Preference Elicitation for Interface Optimization

Krzysztof Gajos and Daniel S. Weld
University of Washington, Seattle
Krzysztof Gajos
Motivation: Supple Model-Based Interface Renderer

Hierarchy of State Vars + Methods

Screen Size, Model of Available Widgets & User’s Behavior Interaction (or that of a Group)

Decision Theoretic Optimization
Supple Output

[Gajos & Weld, IUI’04]
Supple Depends on Weights

Container factor weight: 0.0
Tab Pane factor weight: 100.0
Popup factor weight: 1.0
Spinner for integers factor weight: 5.0
Spinner (domain size) factor weight: 49.5238
Spinner for non-integers factor weight: 6.0
Slider factor weight: 45.7143
Progress bar factor weight: 0.0
Checkbox factor weight: 0.0
Radio button factor weight: 0.5
Horizontal radio button factor weight: 10.0
Radio button (>=4 values) factor weight: 0.0
Radio button (>=8 values) factor weight: 74.2857
Radio button for booleans factor weight: 14.2857

[Gajos & Weld, IUI'04]
RIA

U1 Speech: Show houses near Phelps Memorial Hospital

R1 Speech: I found 3 houses near Phelps Memorial Hospital

Graphics: Display (a)

U2 Speech: Tell me more about it

Gesture: Point to the house on the right

R2 Speech: Here are the attributes of this 6-bedroom home.

Graphics: Display (b)

[Zhou +, UIST’04; IUI’05]
\[ R(d, U) = w_1 \times K(d, U) + w_2 \times I(d, U) \]

\[ F_3(d) = \sum_{i} u_i \times R(d, x_i) \]
\[ \text{sim}(d_i, d_j) = 1 - \text{semanticDist}(d_i, d_j) \]

\[ r(d) = w_1 \times F_1(d) + w_2 \times F_2(d) + w_3 \times F_3(d) \]

\[ D(d, m) = S(d, m) \times C(d, m) \]

\[ \text{compatibility}(M) = \frac{1}{N^2} \sum_{i} \sum_{j} \text{compatibility}(m_i, m_j) \]

\[ \phi_w(d, m) = \text{importance}(d) \times \phi(d, m) \]

[Zhou +, UIST’04; IUI’05]
BusyBody

\[ ECI = \sum_i p(I_i \mid E) C(I_i) \]

- **Expected Cost of Interruption**
- **Probability of an interruptability state** \( I_i \)
- **Cost of interrupting if user is in state** \( I_i \)

[Horvitz +, CSCW’04]
Expected Cost of Interruption

\[ ECI = \sum_i p(I_i \mid E) C(I_i) \]

Probability of an interruptability state \( I_i \)

Cost of interrupting if user is in state \( I_i \)

Needs to be elicited from the user for every interruptability state \( I_i \)

[Horvitz +, CSCW’04]
LineDrive

\[ \text{score}(r_i, r_k) = d \cdot W_{\text{misplaced}} \]

\[ \text{score}(r_i, r_k) = d \cdot W_{\text{missing}} \]

\[ \text{score}(r_i) = |\alpha_{\text{curr}} - \alpha_{\text{orig}}| \cdot W_{\text{orient}} \]

\[ \text{score}(r_i, r_k) = d \cdot W_{\text{extended}} \]

\[ \text{score}(r_i) = \left( \frac{\left( l(r_i) - L_{\text{min}} \right)}{L_{\text{min}}} \right)^2 \cdot W_{\text{small}} \]

\[ \text{score}(r_i, r_k) = \min(d_{\text{orig}}, d_{\text{dest}}) \cdot W_{\text{false}} \]

[Agrawala +, SIGGRAPH’01]
Arnauld: A Tool for Preference Elicitation

Raise level of abstraction:
- instead of directly choosing weights…,
- designers now interact with concrete outcomes
Arnauld: A Tool for Preference Elicitation

- Instead of directly choosing weights..., designers now interact with concrete outcomes.
Arnauld: A Tool for Preference Elicitation

Optimizing UI Application

Arnauld

Weights

Raises level of abstraction:
- instead of directly choosing weights…,
- designers now interact with concrete outcomes
Arnauld: A Tool for Preference Elicitation

Raises level of abstraction:
- instead of directly choosing weights..., 
- designers now interact with concrete outcomes
Benefits

• Saves Developers Time
  – By factor of 2-3x

• Improves Quality of Weights
  – Learned weights out-perform hand-tuned

• Users May Want to Override Default Params
  – Individual preferences
  – Multiple uses
Our Contributions

• Implemented *Arnauld* system for preference elicitation
  – Applicable to most optimization-based HCI applications
  – Implemented on SUPPLE

• Based on two *interaction methods* for eliciting preferences

• Developed a fast *machine learning algorithm* that learns the best set of weights from user feedback
  – Enables interactive elicitation

• Investigated two *query generation algorithms*
  – Keep the elicitation sessions short
Outline

• Motivation

• Elicitation techniques
  – Example critiquing
  – Active elicitation

• User responses → constraints

• Learning from user responses

• Generating queries

• Results & Conclusions
Example Critiquing

![Stereo Interface]

- **Power**
- **Volume**: 4
- **X-Bass**: unchecked

**Tape Mode**
- **Tape 1**: selected
- **Tape 2**: unchecked

**Additional Features**
- **Reverse**: unchecked
- **Dolby Noise Reduction**: unchecked

**Playback Controls**
- **< Play**
- **Play**
- **Stop**
- **Pause**
- **<<**
- **>>**
Via Customization Facilities
Result of Customization
Provides Training Example!
Example Critiquing

👍 Exploits natural interaction
👍 Occurring during process of customizing interface

👍 Effective when cost function is almost correct

But…

👎 Can be tedious during early stages of parameter learning process

👎 Requires customization support to be provided by the UI system (e.g. RIA, SUPPLE, etc.)
Active Elicitation

In general, how do you prefer Level to be displayed?

Option A
Level 7

Option B
Level

Your choice:
- Option A
- Neither
- Option B

Submit
Active Elicitation
UI in Two Parts

In general, how do you prefer Level to be displayed?

Option A

Level 7

Option B

Level

Your choice:

- Option A
- Neither
- Option B

Submit

Structure provided by ARNAULD
Active Elicitation
UI in Two Parts

In general, how do you prefer Level to be displayed?

Option A

Level 7

Option B

Level

Your choice:

- Option A
- Neither
- Option B

Submit

Content provided by the interface system for which we are learning weights
Active Elicitation

👍 Convenient during early stages of parameter learning process

👍 Binary comparison queries easy for user

👍 Doesn’t require any additional support from UI system, for which parameters are generated

But

👎 Doesn’t allow designer to direct learning process

👋 Choice of Best Question is Tricky
Limitations of Isolated Feedback

Both examples so far provided feedback of the form

“All else being equal, I prefer sliders to combo boxes”

But what if using a better widget in one place
Makes another part of the interface crummy?!
In isolation, sliders are preferred

But using them may cause badness elsewhere
Situated Feedback with Active Elicitation
Situated Feedback with Example Critiquing
Summary of Elicitation Interactions

Isolated

Example
Critiquing

Active
Elicitation

Situated
Outline

• Motivation
• Elicitation techniques
• User responses → constraints
• Learning from user responses
• Generating queries
• Results & Conclusions
All systems studied had **linearly decomposable** cost functions; these can be expressed as:

\[
cost(\text{interface}) = \sum_{k=1}^{K} u_k f_k (\text{interface})
\]
From User Responses to Constraints

\[
\text{cost}() \geq \text{cost}() < \\

f_{\text{combo\_box}} = 1 \\
\quad f_{\text{combo\_box\_for\_number}} = 1
\]

\[
u_{\text{combo\_box}} + u_{\text{combo\_box\_for\_number}} \geq u_{\text{slider}} + u_{\text{horizontal\_slider}}
\]

\[
\sum_{k=1}^{K} u_k f_k (\text{interface}_1) \geq \sum_{k=1}^{K} u_k f_k (\text{interface}_2)
\]
Outline

- Motivation
- Elicitation techniques
- User responses $\rightarrow$ constraints
  - Learning from user responses
- Generating queries
- Results & Conclusions
Learning Algorithm

Given constraints of the form:

\[
\sum_{k=1}^{K} u_k f_k (\text{interface}_1) \geq \sum_{k=1}^{K} u_k f_k (\text{interface}_2)
\]

Find values of weights \(u_k\)

Satisfying a maximum number of constraints
And by the greatest amount
Our Approach

Use a max-margin approach

Essentially a linear Support Vector Machine

Reformulate constraints:

\[ \sum_{k=1}^{K} u_k f_k (interface_1) - \sum_{k=1}^{K} u_k f_k (interface_2) \geq margin + slack_i \]
Our Approach

Use a *max-margin* approach

Essentially a linear Support Vector Machine

Reformulate constraints:

\[
\sum_{k=1}^{K} u_k f_k (\text{interface}_1) - \sum_{k=1}^{K} u_k f_k (\text{interface}_2) \geq \text{margin} + \text{slack}_i
\]

Per-constraint slack that accommodates unsatisfiable constraints

Shared margin by which all constraints are satisfied
Learning as Optimization

Set up an optimization problem that maximizes:

\[ \text{margin} - \sum_i \text{slack}_i \]

Subject to the constraints:

\[ \sum_{k=1}^K u_k f_k(\text{interface}_1) - \sum_{k=1}^K u_k f_k(\text{interface}_2) \geq \text{margin} + \text{slack}_i \]
Learning as Optimization

Set up an optimization problem that maximizes:

\[ \min \; \sum_{i} slack_i \]

Subject to the constraints:

\[ \sum_{k=1}^{K} u_k f_k (\text{interface}_1) - \sum_{k=1}^{K} u_k f_k (\text{interface}_2) \geq \text{margin} + slack_i \]

Solved with standard linear programming methods in less than 250 ms.
Outline

• Motivation
• Elicitation techniques
• User responses → constraints
• Learning from user responses
• Generating queries
• Results & Conclusions
Generating Queries

• Important part of *Active Elicitation*
  – Like game of 20 questions, order is key
• Optimality is intractable
• Introducing two heuristic methods
  – Searching $\mathbb{R}^n$ space of weights
    • General method: applies to all opt-based UI
  – Search space of semantic differences
    • Faster
    • Requires tighter integration with the UI application
Generating Queries

• Why is it important?
  – Like game of 20 questions, order is key
• Optimality is intractable
• Introducing two heuristic methods
  – Searching $\mathbb{R}^n$ space of weights
    • General method: applies to all opt-based UI
  – Search space of semantic differences
    • Faster
    • Requires tighter integration with the UI application
Visualizing the search thru $\mathbb{R}^n$ space of weights

A binary preference question *cleaves* the space
Answering Question Creates Region

Preferred Region
Midway thru the Q/A Process…

What is the best *immediate* (greedy) question for cleaving?
Good Heuristics for Cleaving

1. As close to the **centroid** as possible

2. Perpendicular to the longest axis of region
Outline

• Motivation
• Elicitation techniques
• User responses → constraints
• Learning from user responses
• Generating queries
• Results & Conclusions
Informal User Study

• Four users
  – Two Supple developers
  – Two “sophisticated users”
    • I.e. programmers w/o Supple experience

• Developers asked to hand-build cost function
  – Hand-coding took 2-3x longer
  – Resulting function “wrong” 35% of the time!

• Using Arnauld to create cost function
  – Got robust cost function in 10-15 minutes
  – All said Arnauld much easier & more accurate
Learning Rate
Of different question-generation algorithms

![Graph showing the learning rate of different question-generation algorithms. The x-axis represents the number of elicitation steps, and the y-axis represents the ratio of learned function to ideal. The graph compares weight-based query generation, outcome-based query generation, and random query generation.](image-url)
Sensitivity to Noise

10% User Errors

Ratio of Learned Function to Ideal

Number of Elicitation Steps

Weight-based query generation; input error with $p=0.1$
Outcome-based query generation; input error with $p=0.1$
Random query generation; input error with $p=0.1$
Related Work

- **Gamble Queries**
  - \( \text{Outcome}_x \) vs. \( p\text{Best} + (1-p)\text{Worst} \)

- **Bayesian Learning**
  - [Chajewska, ICML’01]
  - Too slow for interactive use

![Graph showing average error as a function of number of samples.](image)
Conclusions

• Implemented **Arnauld** system for preference elicitation
  – Applicable to most optimization-based HCI applications
  – Saves developers time
  – Creates better weights

• Based on **two interaction methods**
  – *Example Critiquing*
  – *Active Elicitation*
  – Investigated two **query generation algorithms**

• Novel **machine learning algorithm**
  – Learns good weights from user feedback
  – Fast enough for interactive elicitation