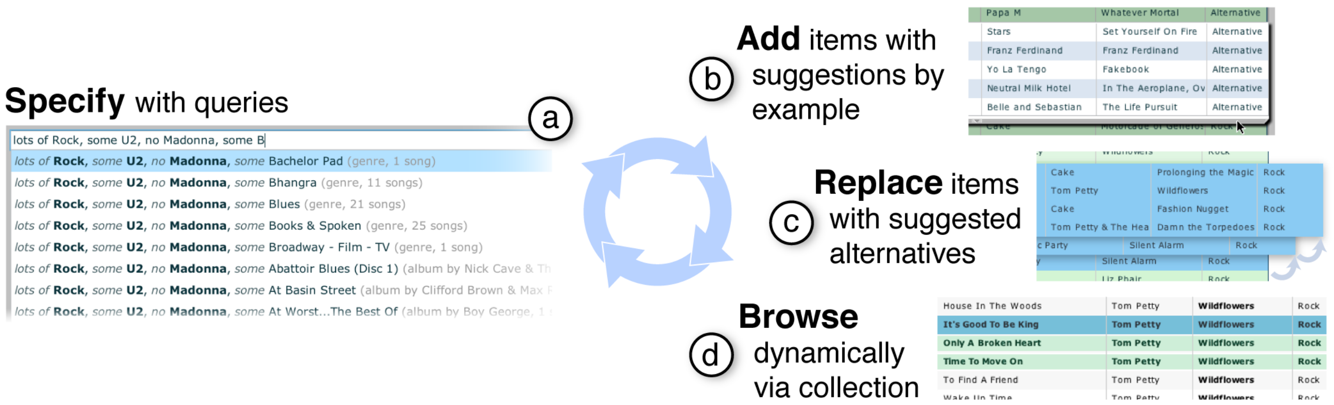
Creating Collections with Automatic Suggestions and Example-Based Refinement

Adrian Secord[1], *Holger Winnemöller*[2], *Wilmot Li*[2], *Mira Dontcheva*[2]

[1]New York University
Dept. of Computer Science
New York, NY 10003
ajsecord@cs.nyu.edu

[2]Adobe Systems, Inc.
Advanced Technology Labs
San Jose, CA 95110
{hwinnemo, wilmotli, mirad}@adobe.com


Specify with queries
a
Add items with suggestions by example
b
Replace items with suggested alternatives
c
Browse dynamically via collection
d

lots of Rock, some U2, no Madonna, some B

lots of **Rock**, *some* **U2**, *no* **Madonna**, *some* Bachelor Pad (genre, 1 song)
lots of **Rock**, *some* **U2**, *no* **Madonna**, *some* Bhangra (genre, 11 songs)
lots of **Rock**, *some* **U2**, *no* **Madonna**, *some* Blues (genre, 21 songs)
lots of **Rock**, *some* **U2**, *no* **Madonna**, *some* Books & Spoken (genre, 25 songs)
lots of **Rock**, *some* **U2**, *no* **Madonna**, *some* Broadway - Film - TV (genre, 1 song)
lots of **Rock**, *some* **U2**, *no* **Madonna**, *some* Abattoir Blues (Disc 1) (album by Nick Cave & Th
lots of **Rock**, *some* **U2**, *no* **Madonna**, *some* At Basin Street (album by Clifford Brown & Max R
lots of **Rock**, *some* **U2**, *no* **Madonna**, *some* At Worst...The Best Of (album by Boy George, 1 s

Papa M	Whatever Mortal	Alternative
Stars	Set Yourself On Fire	Alternative
Franz Ferdinand	Franz Ferdinand	Alternative
Yo La Tengo	Fakebook	Alternative
Neutral Milk Hotel	In The Aeroplane, Ov	Alternative
Belle and Sebastian	The Life Pursuit	Alternative

Cake	Prolonging the Magic	Rock
Tom Petty	Wildflowers	Rock
Cake	Fashion Nugget	Rock
Tom Petty & The Hea	Damn the Torpedoes	Rock
c Party	Silent Alarm	Rock
y	Silent Alarm	Rock
	Liz Phair	Rock

House In The Woods	Tom Petty	**Wildflowers**	Rock
It's Good To Be King	**Tom Petty**	**Wildflowers**	**Rock**
Only A Broken Heart	**Tom Petty**	**Wildflowers**	**Rock**
Time To Move On	**Tom Petty**	**Wildflowers**	**Rock**
To Find A Friend	Tom Petty	**Wildflowers**	Rock
Wake Up Time	Tom Petty	**Wildflowers**	Rock

Figure 1: Our system provides four mechanisms to help users create collections from personal media libraries: a keyword-query interface for generating automatic suggestions (a) and three example-based interaction techniques for iteratively refining a collection (b–d).

ABSTRACT

To create collections, like music playlists from personal media libraries, users today typically do one of two things. They either manually select items one-by-one, which can be time consuming, or they use an example-based recommendation system to automatically generate a collection. While such automatic engines are convenient, they offer the user limited control over how items are selected. Based on prior research and our own observations of existing practices, we propose a semi-automatic interface for creating collections that combines automatic suggestions with manual refinement tools. Our system includes a keyword query interface for specifying high-level collection preferences (e.g., "some rock, no Madonna, lots of U2,") as well as three example-based collection refinement techniques: 1) a *suggestion widget* for adding new items in-place in the context of the collection; 2) a mechanism for exploring alternatives for one or more collection items; and 3) a two-pane linked interface that helps users browse their libraries based on any selected collection item. We demonstrate our approach with two applications. *SongSelect* helps users create music playlists, and *PhotoSelect* helps users select photos for sharing. Initial user feedback is positive and confirms the need for semi-automated tools that give users control over automatically created collections.

ACM Classification: H5.2 [Information interfaces and presentation]: User Interfaces. - Graphical user interfaces.

General terms: Design, Human Factors.

Keywords: Collections, keyword search, constraint solver.

INTRODUCTION

Personal media libraries are an important part of daily life, as users amass music, photos, and videos. One common user task is collecting items from such libraries for a specific purpose. For example, people create music playlists for parties, exercise, and work. And after a trip, people select photos to share with friends, family, and colleagues. The challenge in accomplishing these types of tasks is choosing an appropriate *collection* (i.e., subset of items) from the hundreds or in some cases thousands of potential candidates in the user's entire *library*.

There exist two main types of interfaces for creating collections: manual and automatic. Faceted filtering interfaces [32, 14] have made it easier to manually select items from large libraries by allowing users to narrow down possibilities and focus on a specific group of items. For example, to create a playlist of Michael Jackson songs from the album "Thriller," users can sort or filter their entire music library along those two facets. However, users often want collections that vary along multiple criteria, such as playlists that include multiple artists and genres [13]. To create more varied playlists, users must query or filter by individual selection criteria, such as artist, and manually select songs, for the collection. Automatic solutions make use of examples [3, 9, 26] or generate collections by randomly sampling the library. In the example-based approaches, the user provides one or more example items and a collection is defined automatically based on item properties, such as genre, artist, year, etc. Although automatic interfaces can be effective for specifying goals that are hard to describe in words (e.g. give me more songs with this kind of melody), they give the user much less control over the final collection, as the user can only give examples and can't express goals, such as "I only want rock music."

To better understand user needs for working with collections, we surveyed previous work and carried out contextual-inquiry interviews with ten people. We found that when working with photographs and music, users have *concrete goals* in mind. For example, a playlist for work should have ambient music with few words, while a playlist for a party should have energy but include at most one or two songs from any specific artist. But, although users have concrete goals for their collections, they are often *satisficing*, i.e., they are not looking for a specific item and will often stop when they find a "good-enough" photo or song instead of continuing to search for the "best" item [4]. They also frequently get *sidetracked* by specific photos or songs (often those with personal significance) that inspire them to browse and add other related items. As a result, creating collections is typically an iterative process where users successively refine a set of candidate items until they are satisfied with the final collection. During this process, users sometimes change their minds and end up with a collection that is completely different than the one they initially had in mind.

Learning about user needs brought us to the realization that both automatic and manual approaches are necessary. On the one hand users are satisficing and in many cases are not too concerned about the specific items in their collections. This suggests the need for automatic methods that eliminate the manual effort of selecting individual items. On the other hand, users want to refine their collections based on the relationships between items, which indicates the importance of more user-directed refinement tools. Furthermore, since the selection criteria for some items are often highly subjective and deeply personal, it is hard to imagine how any fully automatic algorithm would be able to reliably create personally useful collections, even with multiple examples.

We propose a semi-automatic interface for creating collections that combines freeform text input with example-based iterative refinement. Users create collections automatically by defining high-level collection preferences using a textual query language. For example, the user might create a music playlist by typing "lots of rock, some U2, no Madonna, 2 hours" (Figure 1a). Our system transforms these preferences into constraints and then generates an appropriate collection using an off-the-shelf constraint solver [11]. To help users refine their collections, we present three different example-based collection editing techniques: an in-place *suggestion widget* for adding related items to the collection based on metadata from a selected item (Figure 1b); a mechanism for exploring automatically suggested *replacement alternatives* for one or more selected items (Figure 1c); and a *linked view interface* that allows users to pivot into their library based on items in their collection in order to browse for related items (Figure 1d).

We demonstrate our approach with two applications, one for creating music playlists (SongSelect), and one for creating collections of photos for sharing (PhotoSelect). We use SongSelect to explain our approach and perform user testing, whereas we use PhotoSelect to demonstrate the adaptation of our approach to a different domain. Early user feedback on SongSelect is positive. Users like the text interface and suggestion widget and are happy to employ automated playlist creation with more user control.

Our work makes the following contributions:

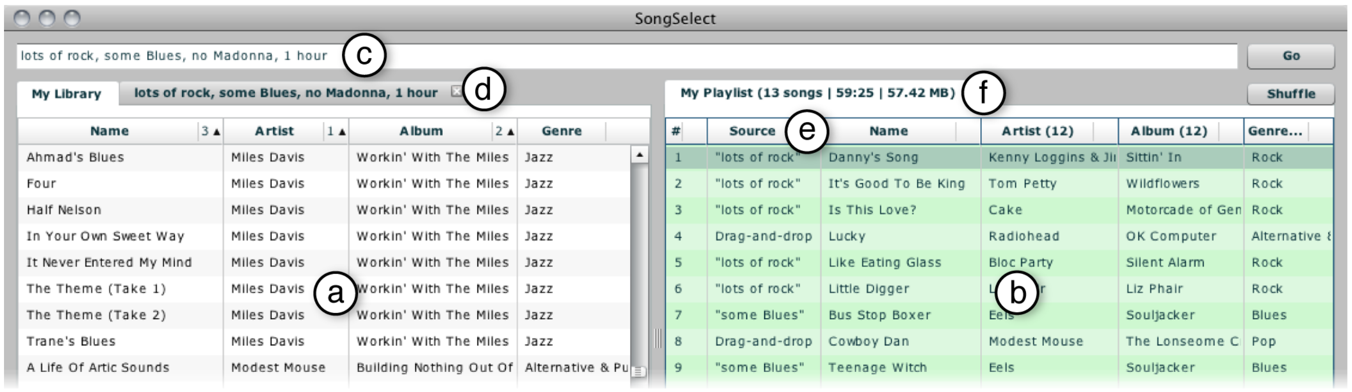
- A textual query language and autocomplete interface for specifying collection preferences
- A constraint-based technique for automatically generating collections based on these preferences
- Example-based user interface elements for (1) adding related items, (2) replacing items with similar alternatives, and (3) browsing the library relative to one or more selected items
- Design considerations for collection-oriented tools

RELATED WORK

Hansen and Goldbeck [13] pose creating collections as a recommendation problem, namely "how can we build recommender systems that can suggest collections, not just single items?" They propose three key aspects that must be considered when building systems for recommending collections: the value of the individual items, co-occurrence interaction affects, and order effects including placement and arrangement of items. In this work we focus on the interface for working with collections and propose a set of interaction techniques that together present the user with a semi-automated approach to building collections. Our text query interface allows users to specify co-occurance preferences, and the constraint solver resolves item interaction effects and satisfies collection-level goals. Our current design does not consider order effects.

Unlike today's faceted- and keyword-search interfaces, which require the user to specify selection criteria item-by-item in order to build a collection that varies across facets, early information retrieval systems returned sets, or collections, of items using boolean logic queries [29] expressed in languages, such as Structured Query Language (SQL). Some modern applications still support boolean logic queries via form-based interfaces (e.g., iTunes Smart Playlists). Boolean logic offers richer control over constructing a collection than simple keyword queries and can enable selecting sets of items that vary along multiple axes. However, aside from the difficulty in learning database languages, databases are optimized for generating collections by applying filters on an item-by-item basis. This focus on items makes it difficult to implement constraints that apply to the collection as a whole. A more natural approach is to treat the problem as a constraint-satisfaction problem on integer sets, where items are mapped to integers and constraints of all types can be specified directly [17]. This is the approach we take, as described in the Implementation section.

Our text interface is inspired by sloppy command interfaces like Inky [20], CoScripter [18], and Chickenfoot [5], but our



lots of rock, some Blues, no Madonna, 1 hour

lots of rock, some Blues, no Madonna, 1 hour

Name 3 ▲	Artist 1 ▲	Album 2 ▲	Genre
Ahmad's Blues	Miles Davis	Workin' With The Miles	Jazz
Four	Miles Davis	Workin' With The Miles	Jazz
Half Nelson	Miles Davis	Workin' With The Miles	Jazz
In Your Own Sweet Way	Miles Davis	Workin' With The Miles	Jazz
It Never Entered My Mind	Miles Davis	Workin' With The Miles	Jazz
The Theme (Take 1)	Miles Davis	Workin' With The Miles	Jazz
The Theme (Take 2)	Miles Davis	Workin' With The Miles	Jazz
Trane's Blues	Miles Davis	Workin' With The Miles	Jazz
A Life Of Artic Sounds	Modest Mouse	Building Nothing Out Of	Alternative & Pu

My Playlist (13 songs | 59:25 | 57.42 MB)

#	Source	Name	Artist (12)	Album (12)	Genre...
1	"lots of rock"	Danny's Song	Kenny Loggins & Ji	Sittin' In	Rock
2	"lots of rock"	It's Good To Be King	Tom Petty	Wildflowers	Rock
3	"lots of rock"	Is This Love?	Cake	Motorcade of Gen	Rock
4	Drag-and-drop	Lucky	Radiohead	OK Computer	Alternative &
5	"lots of rock"	Like Eating Glass	Bloc Party	Silent Alarm	Rock
6	"lots of rock"	Little Digger	L[illegible]r	Liz Phair	Rock
7	"some Blues"	Bus Stop Boxer	Eels	Souljacker	Blues
8	Drag-and-drop	Cowboy Dan	Modest Mouse	The Lonseome C	Pop
9	"some Blues"	Teenage Witch	Eels	Souljacker	Blues

Figure 2: The interface for SongSelect includes a table view with hierarchically sortable columns for browsing the library (a), a similar view for the current playlist (b) along with some playlist statistics (f), a textfield to enter the user query (c), tabs for previous query results (d), and a column documenting the inclusion reason for a playlist song (e).

keyword commands are simpler than those required for executable code [19]. With SongSelect users type commands like "mostly rock, some U2, no Billy Idol, less than 3 hours" and since most terms in the command contain facet values, the parsing becomes a matching task. There remain ambiguities, however, and we use a scoring scheme very similar to that of Inky to choose a single, unambiguous interpretation of the user's input [20]. Visual layout interfaces offer an alternative to text-based interfaces. For example, Musicovery [21] uses graph layout to display related music. This type of interface may be more suitable for exploring unfamiliar libraries and using relationships to discover new content.

Automatic example-based algorithms for creating collections continue to be improved [26, 8] and there are a number of domain-specific approaches [6, 16, 22, 23, 25, 24]. However, as we discuss in the next section, users want to iteratively refine collections and certain items can have deep personal meaning, which makes it challenging to completely automate the process.

USER NEEDS FOR COLLECTIONS

To motivate our interface design and better understand user needs for creating collections from personal libraries, we surveyed prior research on existing practices for creating playlists and sharing personal photographs [4, 10, 15, 31, 28, 27]. In addition, we conducted ten contextual-inquiry interviews to learn more about how users create collections (three interviews for music playlists and seven for photo sharing). For each interview, we first asked participants to describe their song/photo libraries and the tools they use to create collections. Then we asked them to create a collection as they would normally. The interviewer took notes and photographs of the participants, their libraries, and collections. Based on this research, we identify five key observations that inform the design of digital tools for creating playlists and photo sets. Although we focus on music and photographs here, we believe these observations may also apply to working with other types of collections, such as events, books, etc.

Users have specific goals when creating collections. Collections are often created with respect to events [4], situations [7], and people [30]. For example, playlists are often situation-specific (e.g. for driving, entertaining, exercising), and photo sharing can be person-specific (e.g. photos for friends, parents, colleagues). Tools should allow users to describe their preferences and help them understand when a collection fits or does not fit these goals.

People satisfice when creating collections. Bentley et al. [4] performed an in-depth analysis of user behavior with photographs and music and found many similarities in how users work with personal photograph and music libraries. They proposed that when working with photographs and music, users are often satisficing, i.e. they are not looking for a specific item and will often stop when they find a "good-enough" photo or song instead of continuing to search for the best item. They claimed that the reason people satisfice is because they have too many items to evaluate them all. This satisficing behavior implies that automation has a place in helping users create photo and music collections.

Users refine collections iteratively by exploring related items. Users start with an initial rough collection and then iteratively grow and shrink the collection as they encounter new items. For example, when creating a playlist, users may start by adding many potential songs to the playlist and then perform a second pass winnowing down the collection and replacing specific songs to make them fit better with the overall collection. They may look for different songs from the same album or a related song by the same artist. When working with photos users often look at sequences of photos and choose one from multiple similar shots. This iterative process is not merely a consequence of today's manual tools. As users create collections they stumble upon forgotten items that remind them of past events and lead them to explore related songs or photos. To support this iterative process, tools must support users' needs for browsing along varied dimensions and inter-item relationships.

Adding an item to a collection can be just as much about the item's fit in the collection as the item's individual quality [13]. For example, parents often want all children to appear in a set of photographs equally and may include a specific photograph to satisfy this requirement even if the photograph is not of high quality. A music playlist may have a maximum duration, and so a song may be included because it is just the right length. Tools for creating collections should

Item Preferences	
Quantifiers	*all*, *lots of*, *some*, *a couple*
Prepositions	*by*, *from*, *at*
Time	*before*, *prior to*, *after*
Set Preferences	
Durations	*< 30 min*, *about two hours*
Counts	*20 songs*, *more than five songs*
Sizes	*less than 250 Mb*, *2 gigs*

Table 1: Examples of modifiers accepted by SongSelect in addition to titles, artists, albums and genres, with selected examples (not an exhaustive list).

make it easy for people to manage these inter-item dependencies and overall collection requirements.

Collections that are shared with others can be highly personal [31, 28] and specialized. They often tell stories and include items that may seem unrelated to the casual observer. For example, music playlists are often created as gifts and include songs that are meaningful to the person giving or receiving the gift, such as the classical "mixed tape." Similarly, photos often tell personal stories of travels and events. It is hard to imagine that a fully automatic tool for creating collections could incorporate all of the subjective criteria that may be part of selecting items. Tools for collections need to be flexible to shifting user concerns and preferences.

USER INTERFACE

We first describe our interface design in the context of the music playlist application, SongSelect, and then discuss how we adapt this design to photo collections within PhotoSelect.

SongSelect

SongSelect has three areas, a text box at the top for specifying queries and two tabbed panes (Figure 2). The left pane displays the user's music library and query history as tabs (Figure 2a), while the right pane displays the user's playlist (Figure 2b). The user can start making a playlist by selecting songs from the library and dragging them over to the playlist, as is common in many music applications. Or the user may ask SongSelect for suggestions by specifying his preferences with keyword queries, such as "lots of rock, some U2, no Madonna." Given a user query (Figure 2c), SongSelect creates a playlist and stores it as a tab next to the user library in the left pane (Figure 2d). Note that these keyword queries are closer in flavor to the early set retrieval interfaces than today's keyword search interfaces, which retrieve lists of ranked documents. Also, a SongSelect playlist is analogous to a database view, although some user preferences can have relationships making them more complex to satisfy with traditional database query languages, such as "only one song per artist." We chose a freeform text interface over a traditional form-based GUI with checkboxes and sliders as textual queries allow users to specify only the relevant keywords without being overwhelmed by all the available options. For example, a music library with hundreds artists could result in a GUI with hundreds of checkboxes.

To support interactive refinement, we designed with examples in mind. SongSelect supports three example-based interaction refinement techniques. First, to better access the user's intent behind an example, we present a new widget, *the suggestion widget*, which allows users to flexibly add new songs to the playlist using song metadata, such as artist, album or genre. For example, if the user wants to add five more pop songs to his playlist, he can drag on "Pop" in the genre column (Figure 3). Second, the user can select one or more items in a playlist and look for alternatives by simply pressing the keyboard arrow keys. The selected songs are replaced in-place with alternatives derived from the user queries. Finally, to support library browsing as part of collection refinement, we link the library view to the playlist view and allow users to use any song as a pivot for finding related songs in the library (Figure 4).

Keyword queries

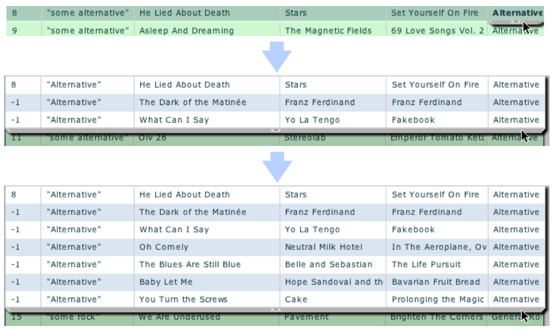
Since users are familiar with keyword search interfaces, which can take any string as input, we designed the playlist query text input interface to be as flexible as possible. Users can type as much or as little as they like. SongSelect will do its best to infer the user's intent and return a reasonable playlist. We define a user query as a list of criteria, such as "some rock, a lot of U2, no Alternative," that is specified as comma-separated *phrases*. Each phrase corresponds to a user criterion. Phrases can be as simple as the name of an artist, album, genre, or song. For more precision users can add modifiers, such as "mostly," "some," "a lot," "no." When a modifier is not specified, we assume the most general "some" modifier. Complex phrases include two or more facet values, such as "rock by U2" and "Michael Jackson before 1990." Phrases can also describe criteria about the playlist length, such as "max 2 hours, no more than 40 songs." Table 1 shows examples of the types of the modifiers SongSelect supports. If a phrase fails to parse properly, SongSelect alerts the user and asks him to correct or remove the phrase.

To aid the user in making queries, we designed an autocomplete widget that recognizes phrases and provides information about items in the library (see Figure 1a). The autocomplete widget includes genres, artists, albums, and songs. It lists the number of available items next to each autocomplete item. We found this critical for setting user expectations. If they only have one hip hop song, they should not expect to get a whole playlist of hip hop songs. Additionally songs and album entries include the artist name.

Finally, user queries are stored as tabs in the left pane (Figure 2d) so that user may go back to previous queries. This is useful as a history mechanism, but it also enables users to merge multiple playlists together or add new songs through keyword queries.

Suggestion widget

The suggestion widget (Figure 3) allows users to grow a collection in-place, removing the need to move back and forth between the library and the collection. The suggestion widget is visible to the user as a thumb and is available for all cells that are associated with more than one item. Since song names are typically unique, cells with song names do not include a suggestion widget, while an artist, album, and genre are not unique and can therefore be used to add more items. The user clicks on the thumb and drags down to add songs.



8	"some alternative"	He Lied About Death	Stars	Set Yourself On Fire	Alternative
9	"some alternative"	Asleep And Dreaming	The Magnetic Fields	69 Love Songs Vol. 2	Alternative

8	"Alternative"	He Lied About Death	Stars	Set Yourself On Fire	Alternative
-1	"Alternative"	The Dark of the Matinée	Franz Ferdinand	Franz Ferdinand	Alternative
-1	"Alternative"	What Can I Say	Yo La Tengo	Fakebook	Alternative
11	"some alternative"	Olv 26	Stereolab	Emperor Tomato Ketc	Alternative

8	"Alternative"	He Lied About Death	Stars	Set Yourself On Fire	Alternative
-1	"Alternative"	The Dark of the Matinée	Franz Ferdinand	Franz Ferdinand	Alternative
-1	"Alternative"	What Can I Say	Yo La Tengo	Fakebook	Alternative
-1	"Alternative"	Oh Comely	Neutral Milk Hotel	In The Aeroplane, Ov	Alternative
-1	"Alternative"	The Blues Are Still Blue	Belle and Sebastian	The Life Pursuit	Alternative
-1	"Alternative"	Baby Let Me	Hope Sandoval and th	Bavarian Fruit Bread	Alternative
-1	"Alternative"	You Turn the Screws	Cake	Prolonging the Magic	Alternative
15	"some rock"	We Are Underused	Pavement	Brighten The Corners	General Ro

Figure 3: The suggestion widget invoked on the facet "Genre" with value "Alternative," the progressive suggestions as the user expands the widget.

SongSelect limits the size of the expansion with the number of available songs.

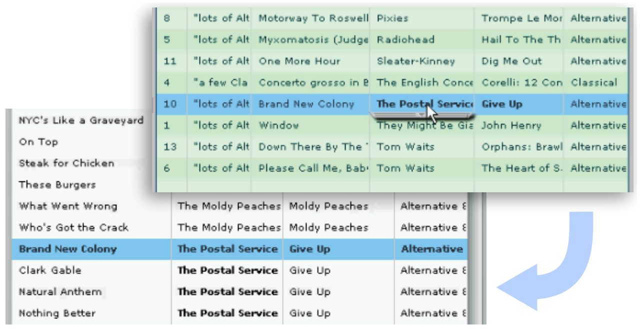
The suggestion widget presents a novel way to add items to collections and may be used independently from the keyword interface. It allows users to create playlists by example, but also to direct how that example is used.

Exploring alternatives

The user can browse the alternatives for a music playlist by selecting all or parts of the playlist and cycling through suggestions with the left and right arrow keys. Alternatives are generated through the phrases specified by the user. So if the user selects a few songs that were suggested because he asked for "some rock," he will see other rock songs that are not already in the playlist. If the user selects a song that was added to the playlist manually, through drag and drop, then SongSelect defaults to using the artist name and gives alternative songs by that artist. Songs that are added to the playlist through the suggestion widget are given a virtual phrase that corresponds to the basis for the suggestion. For example, if the user dragged the "rock" suggestion widget, the songs that are added are assigned the "rock" phrase.

Browsing libraries with linked views

Since users want to browse their libraries when creating collections, we designed a two-panel interface with the library on the left and the playlist on the right. The two panels are linked and the library is sorted and scrolled dynamically with respect to the playlist. When the user clicks on any song in the playlist, the library panel is sorted and scrolled to the song the user selected (Figure 4). The sorting is based on the column the user clicked. If the user clicked on the name of the song, the library is sorted by song name. For better orientation, we support secondary and tertiary sorting. If the user clicked on the genre of a song, the library is sorted first by genre, then by artist, and then by song. We specify default sorting rules for any metadata dimensions. Since not all song metadata is always visible, SongSelect includes a context menu that allows users to pivot on any of the metadata associated with a song, such as rating, number of times played, year released, etc.



8	"lots of Alt	Motorway To Roswell	Pixies	Trompe Le Mor	Alternative
5	"lots of Alt	Myxomatosis (Judge	Radiohead	Hail To The Th	Alternative
11	"lots of Alt	One More Hour	Sleater-Kinney	Dig Me Out	Alternative
4	"a few Cla	Concerto grosso in B	The English Conce	Corelli: 12 Con	Classical
10	"lots of Alt	Brand New Colony	**The Postal Servic**	**Give Up**	Alternative
1	"lots of Alt	Window	They Might Be Gia	John Henry	Alternative
13	"lots of Alt	Down There By The '	Tom Waits	Orphans: Brawl	Alternative
6	"lots of Alt	Please Call Me, Bab	Tom Waits	The Heart of S.	Alternative

NYC's Like a Graveyard			
On Top			
Steak for Chicken			
These Burgers			
What Went Wrong	The Moldy Peaches	Moldy Peaches	Alternative &
Who's Got the Crack	The Moldy Peaches	Moldy Peaches	Alternative &
Brand New Colony	**The Postal Service**	**Give Up**	**Alternative**
Clark Gable	**The Postal Service**	Give Up	Alternative &
Natural Anthem	**The Postal Service**	Give Up	Alternative &
Nothing Better	**The Postal Service**	Give Up	Alternative &

Figure 4: Selecting a song by *The Postal Service* in the playlist sorts the library by artist and scrolls to show other related songs.

Rich feedback

Providing effective feedback was a key design goal, since users can be confused by automatic suggestions. SongSelect offers a number of rich feedback features. First, the autocomplete widget sets user expectations, so that if a library has only one rock song, the user should not expect to create a playlist of only rock songs (Figure 1a). Second, the playlist includes a column, the "source" column, that describes why a song is in the playlist (Figure 2e). It lists a phrase, the facet value that was used with the suggestion widget, "drag-n-drop" for manually added items, or "auto fill" for items that were added to satisfy a length or duration criteria. Each playlist tab displays the number of songs, duration, and size of the playlist (Figure 2f). The grid columns also list the number of unique items in that column, so that the user can quickly evaluate the playlist. For example, Figure 2 shows a playlist that includes 13 songs from 12 artists and 12 different albums.

Since users are often concerned about how the items in a collection relate to one another, we added *brushing* to the playlist view. As the user moves the mouse over the playlist related items are highlighted. When the user puts the mouse over an artist name, such as "Cake," all "Cake" songs are highlighted. When the user moves the mouse over a genre, such as "Rock," all rock songs are highlighted. This type of feedback allows users to assess the playlist and decide whether they need to balance the collection with new songs.

PhotoSelect

To explore how these interaction techniques may be adapted to new domains, we implemented PhotoSelect, which allows users to create collections of photos from their photo library (see Figure 5a). Users often create collections of photos when they want to share photos with their friends, families, and colleagues. Similarly to SongSelect, PhotoSelect has a search box at the top with two panes. The library appears on the left and the collection appears on the right. To create a collection users can type queries such as, "mostly landscapes, none from the hotel, a few group shots," or they can scroll through their library and drag and drop photos into the collection. As with songs, users can use the suggestion widget to add new photos, scroll through alternatives, and dynamically sort and scroll their photo library. Since photos have an inherent time-based ordering, which is preferred by users [12], we se-

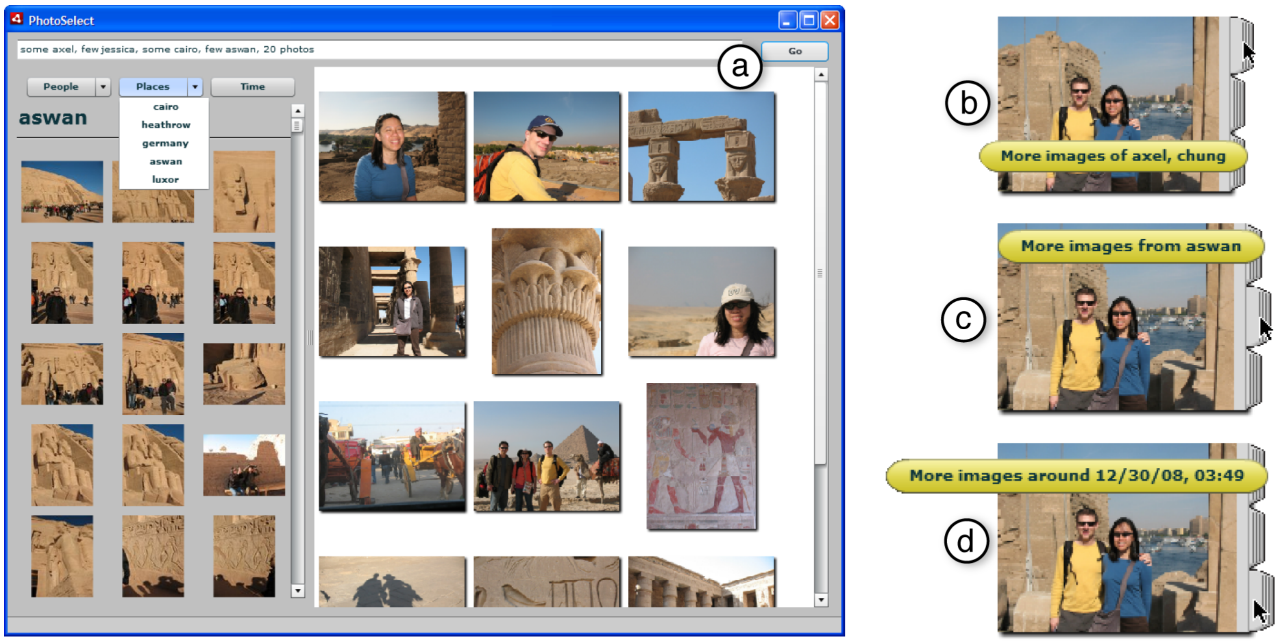

some axel, few jessica, some cairo, few aswan, 20 photos

Figure 5: (a) The interface for PhotoSelect includes a text box for specifying queries (top), a browsing pane for browsing the library (left), and the user's current collection (right). Suggestion widget tabs and tooltips for people (b), places (c), and time (d), respectively.

lect time as the default sorting criteria. PhotoSelect allows users to make use of any available metadata including people and places. Although people and place metadata is not part of all personal photo collections today, with advances in GPS technology and face recognition, we expect this metadata to become a ubiquitous part of personal photo collections.

There are some key differences between PhotoSelect and SongSelect. First, photographs are displayed in a grid instead of a list, which means that metadata incorporated with an image is not readily visible. Since the suggestion widget in SongSelect uses visible metadata, we redesigned it for PhotoSelect (Figure 5b,c,d). When the user hovers over a photograph, three thumbs appear. The thumbs correspond to people, places, and time. The user can add more photos based on the people, place, or time of any photo in the collection. In SongSelect we implemented the suggestion widget as a tray that appears on top of the playlist. One limitation of this approach is that as the user expands the tray and adds new songs, he hides some of the songs already in the playlist. In PhotoSelect the suggestion widget expands in place and moves the items following the selected item. This approach does not hide any elements but can cause extra shuffling, which may be distracting to the user.

We added additional keywords to our parser in order to support domain-specific keywords. For example, users may want to say "mostly landscapes" or "a few group shots." We expect that every new domain will have specialized keywords.

IMPLEMENTATION

In order to support different domains, we employ a client-server model, where the client contains domain knowledge and the server is a generic domain-independent constraint solver. The client transforms user input into a set of constraints using its domain knowledge, passes those constraints to the server, which then returns solutions to the constraint problem. The advantage of translating domain-specific user constraints into a domain-independent form is that we can use powerful off-the-shelf solvers for a variety of applications. The server is implemented in C++ as a web server and solves constraint problems using the Gecode constraint-solving toolkit [11]. For rapid prototyping and iteration, the user interfaces are written in ActionScript using Adobe Flex 3 [2] user interface framework and the Adobe Integrated Runtime (AIR) application framework [1].

Translating queries into constraints

As described in the user interface section, queries are comprised of a series of comma-delimited phrases, each of which produce a constraint on the items in the solution set.

Item constraints The most common type of constraint specifies a class of items in the library and the size or proportion of that class in the solution set. For example, "lots of Madonna" specifies that the class of songs by Madonna should be 50% of the solution set. And "no rock by U2 less than 3 minutes" translates into zero songs that contain "U2" as artist and "rock" as genre and are less than 3 minutes long. Table 2 lists how we translate different types of quantifiers. Users could also directly specify proportions or number of items. In our pilot study we found that users often didn't know exactly how many items they wanted in their collection and preferred using more vague query words.

Before passing the constraints to the server, the client goes through all constraints to normalize the proportions and check to see if any constraint can be trivially rejected. So, for example if the user asked for "some U2" but he only has two U2 songs, the constraint is modified to specify two U2 songs, instead of 25% of what could be a 2 hour collection. All proportions are normalized to add up to 100%. This ensures that

Quantifier	**Quantity**
all, *everything*, *every*,	100%
most, *mostly*, *most*	75%
lots, *lots of/from/at*,	50%
some, *some of/from/at*,	25%
few, *a few*, *a little*	10%
couple, *a couple*,	2 items
one, *one of/from/at*,	1 item
none, *nothing of/from/at*,	0 items

Table 2: We transform quantifiers into quantities in order to issue constraints.

the constraint solver does not return items the user did not request. For example, the query "mostly rock, some U2, some Madonna, no Michael Jackson, a few Billy Idol" becomes 55% rock, 19% U2, 19% Madonna, and 7% Billy Idol for our sample library.

Many simple queries can be fully or partially satisfied without querying the server, such as single-phrase queries ("some rock") and phrases that include or exclude items ("no U2"). However, most non-trivial queries do require the services of the server, such as queries whose phrases' domains overlap ("some Madonna, a few pop"), or phrases that specify a constraint on the set as a whole ("less than 30 minutes").

Set constraints Apart from constraining classes of items, the user can set constraints on the solution set as a whole, such as "some rock, $<$ 90 min". If a phrase only includes a duration, song count or size, we assume that it applies to the entire set. To construct the constraint we sum the individual item values, for example, the phrase "$<$ 90 min" translates to "the sum of song lengths $<$ 90 min." If no other set constraints are specified, SongSelect and PhotoSelect default to a length of 20 songs or photos.

Parsing The parser processes the user query after every keystroke and populates the auto-complete widget. The language for describing collection preferences is inherently ambiguous. Does the word "all" refer to the quantifier from Table 1, or does it refer to a (possibly-partial) title, album, artist or genre? What if the library contains both an artist and an album named "all?" We make use of a number of heuristics to disambiguate user queries but these heuristics can be tuned with user preferences. First, we use a non-deterministic parser which generates all possible interpretations of the user's query and applies a simple scoring function to rank the interpretations. The autocomplete widget uses this ranking in its display. For each parser token like *title* or *duration*, the scoring function assigns a numeric score based on the semantics of the domain. The score of the entire phrase is simply the sum of scores for each token. In the music domain, we score tokens in the following decreasing order: genre, artist, album, title, anything else. The interpretation with the highest summed score wins. In the case of a tie between two interpretations, the most general interpretation wins. We choose this ordering (general to specific) because we expect the user to query by describing the general properties of the solution set, not to pick out individual items. In the case where "all" matches both an album and an artist, the parser favors the artist, which results in more songs in the final solution set. In the photo domain, there is much less metadata, so ties are not very common. The parser scores people higher than places and places higher than other user-specified tags.

Implicit constraints

When users create collections, they have some implicit criteria that apply to most if not all collections. One such criterion is that they want their collections to be diverse (e.g. playlists should include multiple artists and photos collections should include photos from the entire library, not just a small subset). To create diverse collections, we configured the sever to explore the subspaces of constraint-matching sets at random, ensuring that a query of "some rock" would not return a playlist with only songs from the first rock artist in the library, for example. While this simple heuristic works fairly well, it does not ensure diversity. In the future, we plan to explore different strategies for ensuring diversity. Photo collections, for example, often include bursts of photos taken at approximately the same time when users captured several images in an effort to get "the best" one. It seems likely that users would only want one photo from such events and this could be encoded as an implicit constraint. Photo and music quality could also be part of the implicit criteria used to generate a collection.

Limitations

Perhaps the biggest limitation of our approach is that the constraint solver sometimes fails to find a solution and provides no feedback for the reasons of this failure. The SongSelect and PhotoSelect applications pre-process the query in order to pass along constraints that are satisfiable, but sometimes slight changes to the query can yield a better result. To improve our approach, we plan to explore techniques that allow for soft and hard constraints, and fail more gracefully. Since users are satisficing, returning a "good enough" collection is generally better than returning no result at all.

Our approach also does not consider ordering effects within a collection, but such preferences could be expressed as constraints in our current implementation. For example, the user cannot query for a playlist that starts with rock songs and ends with pop songs. An interesting future direction is to explore adding weights to constraints so that users can express queries such as, "prefer rock over pop music" or "pick outdoor scenes over indoor shots".

USER FEEDBACK

We report on user feedback from a pilot study of SongSelect with four participants. We plan a comparative evaluation with manual and automated approaches in the future. The study lasted an hour and included an introduction into the participant's existing music collection and playlists, a training task with SongSelect, and a playlist-creation task of their choice. The participants were between 24 and 40 years of age, two male and two female. All of them use Apple's iTunes software and have over 5000 songs in their libraries (one participant has over 70,000 songs). All of them reported making playlists, but their playlists had very different characteristics, and they varied in the amount of time they spent making them. Some made playlists as a way to put songs on their mobile devices, while others made themed playlists for fun or as part of an exercise class. Two of the participants were very concerned with the order of songs in their playlist, while the other two didn't care about the order at all (to the extent of deliberately shuffling the order of their playlist). All of them reported listening to music every day. All of the participants were aware of iTunes' Genius playlist creation tool, but perceived it to be biased towards popular music and not applicable to their personal libraries. Two participants mentioned using Pandora, an online streaming music service that creates streams of music similar to examples provided by the user. Two participants explicitly said that they enjoyed making playlists and enjoyed the results, but found the process tedious and wished that there were faster ways of generating quality playlists.

Overall, the participants reacted positively to SongSelect. Three commented that they liked the automated-but-malleable nature of SongSelect playlists. All four commented that they would like more automated tools for creating playlists but that they were concerned with the lack of control with existing systems for automatically creating playlists.

When asked about their favorite part of SongSelect, all participants agreed that the in-place suggestion widget was a great way to add new items and that being able to look at the library side-by-side with the playlist was useful. Two of the participants requested a modal suggestion widget so that they could add just the songs they wanted and not all suggestions. Three participants liked the ability to look through alternative item suggestions with the arrow keys but requested more control. Two participants wanted to be able to change the way SongSelect generated alternatives. One wanted to constrain the suggestions to a specific artist, while another wanted to change the genre of the songs.

All of the participants were able to use the text-based interface to generate an initial playlist. One participant said he preferred SongSelect's interface to iTunes' SmartAlbums interface because he didn't have to move the mouse, he could just type. All of the participants wanted to be able to use more metadata dimensions in the text box, in particular the album year, the dates the songs were added, and other, more subjective dimensions such as the "energy" of a song. Three participants remarked on the inaccuracy of the genre metadata in their libraries, and noted that the quality of the query results depended on the accuracy of the metadata.

Of the two participants who struggled with the text interface, one was confused by the results returned by the constraint server and felt that SongSelect was hiding things from her. This was likely due to a software bug and could be improved. The other participant thought of SongSelect's interface as a search interface not as a playlist creation interface and as a result was confused when SongSelect produced only 20 songs and not every matching song. This may be a more serious issue and one that requires careful design. By converting the search box into a query command interface SongSelect takes away the ability to search the library. This could be remedied, by explicitly including a Find feature that could be triggered with common key commands (Ctrl-F). However this confusion may speak to a larger problem: that users have expectations of what appear to be search text boxes. That is, queries in text boxes are for searching for specific items, not entire collections of items with interdependencies. This type of interface may require more training and a longer term field study to evaluate its effectiveness.

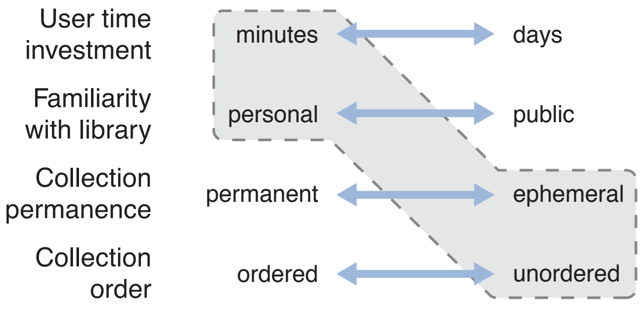


Figure 6: Design space for tasks with collections. SongSelect and PhotoSelect are designed for quickly creating ephemeral collections from personal libraries (shown in grey). SongSelect does allow users to manually order songs in the playlist but order is not used in generating playlists.

DISCUSSION: DESIGN CONSIDERATIONS

Creating playlists and sharing photos are just two situations among many in which users work with collections. For example, when planning a weekend trip, users are often faced with putting together an itinerary with many different activities such as meeting friends, visiting museums, and attending events. When selecting investments, users are often trying to pick a good mix of stocks while satisfying some high-level objectives. When remodeling a house there are many decisions that must be done in concert. For example, the choice of cabinets affects the counters and appliances.

Although in all of these situations, users iteratively create collections while managing multiple constraints, it is important to consider their differences. We propose a design space for discussing the differences and similarities between these problems, illustrated in Figure 6.

User time investment. Selecting stocks is a very different kind of activity than creating a music playlist. Although users are trying to make collections in both situations, the high cost of making a bad investment decision means that users are willing to spend days working on their collection. In contrast, creating a music playlist usually takes an hour or two, and some users are only willing to spend minutes on it. With SongSelect and PhotoSelect we designed for shorter tasks, although some of our participants mentioned that they would use SongSelect to edit playlists over time. Automated tools can shorten the time it takes to complete a task, thereby lowering the barrier to entry. In our interviews many users expressed wanting to create more playlists but being held back by the time investment.

Familiarity. Users approach personal libraries differently than less familiar public libraries, because they know what is available. In the two domains we consider, users can be expected to be familiar with the kinds of metadata available, such as artist names, genres, places, etc. Both SongSelect and PhotoSelect were designed for use with personal libraries, but we found that one participant acquired music at such a high rate (10 new albums per month) that their libraries were a mix between very familiar and unfamiliar songs. Although we believe that the interaction techniques we present in this paper could apply to unfamiliar content, further investigation is necessary.

Collection permanence. Users create collections for different reasons. For example, music playlists created for parties are often transient and get dated quickly. Investment and remodeling decisions, on the other hand, tend to be more permanent. SongSelect and PhotoSelect's designs focus on the collection creation aspects rather than longer-term maintenance aspects. We suspect that long-lived collections may require additional tools that help users track how a collection changes over time.

Item order. For many collections the order of items is important. Although SongSelect lets users manually order items, it does not consider ordering effects when it generates collections. We expected this to be a common complaint among our participants, however we found that many listen to playlists in shuffle mode.

CONCLUSIONS AND FUTURE WORK

We present a semi-automatic interface for creating collections of items from personal media libraries. Our interface combines free-form text input with example-based refinement. We present a *keyword query interface* that lets users create collections by specifying collection characteristics and propose three different example-based refinement techniques. First, we introduce the *suggestion widget* for in-place addition of new collection items that allows the user to flexibly add content using example metadata. Second, we allow users to *automatically scan through alternatives* for one or more collection items while retaining any user-specified collection characteristics. Finally, we use a *two-pane linked interface* to let users dynamically sort and scroll their libraries relative to a collection item. We demonstrate our approach in two applications—a music playlist creation application, SongSelect, and a photo set selection application, PhotoSelect—but we believe these interaction techniques are applicable to other domains. Initial user feedback confirms the need for semi-automated tools that let users direct automatic collection creation.

In future work, we plan to explore working with multiple collections. For example, when users create photo books or calendars, they look at previous photo collections they may have created and shared. Today's interfaces do not let users compare collections and easily see which items are included in multiple collections. More generally, today's folder-based interfaces assume and enforce that items are only present in one location. There are many situations in which this is not optimal, and users make many copies of the same item, which are hard to keep synchronized.

SongSelect and PhotoSelect allow users to create collections from their personal libraries, but users are often looking to make collections of non-personal content, like when they build travel itineraries, select stocks, or select furniture for their house remodel. Non-personal content is often scattered throughout multiple websites and part of the user's task is to find all necessary information. A tool for helping users build non-personal collections will have to be able to manage and integrate metadata from multiple locations.

ACKNOWLEDGMENTS

We would like to thank Wilson Chan for his help with the contextual inquiry inteviews. We would also like to thank all of our study participants and UIST reviewers for their time and efforts. The first author was supported by an Adobe University Collaboration gift and several Adobe internships.

REFERENCES

1. Adobe AIR application runtime library. http://www.adobe.com/products/air/, 2010. Retrieved June 21, 2010.

2. Adobe Flex web application software development. http://www.adobe.com/products/flex/, 2010. Retrieved June 21, 2010.

3. Apple. iTunes Genius. http://www.apple.com/itunes/features/#genius, 2010. Retrieved March 29, 2010.

4. Frank Bentley, Crysta Metcalf, and Gunnar Harboe. Personal vs. commercial content: the similarities between consumer use of photos and music. In *CHI '06: Proceedings of the SIGCHI conference on Human Factors in computing systems*, pages 667–676, New York, NY, USA, 2006. ACM.

5. Michael Bolin, Matthew Webber, Philip Rha, Tom Wilson, and Robert C. Miller. Automation and customization of rendered web pages. In *UIST '05: Proceedings of the 18th annual ACM symposium on User interface software and technology*, pages 163–172, New York, NY, USA, 2005. ACM.

6. Zeina Chedrawy and Syed Sibte Abidi. A web recommender system for recommending, predicting and personalizing music playlists. In *WISE '09: Proceedings of the 10th International Conference on Web Information Systems Engineering*, pages 335–342, Berlin, Heidelberg, 2009. Springer-Verlag.

7. Sally Jo Cunningham, Matt Jones, and Steve Jones. Organizing digital music for use: an examination of personal music collections. In *ISMIR*, 2004.

8. Marie desJardins, Eric Eaton, and Kiri L. Wagstaff. Learning user preferences for sets of objects. In *ICML '06: Proceedings of the 23rd international conference on Machine learning*, pages 273–280, New York, NY, USA, 2006. ACM.

9. James Fogarty, Desney Tan, Ashish Kapoor, and Simon Winder. Cueflik: interactive concept learning in image search. In *CHI '08: Proceeding of the twenty-sixth annual SIGCHI conference on Human factors in computing systems*, pages 29–38, New York, NY, USA, 2008. ACM.

10. David Frohlich, Allan Kuchinsky, Celine Pering, Abbe Don, and Steven Ariss. Requirements for photoware. In *CSCW '02: Proceedings of the 2002 ACM conference on Computer supported cooperative work*, pages 166–175, New York, NY, USA, 2002. ACM Press.

11. Gecode generic constraint development environment. http://www.gecode.org/, 2010. Retrieved March 29, 2010.

12. Adrian Graham, Hector Garcia-Molina, Andreas Paepcke, and Terry Winograd. Time as essence for photo browsing through personal digital libraries. In *JCDL '02: Proceedings of the 2nd ACM/IEEE-CS joint conference on Digital libraries*, pages 326–335, New York, NY, USA, 2002. ACM.

13. Derek L. Hansen and Jennifer Golbeck. Mixing it up: recommending collections of items. In *CHI '09: Proceedings of the 27th international conference on Human factors in computing systems*, pages 1217–1226, New York, NY, USA, 2009. ACM.

14. Marti A. Hearst. *Search User Interfaces*. Cambridge University Press, New York, NY, USA, 2009.

15. David Kirk, Abigail Sellen, Carsten Rother, and Ken Wood. Understanding photowork. In *CHI '06: Proceedings of the SIGCHI conference on Human Factors in computing systems*, pages 761–770, New York, NY, USA, 2006. ACM Press.

16. Peter Knees, Tim Pohle, Markus Schedl, and Gerhard Widmer. Combining audio-based similarity with web-based data to accelerate automatic music playlist generation. In *In Multimedia Information Retrieval*, pages 147–154, 2006.

17. Vipin Kumar. Algorithms for constraint-satisfaction problems: A survey. *AI Magazine*, 13(1), 1992.

18. Greg Little, Tessa A. Lau, Allen Cypher, James Lin, Eben M. Haber, and Eser Kandogan. Koala: capture, share, automate, personalize business processes on the web. In *CHI '07: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 943–946, New York, NY, USA, 2007. ACM.

19. Greg Little and Robert C. Miller. Translating keyword commands into executable code. In *UIST '06: Proceedings of the 19th annual ACM symposium on User interface software and technology*, pages 135–144, New York, NY, USA, 2006. ACM.

20. Robert C. Miller, Victoria H. Chou, Michael Bernstein, Greg Little, Max Van Kleek, David Karger, and mc schraefel. Inky: a sloppy command line for the web with rich visual feedback. In *UIST '08: Proceedings of the 21st annual ACM symposium on User interface software and technology*, pages 131–140, New York, NY, USA, 2008. ACM.

21. Musicovery. http://musicovery.com/, 2010. Retrieved June 22, 2010.

22. Steffen Pauws, Wim Verhaegh, and Mark Vossen. Music playlist generation by adapted simulated annealing. *Inf. Sci.*, 178(3):647–662, 2008.

23. Steffen Pauws and S. Van De Wijdeven. User evaluation of a new interactive playlist generation concept. In *In Proceedings of the ISMIR International Conference on Music Information Retrieval*, pages 638–643, 2005.

24. J.C. Platt. AutoAlbum: Clustering digital photographs using probabilistic model merging. pages 96–100, 2000.

25. R. Ragno, C. J. C. Burges, and C. Herley. Inferring similarity between music objects with application to playlist generation. In *MIR '05: Proceedings of the 7th ACM SIGMM international workshop on Multimedia information retrieval*, pages 73–80, New York, NY, USA, 2005. ACM.

26. Alan Ritter and Sumit Basu. Learning to generalize for complex selection tasks. In *IUI '09: Proceedings of the 13th international conference on Intelligent user interfaces*, pages 167–176, New York, NY, USA, 2009. ACM.

27. Lalatendu Satpathy, Saara Kamppari, Bridget Lewis, Ajay Prasad, Yong Woo Rhee, Benjamin Elgart, and Steven Drucker. Pixaura: supporting tentative decision making when selecting and sharing digital photos. In *BCS-HCI '08: Proceedings of the 22nd British CHI Group Annual Conference on HCI 2008*, pages 87–91, Swinton, UK, UK, 2008. British Computer Society.

28. Robin Sease and David W. McDonald. Musical fingerprints: collaboration around home media collections. In *GROUP '09: Proceedings of the ACM 2009 international conference on Supporting group work*, pages 331–340, New York, NY, USA, 2009. ACM.

29. Daniel Tunkelang. *Faceted Search*. Morgan and Claypool Publishers, 2009.

30. Fernanda B. Viegas. Collections: Adapting the display of personal objects for different audiences. In *Master of Science Thesis*. Massachusets Institute of Technology, 2000.

31. Amy Voida, Rebecca E. Grinter, Nicolas Ducheneaut, W. Keith Edwards, and Mark W. Newman. Listening in: practices surrounding iTunes music sharing. In *CHI '05: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 191–200, New York, NY, USA, 2005. ACM.

32. Ka-Ping Yee, Kirsten Swearingen, Kevin Li, and Marti Hearst. Faceted metadata for image search and browsing. In *CHI '03: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 401–408, New York, NY, USA, 2003. ACM.