Simple agglomerative visual grouping for ACT-R

John K. Lindstedt (j.k.l@rice.edu)
Department of Psychology, 6100 Main Street
Houston, TX 77005 USA

Michael D. Byrne (byrne@rice.edu)
Departments of Psychology and Computer Science, 6100 Main Street Houston, TX 77005 USA

Abstract

The human visual system tends to group related objects in the environment, allowing for more efficient use of attention, but sometimes leading to critical errors in executing a task. ACT-R's vision module currently has no concept of visual grouping, per se. We present both theoretical and practical motivations for imbuing ACT-R with visual grouping processes, and then walk through our implementation of a simple, minimally disruptive, generally applicable, and extensible system for assigning visual objects groups based on proximity, accounting for both spatial and temporal extension. Code is available and implications are discussed for employing the visual grouping system in ACT-R models. Finally, we discuss the system's limitations, extensibility, and its future development.

Keywords: cognitive modeling; visual grouping; UI; voting; human factors; model generalizability

Introduction

The human visual system employs visual grouping to more efficiently interact with the environment. Related elements are considered together, enabling our cognitive systems to shift attention to or away from groups of visual elements that are related in some meaningful way. But, as with any human system, there are kinks in how the visual grouping system works that cause us to make occasional, sometimes critical, errors in how we parse the world around us.

An example of how visual groups appear to impact task performance lies in the literature on voting ballots, wherein some research has been done demonstrating humans' sensitivity to the layout of the screen information over time. A Brennan Center for Justice report titled "Better Ballots" (Norden, Kimball, Quesenbery, & Chen, 2008) highlights a variety of voting ballot designs that produced voting errors, such as omitting a vote or voting twice. Figure 1 shows one such "bad" ballot, wherein so many voters in one precinct skipped a specific race on the second screen that it changed the outcome of the congressional election. Initially, one might suspect this error could be due to distraction by extreme salience, such as the bold, colored header drawing the attention down to the second race on the page. But according to Greene (2010), the effect appeared to be due the number or arrangement of races (and instructional elements) on the previous screen:

> While the highlighting of race headers did not reliably predict initial omissions of the critical race, the number of races presented on the first voting screen did: when voters saw two races on the first screen, they were less likely to omit the critical race on the following screen than were voters who saw only a single race on the first screen.

Indeed, it appears that the culprit in this particular ballot was the difference in layouts between the two screens, in which visual grouping processes presumably play a large role.

As the ACT-R cognitive architecture (Anderson, 2007) is often used in human factors applications for the purposes of evaluating interfaces and predicting performance, we believe this is precisely the sort of error ACT-R should have the capacity to predict. However, at present the ACT-R's "visicon" system, for all of its features, lacks any concept of visual grouping– it simply lists all of the available visual objects in isolation. Our intention with the present work is to implement a simple, consistent, and transparent method of grouping visual elements in ACT-R that works for any task and does so in a minimally invasive manner.

Motivations

Implementing a visual grouping algorithm for ACT-R has both theoretical and practical value. On the side of theory, these are well-documented processes that occur in human vision and if implemented correctly, can improve the validity and plausibility of models written in ACT-R. In terms of practical value, making visual groups available to modelers would offer more generalizable models, as well as a handful of conveniences for interacting with the visicon.

There is already much work on the human processes involved in segmenting a visual scene into separate visual groups. Rosenholtz et al. (2009) present a model that synthesizes a variety of visual features to simulate human visual grouping. Those features include proximity, similarity of color or luminance, continuity, and orientation, among others (and offers a nice review of relevant research on each). Their model:

> "... translate[s] a complicated two-dimensional image, in which segmentation is difficult, into a higher-dimensional representation where straightforward methods yield good results. Our particular technique uses a high-dimensional blur operation, which is simple to implement and understand."

Our work borrows– if not their specific techniques– their philosophy by taking a handful of the features available in the visicon and using them to perform a simple and clear process to segment them into visual groups.

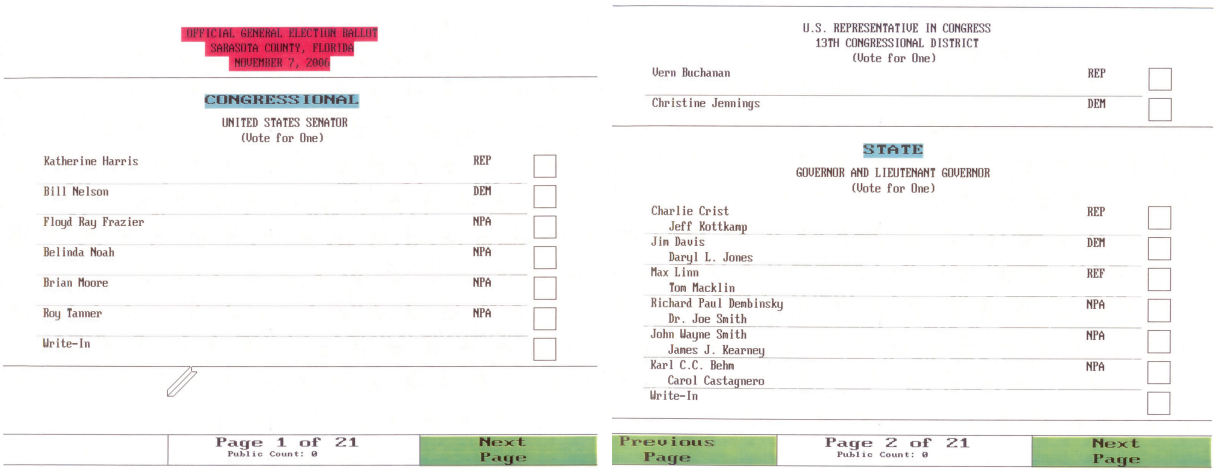

Figure 1: Two screen captures from the 2006 Sarasota County electronic voting system (first screen on the left, second screen on the right). So many voters failed to notice the race for U.S. Representative (top of right) that it changed the result of the election for that race. This error is thought to be due to the layout not accounting for human error due to visual grouping.

The introduction of a visual grouping system also fixes some problems and enhances some functionality in ACT-R modeling. Models often need to search the visual environment for particular elements in order to proceed with their tasks. Current modeling solutions for visual search tend to involve either (a) searching for the nearest unattended visual object, or (b) using specific SCREEN-X and SCREEN-Y coordinates to restrict that search. These solutions tend to suffice, but are problematic. In the first case, depending on the scan-path already taken, the model may find a visual location that is nowhere near the relevant section of the screen simply because it has already examined other nearby elements. In the second case, we are hard-coding knowledge of the screen layout, somewhat decreasing the model's cognitive plausibility and preventing it from generalizing across screen layouts without continuously spoon-feeding it special task knowledge. By introducing visual grouping, modelers can both restrict visual search to a particular region of the screen, and also build their models in such a way that the model should be able to locate the task-relevant regions of the screen regardless of their particular configuration and layout.

Visual grouping algorithm

In developing the visual grouping algorithm, we wanted a method that would require the fewest parameters from the modeler. An obvious, and potentially parameter-free, option would be the well-established *k*-means clustering method. Im et al. (2016) find success in developing a model of visual grouping using *k*-means. However, early in development we decided to go a different direction because: first, *k*-means and other density-based clustering methods are best suited for displays featuring masses of simple points, whereas most tasks modeled in ACT-R tend to involve more sparse, highly structured visual objects that possess width and height. Second, these clustering algorithms also tend to involve an element of randomness (e.g., bootstrapping) which can be quite frustrating for the purposes of writing and even understanding our own models. Though it is possible there are some instabilities in the way humans actually group visual objects, it is unclear whether the kind of uncertainty produced by these clustering algorithms would bear any resemblance to the uncertainty in the human visual system.

Instead, we decided to start with a method that is more consistent (for our modelers' sake) and transparent (for our developers' sake). In the same vein as Rozenholtz et al. (2009), we seek to apply relatively simple, general methods to the task of segmenting a visual scene into meaningful groups, provided with the representation of the visual objects already present in the ACT-R visicon. In the first version of our system, we account only for a visual object's positional features, SCREEN-X and SCREEN-Y, and its features of spatial extent, WIDTH and HEIGHT. We then further propose a system for how these visual groups propagate through time.

Interested readers can access our visual grouping code at the following github repository:

https://github.com/john-k-lindstedt/visual-grouping-actr

Installation is as simple as dropping the visual-grouping.lisp file into the user-loads folder within the ACT-R file tree. From there, the modeler needs only to specify the grouping radius and collision type desired, and the groups will be automatically generated and seamlessly added to objects in the visicon.

Integration with ACT-R

Our implementation of the visual grouping algorithm functions by intercepting the list of visual features for all of the elements of the scene used by ACT-R before the visicon is constructed, determining the visual groups for those elements,

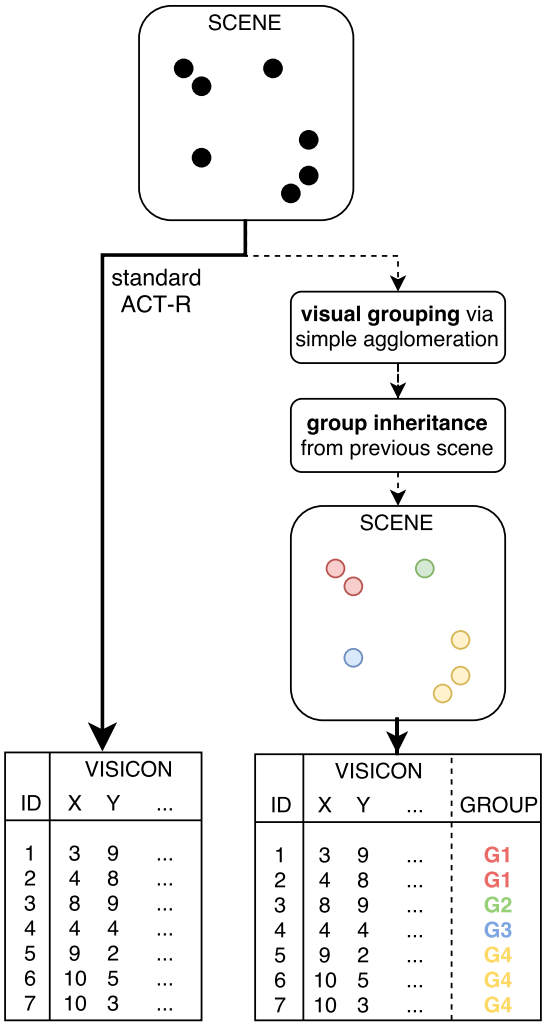

Figure 2: Flow of information from the visual scene to the visicon by default (left), and through our visual grouping system (right). The process is purely additive, giving each visicon entry a GROUP slot, and not disturbing any of the others.

and then returning that list intact but with the new group information attached. ACT-R then constructs the visicon as normal, but each visual location now has a GROUP slot that can be used in visual location requests like any other. Figure 2 illustrates this process.

Visual grouping by simple agglomeration

To determine which visual objects belong to which visual groups, we use a method called "simple agglomeration." Each group begins with a single visual element and recursively adds other nearby elements by checking for "collisions"; i.e., whether any nearby elements are within a "grouping radius" (a parameter presently left to the modeler to adjust). Two methods of collision detection are available: point-collision and box-collision. Ultimately, each object on the screen is assigned a unique and arbitrary symbol corresponding to its visual group.

Point collision method: Simple and fast grouping The point-collision method is minimalist: simply check whether the screen coordinates of two visual objects is within the grouping radius:

```
point-collision(obj1, obj2, radius):
1. if distance(obj1,obj2) < radius,
   obj1 and obj2 have collided,
   return true
```

This method is simple to calculate, but does not account for an object's size on the screen. As such, it is best used when objects are of similar sizes and shapes (i.e., singular characters or symbols), or when the screen is very dynamic and needs to update often. Figure 3 (top left) depicts this collision method.

Box-collision method: Accounting for extension in space Many objects in user interface displays have meaningful extension in space (text, buttons, images, etc), so we also implemented a box-collision method that accounts for an object's width and height by determining if the nearest point on one object's bounding box is within the grouping radius of the edges of another object's bounding box:

```
box-collision(obj1, obj2, radius):
1. target = the nearest location to obj1
   on the perimeter of obj2's bounding box
2. check point-collision(corner, target) for
   each corner of obj1's bounding box
3. check if target is within radius of the
   top, bottom, left, or right of obj1's
   bounding box
4. check whether target is overlapping with
   obj1's bounding box (just in case!)
5. if any of the above is true, obj2 and
   obj2 have collided, return true
```

The box-collision method requires more computation, but accounts for width and height in a more realistic way than the point-collision method. In practice, both the point-collision and box-collision methods are quite fast, and many tasks modeled in ACT-R (especially user interface tasks) involve static displays with relatively few elements. As such, we recommend the box-collision method over the point-collision for most applications. Figure 3 (top right) depicts this collision method.

Growing visual groups via simple agglomeration The simple agglomeration method begins with a visual group containing a single visual object and then "grows" that group by iteratively adding nearby points until none remain within the grouping radius. Then a new, unexamined visual object is selected, and the process is repeated until every object is assigned a group:

```
grouping(scene):
1. find an unexamined point, a, in the scene
2. find another unexamined point, b, in the
   scene
```

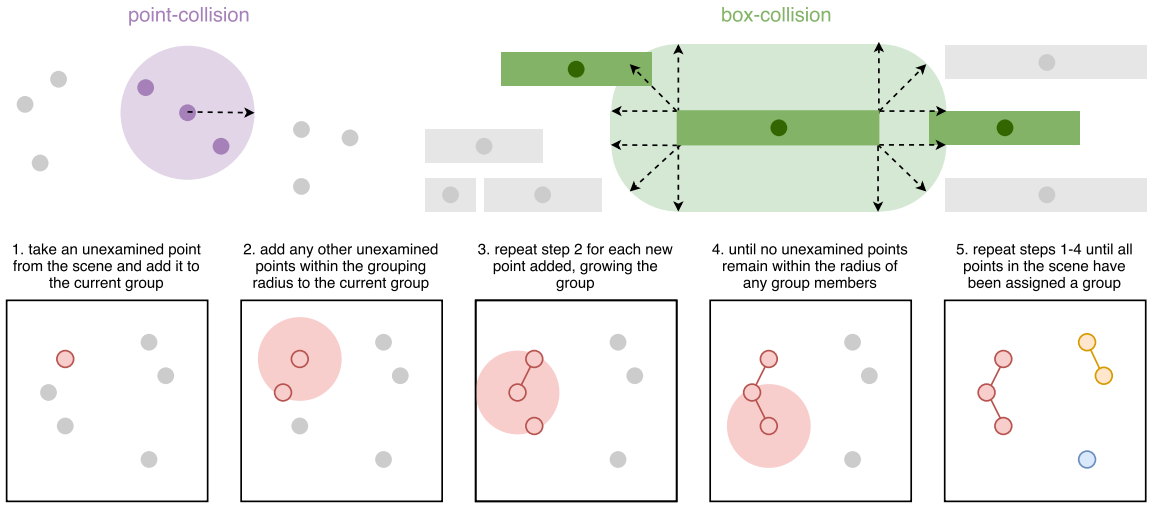

Figure 3: The two methods of collision detection available. Point-collision (top-left) is relatively fast and simple in that it involves a single comparison, examining whether the distance between two points is within the grouping radius. Box-collision (top-right) is more complex, as it involves several more comparisons and detecting the nearest point on a neighboring visual object's bounding box, but it allows for all objects' spatial extent to be considered when determining visual groups. The bottom half of the figure illustrates the steps of the simple agglomeration grouping method.

```
3. if collide(a,b,r): add b to the group
4. grow the group by repeating steps 2 and 3
   for each new point added until there are
   no more nearby points
5. assign all members of the group a
   new group-id
6. repeat steps 1-5 until all points in the
   scene have been examined, and all points
   now have an associated group-id
```

Figure 3 (bottom) depicts this process visually. Figure 4 shows a sample output of the visual grouping algorithm as applied to a screen of our VoteBox system. It is notable that the order in which points are added to groups is irrelevant– all point collisions are mutual and group growth is both strictly additive and exhaustive, so there is no "competition" to speak of between groups.

Inheritance: visual grouping extended in time

We also want to allow visual groups to extend in time, lest the model be forced to study a new set of visual groups every time the display is processed. To achieve this, we employ the same collision-detection method used in the simple agglomeration method to detect whether visual groups in a new scene overlap with any known groups from the previous scene. We achieve this with the following steps:

```
inheritance(current-scene, previous-scene):
1. count the number of unique group-wise
   overlaps between each pair of groups in
   current-scene and previous-scene
2. a current group inherits a previous
   group's group-id only if both the current
   group and a previous group exclusively
   overlap with one another
3. assign a new group-id if:
3a. the current group is new, i.e. it
    does not overlap with any previous groups
3b. the current group is the result
    of a merge, i.e. it overlaps with more than
    1 previous group
3c. the current group is the result
    of a split, i.e. it overlaps with a previous
    group that also overlaps with at least one
    other current group
4. if a previous group overlaps with no current
   groups, that group is dead and its group-id
   will not propagate forward in time.
```

Note that, at present, we assume that the only way a group can directly inherit a previous group's identity is for both the previous and current group to have mutually exclusive overlap. All other cases are considered to be "confusing," and generate new group IDs (likely triggering the model to need to re-study the screen layout before proceeding). The temporal duration and propagation of these visual groups is an open research question, which we discuss in the next section.

Figure 5 demonstrates how the visual groups would propagate in an example bearing some resemblance to the Sarasota ballots mentioned above. The ability for visual group identities to be inherited over subsequent scenes enables models to make the same kinds of errors as voters did in the Sarasota congressional election– when a task-critical section of the screen (i.e., a congressional race) directly inherits its group identity from a segment containing only task-adjacent information (headers or instructions), the model can mistakenly

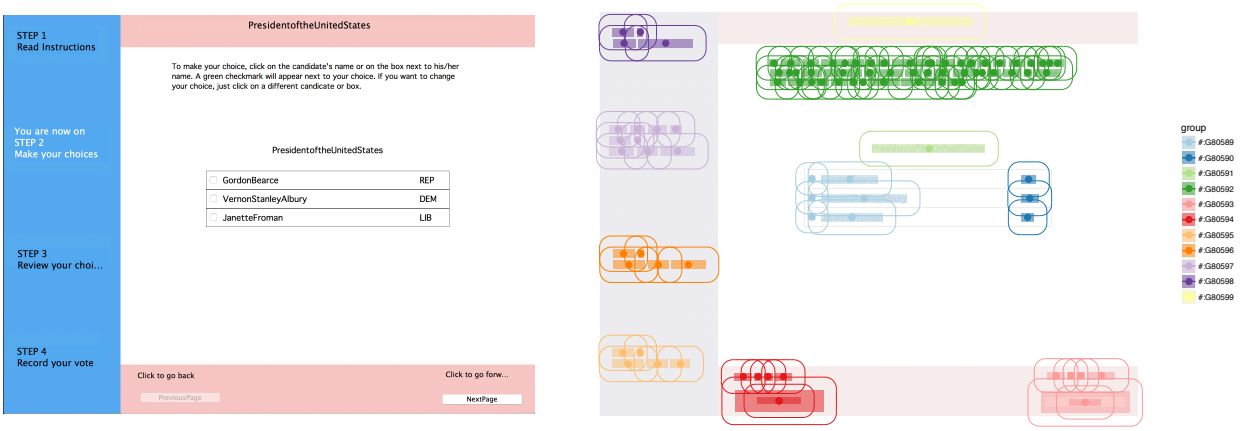

Figure 4: An example outcome of the visual grouping algorithm. The VoteBox task environment (left) is a relatively simple display with visual elements distributed somewhat sparsely across the screen. The right depicts the VoteBox task environment (dimmed) with the visual groups (colors) produced by the simple agglomeration method (grouping radius = 25 pixels) layered over top. For each visual location, the points represent the position, the shaded boxes represent the width and height, and the curved lines indicate the radius within which a box-collision occurs.

consider the task-critical segment as irrelevant and skip it entirely.

Implications for modeling

One of our goals in developing a visual grouping algorithm for the visicon was to provide functionality with as little disruption as possible to the way the ACT-R visicon works. As such, our implementation is designed to work exclusively with the information the visicon would already use, requiring no custom-built devices whatsoever. We also wanted to be sure the system was, by design, minimally disruptive to existing models. Because the algorithm does not remove or alter any information from the default visicon, modelers will not need to rewrite any of their models in order to install and start using the visual-grouping system.

Concerns for ease of use aside, modelers still must introduce new routines to their models to make use of the visual groups provided. It is important to note that the group IDs generated by the visual grouping system are intentionally generic and anonymous, meaning they strictly cannot impart any task-relevant information. Instead, models will need to use knowledge of the task at hand (such as labels of relevant buttons and information about the stimuli expected) to study the scene and commit to memory (i.e., the imaginal buffer) the relevant visual groups in the scene. Models will also need to return to this state to re-study the scene should the interface undergo any radical changes between parts of the task.

In exchange for the effort of having to study the screen, models will no longer need to be imparted with special knowledge about relevant screen locations, thereby becoming inherently more robust to changes in the layout of the screen–so long as the same buttons are used and the same kinds of information are presented, a model that studies the screen first will always know how to do its task.

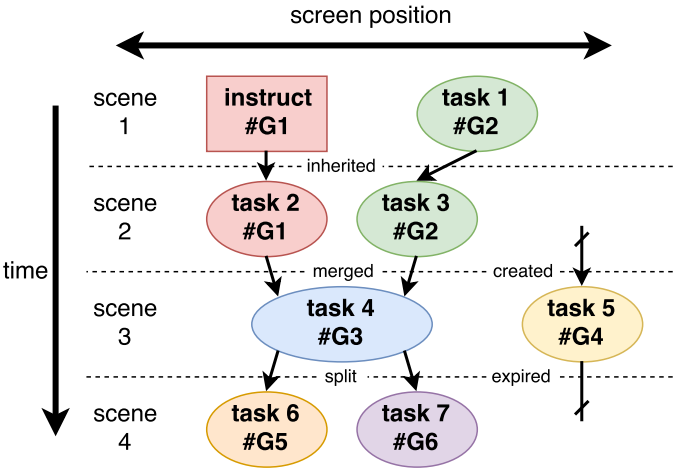

Figure 5: Sample scenario demonstrating how visual grouping would play out over multiple scenes in sequence. The first scene consists of an instructional portion and a task portion, assigned groups #G1 and #G2 respectively. In the second scene, task 2 inherits #G1 (the same group as the instructions, likely causing the model to skip task 2 entirely), and task 3 inherits #G2, despite being slightly offset from task 1. In scene 3, task 4 is considered the result of a merge between tasks 2 and 3, and is assigned the new group #G3, while task 5 is in a position that does not overlap with anything from the previous scene, so it receives the new group #G4. Finally, in scene 4, both tasks 6 and 7 are considered the result of a split from task 4 and receive the new groups #G5 and #G6, while group #G4 expires due to overlapping with no other groups.

Future directions

This visual grouping algorithm is meant to be a "first draft," initially aimed at addressing a handful of specific human error phenomena in the relatively simple domain of voting system usability. There is much work yet to be done, and many directions available for further development.

Currently, the appropriate grouping radius is left up to the modeler to determine, likely through trial and error (clearly not ideal). To provide a better default value for this radius– or an equation for finding one– questions such as "are groupings sensitive to scale?" and "what is the role of the retina?" will need answers either from a deeper delve into the visual grouping literature or by performing a battery of simple empirical studies.

Similarly, many questions remain about how exactly group inheritance functions. Currently, we assume that phenomena like splitting or merging two groups results in a sort of "confusion" for the model, so a new group label is applied. Instead it may be the case that there ought to be a "winner" in such an event: perhaps the largest group inherits? Or the group with the most overlap? Other questions about inheritance include: do group identities have any sort of "memory," recovering after a period of not being present? Are groups always constructed the same way, or is there an element of uncertainty we need to capture? Under what circumstances do we perceive visual groups as having "moved," rather than as two distinct groups? Do these processes function the same way when the screen is updating in real time as opposed to static, self-paced scenes? Simple empirical studies can illuminate how visual groups propagate forward in time, and eye tracking studies will help to illuminate what triggers humans to begin re-studying the screen.

We would also like to expand the scope of the visual features available for determining groupings beyond simple position and spatial extent. We believe our efforts are compatible with Rosenholtz et al.'s (2009) model, and as such we can tap into the existing literature on visual grouping, and their model in particular, to achieve even more robust visual groupings using a wider array of visual features, such as: contrast, color, luminance, orientation, continuity, etc. Representing these features, as well as SCREEN-X and SCREEN-Y, as a vector of numerical values would allow us to use a nearly identical measure of euclidean distance to detect "collisions" and classify groups in more interesting ways than simple spatial proximity.

We will also need to investigate to what extent humans employ hierarchical grouping. Dividing lines and containing boxes are commonly used by interface designers to partition the screen (like those shown separating the sections in Figure 1), but there are clearly smaller sub-groupings of information within those regions. Currently, our system can achieve something to the effect of identifying super- and sub-groupings by using more than one visual grouping radius, though the method for determining those radii will require thorough exploration.

Conclusion

Visual grouping processes help humans make sense of their visual environment. ACT-R, by default, lacks any sense of visual grouping. We attempted to remedy this because both (a) visual grouping is a well-documented phenomenon that explains certain elements of human behavior, and (b) the use of visual grouping offers modelers some practical conveniences and improvements to the generalizability of their models.

The visual grouping system we have implemented is a first pass at the problem, but still succeeds in many ways: the system is straightforward (we believe), it is minimally disruptive to existing models, it is stable in that it always produces the same visual groupings given a particular display, and it is extensible. In particular, we see the ability for models employing visual grouping to generalize across different screen configurations as a contribution to the overall plausibility of cognitive models in ACT-R.

Future empirical work and reviews of the literature will address: the extent to which the model can reproduce human errors on voting ballots, investigating the mathematical nature of visual grouping, and expanding the capabilities of the system to fit the needs of the modeling community. To that end, we extend an open invitation to interested modelers to propose– or implement– any desired additional features or alternative methods as we continue to develop this visual grouping system.

Acknowledgments

This research was supported by grant #CNS-12550936 from the National Science Foundation. The views and conclusions contained herein are those of the authors and should not be interpreted as representing the official policies or endorsements, either expressed or implied, of NSF, the U.S. Government, or any other organization.

References

Anderson, J. R. (2007). *How can the human mind occur in the physical universe?* New York: Oxford University Press.

Greene, K. K. (2010). *Effects of multiple races and header highlighting on undervotes in the 2006 sarasota general election: A usability study and cognitive modeling assessment*. Doctoral dissertation, Department of Psychology, Rice University, Houston.

Im, H. Y., hua Zhong, S., & Halberda, J. (2016). Grouping by proximity and the visual impression of approximate number in random dot arrays. *Vision Research*, *126*, 291–307.

Norden, L., Kimball, D. C., Quesenbery, W., & Chen, M. C. (2008). *Better ballots.* New York, NY.

Rosenholtz, R., Twarog, N. R., Schinkel-Bielefeld, N., & Wattenberg, M. (2009). An intuitive model of perceptual grouping for HCI design. In *Proceedings of the 27th international conference on human factors in computing systems - CHI 09* (p. 1331). New York: ACM.