, these give the user a false sense of the causes for lowered relevance when multiple constraints are being violated. This primarily occurs because the user has no sense that the wedges of the circle are anything but distinct categories into which data points may fall. A more appropriate coloring and rendering of this division is worth exploring, and might involve gradients to give the user a sense that these regions blend with one another.
- Visibility of relevant data – the radial visualization currently offers less overall space to more relevant data. Any data points that fully match a query will overlap at the center of the visualization, while the concentric circles with small radius offer little additional space for placement of the near misses. A possible remedy might be to reserve a larger region in the center for perfect matches to a query, then use the remaining radius (outside of the inner circle) to map relevance.
- Difficulty in comparison – small differences in radius are very difficult to detect if data points are not near each other. In Figure 8, it is unclear whether the data points strewn across the green, red, and purple wedges all share the same approximate relevance, or whether they are spiraling outward. While such comparisons are far less frequent with these types of data domains, there may be other types of data that do warrant closer comparison of the relevance of various data points.

Strengths:

- Centrality of high-relevance results – the radial visualization for Releviz has the benefit of placing highly relevant results at the center of the visualization for users. This helps users focus on perfect matches and near misses, while animations allow users to see how

changes in constraints bring data points to more central locations from afar.

- Real-world analogy – first reactions to the radial visualization in Releviz have been that the "dartboard" analogy "makes sense" for visualizing relevance, and making its use more familiar to users

5 Discussion

To no great surprise, the most challenging and prominent aspect of this project was the compression of *n* data dimensions into a single relevance and a single grouping value. While we believe that our solutions are novel and advanced, we are aware that this representation is a complex problem rooted in information theory, and needs plenty of work.

One of the most interesting aspects to the visualizations we developed is that data outputs incorporating "near misses" are inherently more useful when the user overconstrains his search results, allowing him a small or empty overlapping set that matches all constraints, and instead starting with a very small optimal set and moving outward. This is a potential change in behavior afforded by rich constraint specification and weighting UI along with a multivariate representation of the "top" search results.

However, in creating the visualizations, we discovered that controlling the weight of the constraints provides wildly different views of the same dataset and the same main constraints; therefore, the additional power must be weighed against the additional, arguably arcane UI presented to the user. A similar problem in the Attribute Explorer [Smith 2001] was solved by allowing user to order the constraints by personal priority, defining an order requirement instead of a numeric weight.

The functions we provided did not provide for control, with the exception of Price in the laptop data set could be defined as "hard" (discrete) or not hard (continuous). Though constraints could ostensibly be much more complex than this, our implementation is agnostic to the calculations involved.

The data sets we tested are all richly multivariate, and numeric in nature: They contain large amounts of data, and large numbers of numeric or true/false fields. We did not test the repercussions of data that contains duplicate or near-duplicate records, or data domains that contain text data; the latter, however, could still be expressed using a real-valued scalar between 0 and 1, and such could still be integrated into our system.

The use of animation, an afterthought, was unexpectedly successful with the weighted average clustering function, because the effects of a constraint could be isolated by disabling it (ignoring it, or setting its weight to zero) and re-enabling it.

In contrast, the linear visualization seems highly appropriate for fixed data delivery, as (between the two) it uses space and prevents overlap most effectively. In this way, the linear visualization could be applicable in systems that do not have graphics acceleration, or systems where interactivity is minimal or where the graph specified here is output into a fixed format (e.g. through a printer).

Finally, in observation of current trends, the creation and development of large and richly-described datasets—notably through XML and Web Services—will increase the need for domain-dependent search. For instance, a set of constraints could be generated from an XML Schema, or metadata document defining an XML structure. Though it appears that the current tool of choice is generic text search (such as Google), the creation of rich cross-site datasets highlights the usefulness of this project.

6 Future work

The most obvious extension to our current work would be to evaluate the visualization on a number of qualitative and quantitiative factors. We did not provide for user studies or user evaluation in this study, and we are eager to know whether users would appreciate data to be visualized in this way, and whether it allows them to access data faster.

To extend the visualization we offer, we seek further development on better grouping functions to group "near misses". For instance, a function that better groups closely-related results at the expense of slightly-varying relevance values could increase the usefulness of the visualizations we present. While our current grouping functions, clustering and weighted averaging, provide a starting point for compressing the multivariate constraint data into a single dimension of representation, they are by no means the best ways to provide for data interpretation.

Though the visualizations we provide are open to limited interactivity, additional research could provide a more space-efficient way of "drilling down" and refining the search. In particular, the concept of "data brushing"—graphical selection of multivariate data on an ordered X-Y plane—could be applied to select results.

In addition, the graphical representation can be extended into three dimensions, and by giving the data points additional encodings (such as size, shape, and color).

Though the radial visualization included tooltips to represent exact data, these visualizations could be paired with alternate UI controls (such as a map, or a sortable columnar data grid) which represent the same data and which interact with one another. This would allow the data to be used in a richer environment.

Finally, like in Mann and Reiterer [1999], our clustering algorithm can be applied to search result visualization, noting each search result keyword as a constraint. This is a direct extension of Mann and Reiterer's work, but provides for an intelligent form of grouping, and could also be used in further development of our radial mapping.

References

Adobe, Inc. Adobe - Flex 3. 2008 13 12 <http://www.adobe.com/products/flex/>.

Flare | Data Visualization for the Web. 13 12 2008 <http://flare.prefuse.org/>.

Google. 12 12 2008 <http://www.google.com>.

Zillow. 12 12 2008 <http://www.zillow.com>.

AHLBERG, C. AND SHNEIDERMAN, B. 1994. "Visual information seeking: tight coupling of dynamic query filters with starfield

displays." In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems: Celebrating interdependence* (Boston, Massachusetts, United States, April 24 - 28, 1994). B. Adelson, S. Dumais, and J. Olson, Eds. CHI '94. ACM, New York, NY, 313-317. http://doi.acm.org/10.1145/191666.191775

MANN, T., REITERER, H. 1999. "Case Study: A Combined Visualization Approach for WWW-Search Results." First publ. as paper in: *IEEE Information Visualization Symposium 1999, Late Breaking Hot Topics Proceedings*, Supplement to: *1999 IEEE Symposium on Information Visualization* (InfoVis 99), San Francisco, CA, USA, October 24-29, 1999. http://w3.ub.uni-konstanz.de/v13/volltexte/2007/3198//pdf/tm_hr_iv_1999.pdf

SEBRECHTS, M., VASILAKIS, J., MILLER, M., CUGINI, J., AN D LASKOWSKI, S. 1999. "Visualization of Search Results: A Comparative Evaluation of Text, 2D, and 3D Interfaces." http://zing.ncsl.nist.gov/cugini/uicd/sigir-paper-jun99.pdf

SMITH, ANDY. 2001. "Attribute Explorer: A Dynamic Query Mechanism." 2 April 2001. http://www.ibm.com/developerworks/library/us-atex/

WILLIAMSON, C., SHNEIDERMAN, B. 1992. "The dynamic HomeFinder: evaluating dynamic queries in a real-estate information exploration system." In *Proceedings of the 15th Annual international ACM SIGIR Conference on Research and Development in information Retrieval* (Copenhagen, Denmark, June 21 - 24, 1992). N. Belkin, P. Ingwersen, and A. M. Pejtersen, Eds. SIGIR '92. ACM, New York, NY, 338-346. http://doi.acm.org/10.1145/133160.133216