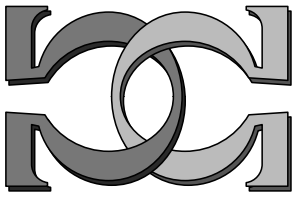
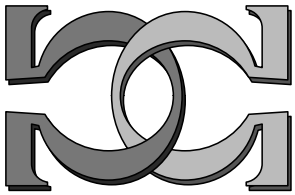
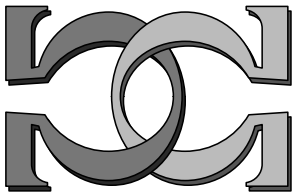
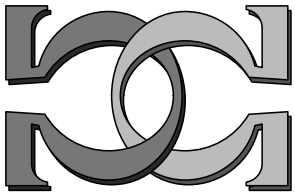


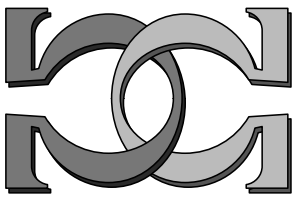




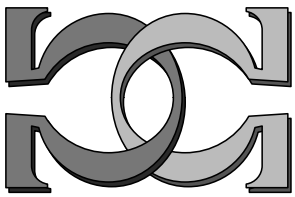
CDMTCS Research Report Series



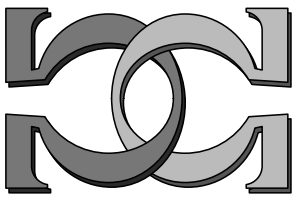
Proceedings of the Computer Graduate Workshop 2007



J. Teutenberg (ed.)
University of Auckland, NZ



CDMTCS-321
April 2008



Centre for Discrete Mathematics and
Theoretical Computer Science

CODEANNOTATOR: DIGITAL INK ANNOTATION WITHIN ECLIPSE

Xiaofan Chen Dr. Beryl Plimmer

ABSTRACT
Programming environments do not support ink annotation. Yet, annotation is the most effective way to actively read and review a document. This paper describes a tool, CodeAnnotator, which integrates annotation support inside a programming development environment (PDE). This tool is designed and developed to assist programmers in directly annotating on code within the programming development environment. Programmers will benefit from a more intuitive interaction space to record notes and comments just as they would on paper documents.

Author Keywords
Digital ink annotation, annotation, sketch, eclipse

1. INTRODUCTION
Annotating documents when reading assists readers by enhancing reading-comprehension activities [5, 6]. Moreover, reading documents that have been annotated by previous readers improves subsequent readers' understanding and recall of the emphasized items [15, 18]. People are accustomed to make annotations with pens on paper documents. More recently, researchers have been exploring adding digital ink functionality to provide annotation over digital documents [8, 9, 11, 12]. However, programming development environments (PDE) do not support digital ink annotation. People cannot make annotations directly inside the PDEs without the addition of annotation support.

This project aims to provide people with an electronic environment to annotate code documents within a PDE. We designed and developed a tool, CodeAnnotator, to support users to directly annotate over code within Eclipse. As this tool integrates annotation support inside the PDE, it offers additional advantages inherent from typical code review processes, as explained by Priest [12].

With CodeAnnotator after loading a code file in a PDE, users can write annotations using a digital pen while also having all the functions offered by the PDE, such as debugging, compiling, and executing. Therefore, users have both PDE and annotating support.

This paper begins with a summary of related work on digital ink annotation. Then it presents the design requirements and implementation of CodeAnnotator, followed by conclusions and future work.

2. RELATED WORK
Most early annotation tools, such as Wang Freestyle [3] and XLibris [4, 14], only support annotating over static documents. These tools provide users with simple free-form ink annotation and an interface and features similar to that of paper. They are self-reliant, independent from environments and may be used by people when editing and reading digital documents.

With the experiences researchers gained from development of self-reliant annotation tools, tools that integrate annotation support insider other developing environments have emerged. For instance, people can annotate over Word documents directly in Microsoft Word [7]. MADCOW [2], WPM [17] and Web annotation tool [13] assist users in making annotations in web browsers.

However, code documents have a unique feature that other types of documents do not have: code is non-linear; it is arranged in logical classes and procedures that are not intended to be read sequentially like a book [12]. Some digital ink annotation tools were designed and developed to meet the unique features of code. Penmarked [11] is a tool for annotating and marking student assignments through tablet PCs. Teachers can read the assignments and add digital ink annotations directly on the document. The limitations of this system are: it only deals with static documents and opens files as a text file, so code files (Java, C# etc.) can not be compiled or run under this system. Gild [10] is a set of plug-ins for Eclipse to mark assignments written in Java. However, it doesn't support digital ink annotation. RCA [12] is a code annotation tool added to Visual Studio .Net 2005 (VS). Users record digital ink annotation on a code file opened in VS, and also use all the functions provided by VS. However, it doesn't provide navigation of annotations to support users finding and moving between annotations.

A program document is normally very long and split among many files. Navigation support of annotations can provide users with an outline of existing annotations and the information of how many and where annotations exist. Also it assists users to easily locate a specific annotation. Among the current available annotation tools, there are no tools that provide navigation support.

3. DESIGN
Digital ink annotating requires support for three major functions: the annotations must be free-form and modifiable, the annotations can be reflowed when the underlying text is changed, and a navigation support is provided to help users to overview, select and find annotations.

3.1 Requirements
The PDE must be extensible to allow developers to extend its functionality and add in other functions. In order to allow the PDE to perform normally while supporting digital ink annotation, CodeAnnotator must integrate into the PDE seamlessly.

When a user first selects a code file to annotate, a new annotation file will be created and stored as a part of the development project. Once the user finishes the annotation, the annotation file needs to be saved to allow later review. The annotation file will be loaded when the user wants to review or annotate the selected code file.

3.2 Free-form and modifiable annotation

Free-form annotation consists of words, symbols and text selection marks, and allows annotations to be made anywhere without limitation on shape or content [14]. This is achieved by overlaying the code window with a transparent canvas that holds ink annotations and associating annotations to the underlying code by making the code window and the transparent canvas scroll together. This approach allows users to annotate anywhere inside the code window; there is no restriction of a pagination boundary in the code window as there is in document formats such as PDF that set artificial restraints on the users' positioning of the annotations

When people annotate over paper documents with pens, it is hard to modify existing annotations. Modifying real ink annotations usually involves concealing or crossing out the annotations then rewriting them; this looks messy. A digital ink annotation tool provides users with functions to efficiently and cleanly erase, select, move and recolor a selected annotation.

3.3 Reflowing annotations

One of the most important features of digital ink annotation tools is to be able to deal with dynamic digital documents. As annotation is most often a function of some type of review it must be expected that the text inside the documents will be changed. This requires the existing annotations be reflowed to remain consistent with the underlying code when the code file is modified. In order to reflow annotations, ink strokes have to be grouped. Annotations belonging to the same underlying context must be grouped together by taking into consideration location and temporal properties of the ink strokes that make up the annotation [1, 16]. Then when underlying code moves up/down, any attached annotation group has to move too. And when underlying code is deleted, its attached annotation must either be deleted or archived.

A code file is arranged line by line. Each code line cannot be rearranged like a sentence and typically lines are not wrapped with resizing the code window. Therefore, it is unnecessary to consider the issue of reflowing annotations horizontally. We only need to focus on moving annotations up or down inside the code window.

3.4 Navigation support

A project contains many code files. A code file is structured by its classes and procedures. PDEs often offer a navigation system to assist programmers to know which code files exist in the project and how classes are related in a code file, and to locate a specific class.

For digital ink annotation tools, it is useful to provide a similar navigation system to users. A code file often extends over many lines and cannot be fully displayed in one screen. We have to scroll the screen up/down to review existing annotations, which is very inconvenient. Through the navigation system, users can easily locate a specific annotation inside the code window.

4. IMPLEMENTATION

CodeAnnotator is developed inside Eclipse as a set of plug-ins. Because Eclipse supports Java language it is a popular open source PDE that used extensively in teaching departments and industry.

CodeAnnotator can be used on tablet PCs and desktop computers with a tablet USB input device. It allows users to directly annotate onto the output screen. This tool is implemented in Java using both the Java API and Eclipse framework.

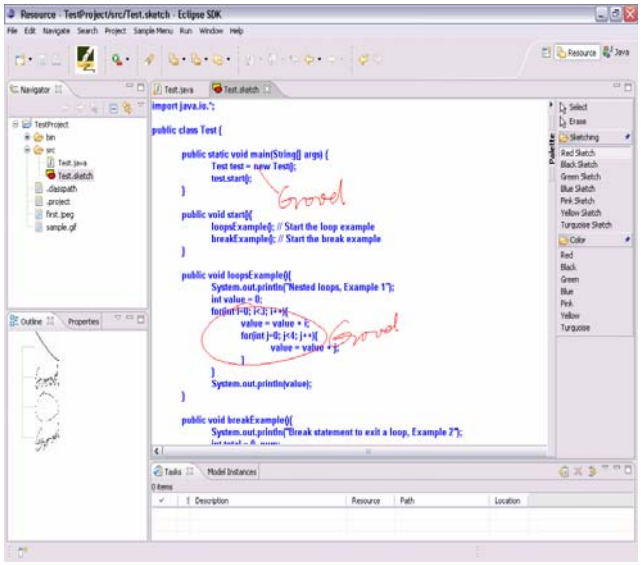


Fig. 1. CodeAnnotator in Eclipse

Figure 1 shows an annotation window in Eclipse. When a user loads a code file and selects annotation model from the Eclipse toolbar, a new annotation window is created based on the current code window. An Outline window is created. The flyout-Palette at the right side of the annotation window contains the annotation tools.

The major features of CodeAnnotator are: attaching annotations to a specific code line, grouping annotations and editing them, reflowing annotations, and navigating annotations.

4.1 Attaching Annotations

Ink strokes are used as linkers to link a group of annotations to a specific code line [12]. Linkers can only be a line or a circle. Each is treated differently so first we have to recognize the linkers.

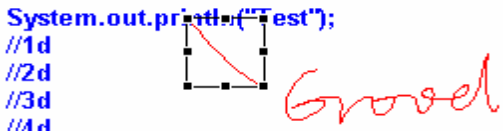


Fig. 2. Line linker

A feature of a straight line is that its start point and last point touch the left and right board of the bounding box (See Figure 2). And this can be discriminated from a circle by measuring whether the distance between the first point and the last point (ab) is less than the hypotenuses of 25% of the bounding box's width and height (a'b') (See Figure 3).

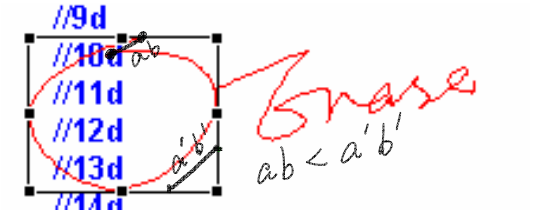


Fig. 3. Circle linker

If the linker is a line, then its corresponding annotation group is linked to the code line closest to the start point of the linker. Otherwise, for a circle the corresponding annotation group is attached to the line closest to the middle point of the bounding box.

4.2 Group and edit annotations

We group annotations based on the spatial and temporal properties of the annotations [1, 16]. After creating a linker, if the subsequent annotation is created in less than two seconds, then it belongs to this linker. If the following annotation is created in less than two seconds, then it also belongs to this linker. Otherwise, we use the spatial property of the annotation to decide which group it belongs to.

Spatially adjacent annotations are considered to be in the same group. We calculate the position of the new annotation; say at the coordinate (x1, y1), and its height is h1, its width is w1. According to this calculated position, we find a closest annotation group and then this annotation belongs to this group. For example, the first group is located at the coordinate (x2, y2), and its height is h2, its width is w2. The second group is located at the coordinate (x3, y3), and its height is h3, its width is w3.

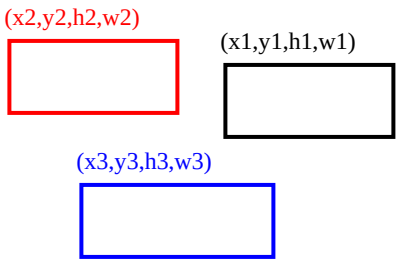


Fig. 4. Group annotations

- If the range from y1 to y1+h1 partially overlaps with the range from y2 to y2+h2, then the new annotation belongs to the first group.
- If the range from y1 to y1+h1 partially overlaps with the range from y3 to y3+h3, then the new annotation belongs to the second group.
- If the range from y1 to y1+h1 overlaps with both the range from y2 to y2+h2 and the range from y3 to y3+h3, then we have to decide which is closest:
  - If x1 is bigger than x2 and x3, then if x1 is closer to x2+w2 than to x3+w3, then the new annotation belongs to the first group, otherwise it belongs to the second group.
  - If x1 is less than x2 and x3, then if x1 is closer to x2 than to x3, the new annotation belongs to the first group, otherwise it belongs to the second.
  - If x1 is bigger than x2 and is less than x3, then x1 belongs to the first group because we are more likely to append comments to the existing annotation.
  - If x1 is bigger than x3 and is less than x2, then x1 belongs to the second group
- Otherwise, we consider the new annotation to be a new group.

Editing annotations is a very important function in annotation tools. We support moving a specific annotation group around by selecting it. Also an annotation can be deleted or erased by selecting it with the eraser tool. Annotations can be written in different colors by selecting a color before writing. Moreover, we can recolor an existing annotation by selecting it and then selecting a different color from the color tool.

4.3 Reflow annotations

Reflowing existing annotations is necessary to maintain consistency with the underlying code. We only need to consider the vertical reflow as code is line-based. There are three situations we need to handle, as explained by Priest [12]. First, when several code lines are added ahead of a code line with annotations, this code line and its associated annotation group moves down together. Second, when several code lines are deleted ahead of a code line with annotations, this code line and its corresponding annotation group moves up together. Third, when a code line with annotations is deleted, its associated annotation group is also deleted.

4.4 Navigation support

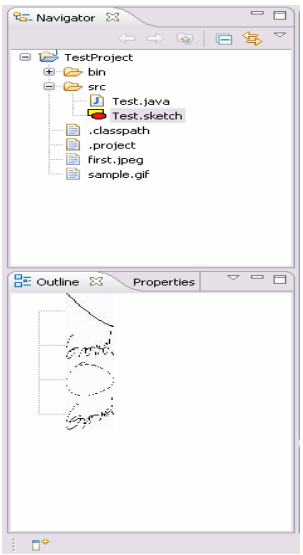


Fig. 5. Navigation support

The navigation system includes a Navigator window and an Outline window. The Navigator window tells users which code files were annotated. The Outline window guides users to locate the position of a specific annotation inside the annotation window, and provides a graphical outline of existing annotations. Users can directly open an annotation file in the Navigator window. After this file is opened, an Outline window is automatically opened and shows a graphical outline of existing annotations in this file. When users make an annotation in the annotation window, a copy of this annotation and its corresponding information such as location will be put inside the Outline window (See Figure 4). When users select a specific annotation inside the Outline window, the annotation window will scroll to the page containing this annotation.

5. CONCLUSTION and FUTURE WORK

This paper describes a code digital ink annotation tool, CodeAnnotator. The objective of this tool is to provide users with an electronic environment to annotate in Eclipse. This tool lets users enjoy both digital ink annotation and Eclipse support. In other words, after loading a code file in Eclipse, users can write annotations with a digital pen and can also access all the functions provided by Eclipse, such as debugging, compiling, and executing.

Including handwriting recognition is a challenge because the writing and iconic annotation must first be separated. We are yet to extend CodeAnnotator to recognize handwriting. When this is completed the next steps are to conduct usability testing and evaluation studies to assess the efficacy of annotation within a PDE.

Another interesting extension would be functionality to consolidate digital ink annotations made by different users into one file to support collaborative code review. Then the separate digital ink annotations can be merged automatically, and some mechanism would differentiate each person's annotations, particularly when they are in the same place. Furthermore, we can extend it to support assignment marking. This would allow markers to directly mark program assignments in Eclipse.

6. REFERENCES

[1] Bargeron, D. and Moscovish, T. Reflowing digital ink annotations. *CIII 2003, April 5-10*, ACM Press (2003),385-392

[2] Bottoni, P., Levialdi, S., Labella, A., Panizzi, E., Trinchese, R. and Gigli, L. MADCOW: a visual interface for annotating web pages. *Proc. of the working conference on Advanced visual interfaces AVI '06*, ACM Press (2006), 314-317

[3] Francik, E. Rapid, Integrated design of a multimedia communication system. *HumanComputer Interface Design (1995)*, 36-69

[4] Golovchinsky, G. and Denoue, L. Moving Markup: Repositioning freeform annotations. *Proc. UIST 2002*, ACM Press (2002), 21-30

[5] Marshall, C. C. Annotation: from paper books to the digital library. *Proc DL 1997*, ACM Press (1997), 131-140

[6] Marshall, C. C. Toward an ecology of hypertext annotation. *Proc. HyperText 1998*, ACM Press (1998), 40-48

[7] Microsoft Word. http://www.microsoft.com/office/word/using.htm

[8] Mock, K. Teaching with Tablet PCs. *Proc. Journal of Computing Sciences in Colledges 20, 2 (2004)* 17-27

[9] Moran , T. P., C, P. and Van Melle, W. Pen-based interaction techniques for organizing material on an electronic whiteboard. *In Symposium on User Interface Software and Technology (1997*), 45-54

[10] Myers, D., Hargreaves, E., Ryall, J., Thompson, S., Burgess, M., German, D. and Storey, M. Developing marking support within Eclipse. *Proc. Of OOPSLA 2004, ACM Press (2004)*, 62-66

[11] Plimmer, B. and Mason, P. A Pen-based paperless environment for annotating and marking student assignments. *Proc. 7th Australasian User Interface Conference*, CRPIT press (2006), 37-44

[12] Priest, R. and Plimmer, B. RCA: Experiences with an IDE annotation tool. *Proc. 6th ACM SIGCHI New Zealand chapters international conference 2006*, ACM Press (2006), 53-60

[13] Ramachandran, S. and Kashi, R. An architecture for ink annotations on web documents. *Proc. 17th International Conference on Document Analysis and Recognition*, IEEE Computer Society (2003), 256-260

[14] Schilit, B. N., Golovchinsky, G. and Price, M.N. Beyond Paper: Supporting active reading with free form digital ink annotations. *Proc. CHI 98*, lOS Angeles, CA, ACM Press (1998), 249-256

[15] Schumacher, G. M. and Nash, G. J. Conceptualizing and Measuring knowledge change due to writing. *Research in the teaching of English, vol 25*, pp. 67-96, 1991

[16] Shilman, M. and Viola, P. Spatial recognition and grouping of text and graphics. *EUROGRAPHICS Workshop on Sketch-Based Interfaces and Modeling 2004*. Eurographics digital library

[17] Takahiro, K., Tashiro, N., Ozono, T., Ito, T., and Shintani, T. Web Page Marker: a web browsing support system based on marking and anchoring. *WWW 2005, May 10-14*. Chiba. Japan. ACM, 1012-1013

[18] Wolf, J. L. Effects of annotations on student readers and writers. *Digital Libraries, San Antonio*, TX, ACM Press (2000) 9-26

INTELLIGENT MIND-MAPPING

Vincent Chik, Beryl Plimmer

ABSTRACT

Current computer based mind-mapping tools are much slower to use than pen and paper because users are distracted by tool operations such as finding and arranging widgets. The shift in focus from brainstorming to tool management interrupts the rapid brainstorming process that mind-maps are intended to support. Our pen based mind-mapping software that includes intelligent ink recognition, editing and export alleviates these intrusions as the user only has to worry about writing on the canvas, yet usual digital document support is provided. The digital ink recognition and manipulation techniques described here will be of interest to others working with informal documents. .

1. INTRODUCTION

A mind-map is a sketchily structured visual representation of one's thoughts which may lead to a train of related ideas. It is based on radiant thinking, a concept which describes how the human brain processes ideas and information, whereby different ideas are associated to each other through relationship hooks [5]. The four main features of a mind-map (figure 1) are as follows:

- Each mind-map has a starting location, the center node that contains the central theme or idea.
- The ideas of the mind-map "radiate" from the central node as branches with sub-nodes connected to each other in parent-child relationships.
- The final structure of the mind-map becomes a hierarchy of linked nodes.
- Each connector/branch has keywords or an image associated with it.

Mind-maps are traditionally hand drawn and used for critical thinking tasks such as strategic planning. They are an effective way of rapidly jotting down and arranging information, affording reinforced association of ideas and recall.

Compared with normal note taking or brain-storming, mind-maps have several advantages. For instance, time is saved by just noting down relevant key words. Associations between key points are highlighted while passively creating a hierarchy of ideas. Reviewing a mind-map takes considerably less time than to overview a set of written notes as the mind-map is effective in displaying the relevant keywords associated with a particular topic. By providing a visually stimulating environment, the retention of information by the brain is made easier.

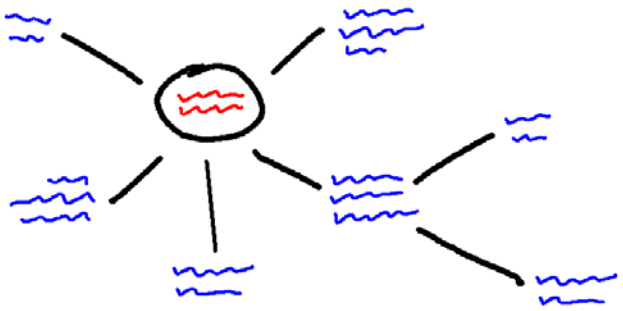


Figure 1. Basic mind-map structure of: a centre node and branches that connect nodes together in a hierarchy.

There are a number of computer applications for mind-mapping. However these are widget-based tools that require the user to select an appropriate node widget or connector before they can enter data. We hypothesize that these tools will adversely affect the idea generation process in the same way as widget-based design tools have been shown to adversely affect the design process [8]. In contrast, this adverse affect of the computer is minimal with sketch-based computer tools [3, 13].

In order to explore the validity of our hypothesis we must first build a computer-based mind-mapping tool that more closely matches traditional pen and paper environments. Sketch-based computer design tools must also provide usual computer editing and archiving support. Here we report on the design and development of our pen-based mind-mapping software.

2. RELATED WORK

As a background to this project we have reviewed the functionality available in current widget-based mind-mapping tools and compared that with the functionality provided in sketch-based tools designed for other tasks. From these we have formulated a list of the technical challenges that this project must address and related research on those topics.

2.1. Mind-mapping Tools

Mind Manager 6 [1] (figure 2) is typical of current mind-mapping tools. The user clicks on the canvas to create a new node. Text can be added to a pre-selected node via the keyboard. Connections are made by dragging nodes on top of other nodes. Once the connection is made the layout is

automatically imposed by the software. For example the distance between nodes is restricted and prevents the user from utilizing all the canvas space. A useful function of mind-mapping tools is support for moving a branch and its associated sub-branches to a different location. The advantage of an electronic environment for creating mind-maps over paper is the support for editing, exporting and archiving.

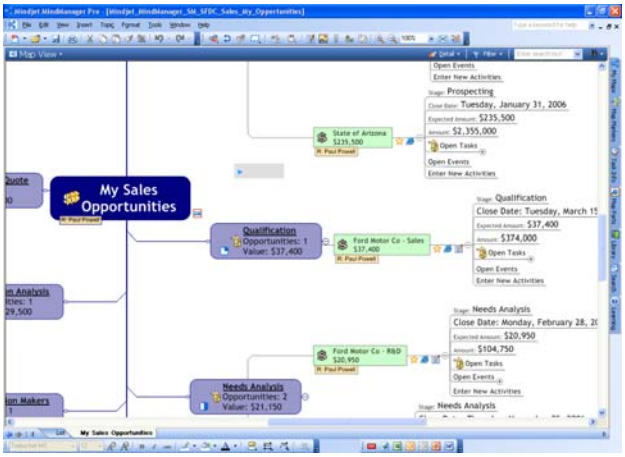


Figure 2. An example of a mind-mapping tool is the Mind Manager 6 Pro.

Although these computer based mind-mapping tools possess many capabilities, much time is consumed with dragging and dropping nodes and connectors from the toolbar before the user can start filling in the nodes with information and form a comprehendible diagram. Research from design domains [3, 8, 13] suggests that users become distracted with arranging the interface as opposed to concentrating on the problem solving process.

2.2. Sketch Tools

Rough design is often performed with pen and paper. The advantage of using such a medium is the unconstrained drawing space and low cognitive load [8]. There is no restriction as to the layout of the document. Studies comparing pen and paper with widget-based computer tools and sketch-based computer tools have found that using ink is preferable to widgets for design tasks such as UI design [13], multi-media design and graphic design [3]. There are two attributes of sketches that are believed to be important in early design. First, the creation of documents with ink is quicker and using the pen requires less cognitive effort than a widget based design tool. Second, the hand-drawn appearance of the ink implies incompleteness, which in-turn suggests that the document can and should be reviewed and changed [12]. While mind-mapping is a problem solving technique rather than a design technique, there is significant similarity between the two tasks that suggest that sketch tools may be more useful than widget based tools.
A pen-based computer tool that recognizes the user's intention intelligently, has the advantages of both computer technology (for editing and so on) and the simplicity of pen & paper to alleviate distractions.

2.3. Challenges

The technical challenges that this project poses consist of three parts: ink recognition and grouping; structural analysis; and ink reflow.

- Ink recognition is vital to discerning the elements of a mind-map. It is a precursor to understanding the structure of the mind-map by correctly identifying text from drawings. As a part of the recognition, individual ink grouping is required to group related ink strokes into nodes.
- The structure of the mind-map can be established once the nodes and drawing elements have been identified. This structure must be known to support intelligent editing and exporting into other formats.
- Intelligent editing includes the ability to move a branch and its sub-branches to another position on the mind-map. Ink reflow is essential when users make adjustments to the mind-map, maintaining the look of the mind-map by rearranging the Ink into a more suitable shape or position.

Existing research suggests techniques we can adopt and modify for ink recognition algorithms [11, 14], grouping [6, 10], structure [7] and ink reflow [2, 4, 9].

3. OUR APPROACH

We have developed a mind-mapping tool for the Tablet PC that is able to recognise text and annotations that are made by the user and treat them as objects as opposed to just a visual representation (figure 3). This allows the mind-mapping tool to generate an internal logical representation of the user's mind-map with the intent of enabling the user to revise its structure.

In addition, in order to allow user flexibility, the system adapts as the user deletes, moves or creates additional data on the existing map. For example, when a user selects a branch of annotations and moves them to another section of the mind-map, the system intelligently reflows the ink to reflect the new orientation and rearranges adjacent branches as necessary to create space for the newly transplanted branch.

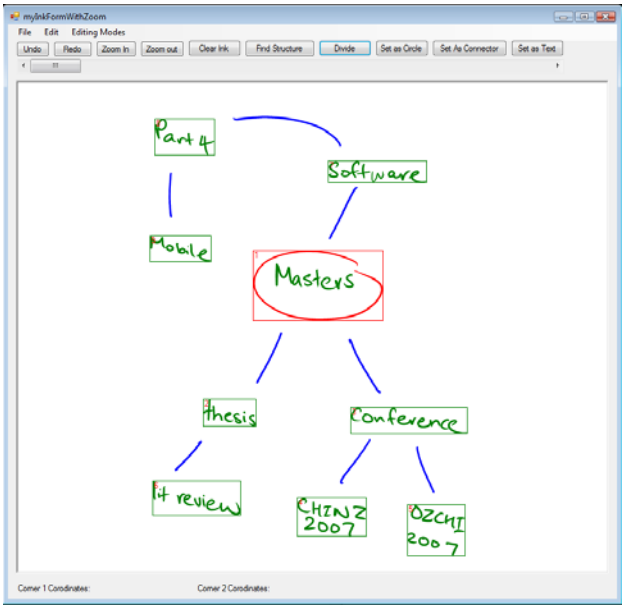


Figure 3. Screenshot of Current Prototype.

3.1. Features

The following are features that were deemed necessary for this mind-mapping application.

Table 1. Mind-mapping Features

Ink recognition	The system must be able to correctly recognize the ink strokes the user writes.
Eager Recognition	The system should be able to recognize the text or drawing as the user writes so that editing is continuously supported
Structure Analysis	The system must be able to determine the hierarchical structure of the mind-map
Editing (Undo / Redo)	The system should support the user to add, or delete ink. Also, if recognition errors occur, the system should allow the user to manually change the type of component an ink/group of ink represents.
Move + Reflow	The user must be able to select a connector and be able to move it to another node
Branch Collision Detection	The system should intelligently detect if ink objects overlaps other objects when moved
Auto Colour Wheel Segmented	Each connector that branches off the central node can be colour coded relative to its position to the central node
Load & Save	The system should be able to save and reload an existing mind-map
Export Capability	The system should allow the user to export the mind-map into another format

3.2. System Architecture

The following is a flow diagram of the back end of the system. Each ink stroke drawn by the user is first parsed by the ink divider.

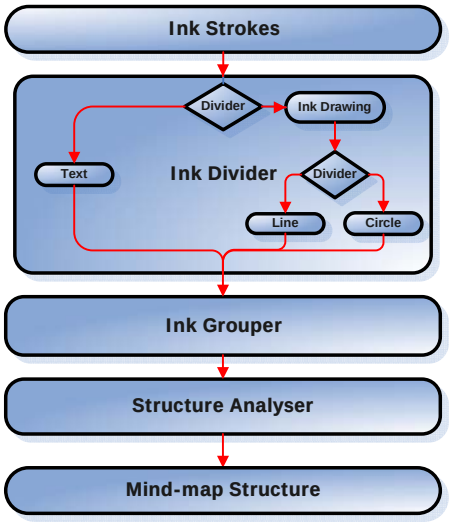


Figure 4. Mind-mapping Tool system architecture.

The ink is divided into 3 categories, text, circle or line. Upon stroke division, the ink is then passed into the ink grouper which takes strokes that are geographically close to each other and groups them together as a node. After grouping, a hierarchical order of the nodes is established. As new ink strokes are added the software may alter the categorization of previously entered strokes.

3.3. Ink Divider

The Ink divider is based on the divider from [11] to separate text and drawing ink. The divider looks at specific features of each ink stroke and categorizes them using a binary tree structure. We replaced the bounding box width feature specified in [11] with the ink bounding box diagonal as mind-maps differ from the diagrams used in their training set, with frequent use of sloping lines for connectors. Once the ink stroke is classified as text stroke or drawing, text is passed to the OS recognizer. The drawing strokes are then separated again into lines or circles by a filter [14]. For a circle, the start and end points of the stroke are generally in close proximity as opposed to a line. The bounding box height and width are again somewhat similar for a circle while a line stroke may not

be. The filter consists of two checks; the distance between the start and end points of the stroke is checked to see if it is below 1200 hi metric units and that the ratio of the longest bounding box side and the distance between the start and end points of the stroke is less than 0.5. This ensures ellipses are also categorized as a circle. The recognized writing and drawing strokes are then piped to the ink grouper.

3.4. Ink Stroke Grouping

The ink grouper takes the recognized strokes and groups them into coherent entities of node or connector. Ink classified as lines and circles are automatically categorized as connectors and nodes respectively. All strokes that reside in a circle are automatically grouped together and are put into the same node as the circle. Remaining text strokes must be grouped together into words and groups of words to form uncontained nodes. Text strokes are grouped by examining the top, bottom and sides of each stroke's bounding box. If the bounding boxes top values are within 200 hi metric units of each other and the sides are within 500 hi metric units, the strokes are grouped together. We have arrived at these values with informal testing and will check their validity as a part of our usability testing.

3.5. Mind-map Structure

The mind-map is sorted hierarchically. The first node that is drawn is typically the central node and all other nodes connected to it via connectors are its sub nodes. On this basis, the system implements a recursive algorithm, starting by finding the sub nodes of the central node. This is carried out by finding the branches linked to the main node. The first branch is determined by the first connector that is drawn and its subsequent branches are found until there is none left before coursing down the next branch linked with the main node. With recursion down the tree, the complete structure of the mind-map can be found. With this structure, the mind-map can be easily exported into ordered digital formats. At times people create mind-maps with cross connections. This causes our recursive algorithm to loop endlessly. Hence a checklist was implemented that records the nodes that the system has traversed and skips the nodes already analyzed.

3.6. Editing and Reflow

To support the relocation of a branch to another part of the mind-map basic reflow has been implemented. The relocated nodes repel other nodes away should they overlap. The area around the relocated node's bounding box is divided into 8 sectors. A starburst metaphor is used to move existing nodes. The relative positions of the centre point the relocated and existing nodes determines the movement direction. The distance nodes are pushed back is calculated by the overlap amount. The connectors that link the nodes are then elongated or compressed. The amount of compression and elongation is found by the distance and direction the node was shifted and the connector transformed and rotated accordingly.

4. CONCLUSIONS

Sketch-based mind-mapping tools offer an alternative to paper or widget-based tools. Our first prototype presented here concentrates on text nodes to reduce recognition problems. It can formulate a hierarchy of nodes and supports basic restructuring. We have conducted some informal evaluations and participants have commented favourably on the similarity of the tool to mind-mapping on paper. Some recognition errors occurred, which annoyed them, but they were satisfied with manual correction functionality.

Upon completing the initial system, a formal usability study will be undertaken and a second prototype developed. The mind-mapping tool can then be compared with other computer/paper oriented mind-mapping techniques to test our hypothesis on the affect of tools on mind-mapping.

5. ACKNOWLEDGMENTS

We would like to thank Microsoft Research Asia for their support of this project.

6. REFERENCES

[1] Mindjet: Software for Visualizing and Using Information, Retrieved on 27th June 2007 from: http://www.mindjet.com

[2] Arvo, J., Novins, K., Appearance-preserving manipulation of hand-drawn graphs, Proceedings of the 3rd international conference on Computer graphics and interactive techniques in Australasia and South East Asia, ACM Press, (2005),

[3] Bailey, B. P., Konstan, J. A., Are Informal Tools Better? Comparing DEMAIS, Pencil and Paper, and Authorware for Early Multimedia Design, in proc CHI 2003, ACM, (2003), 313-320

[4] Bargeron, D., Moscovich, T., Reflowing digital ink annotations, Proceedings of the SIGCHI conference on Human factors in computing systems, ACM Press, (2003),

[5] Buzan, T., Buzan, B., The Mind Map book, BBC Worldwide Limited, (2000), 269

[6] Chung, R., Mirica, P., Plimmer, B., InkKit: a generic design tool for the tablet PC, Proceedings of the 6th ACM SIGCHI New Zealand chapter's international

conference on Computer-human interaction: making CHI natural, ACM Press, (2005),
[7] Freeman, I. J., Plimmer, B., Connector semantics for sketched diagram recognition, Proceedings of the eight Australasian conference on User interface - Volume 64, Australian Computer Society, Inc., (2007),
[8] Goel, V., Sketches of thought, The MIT Press, (1995),
[9] Golovchinsky, G., Denoue, L., Moving markup: repositioning freeform annotations, Proceedings of the 15th annual ACM symposium on User interface software and technology, ACM Press, (2002),
[10] Moran, T. P., Melle, W. v., Chiu, P., Spatial interpretation of domain objects integrated into a freeform electronic whiteboard, Proceedings of the 11th annual ACM symposium on User interface software and technology, ACM Press, (1998),
[11] Patel, R., Plimmer, B., Grundy, J., Ihaka, R., Ink Features for Diagram Recognition, Sketch Based Interfaces and Modeling IEEE, (2007),
[12] Plimmer, B., Apperley, M., Evaluating a sketch environment for novice programmers, CHI '03 extended abstracts on Human factors in computing systems, ACM Press, (2003),
[13] Plimmer, B. E., Apperley, M., Software for Students to Sketch Interface Designs, in proc Interact, (2003), 73-80
[14] Priest, R., Plimmer, B., RCA: experiences with an IDE annotation tool, Proceedings of the 6th ACM SIGCHI New Zealand chapter's international conference on Computer-human interaction: design centered HCI, ACM Press, (2006),

MULTI-HEURISTIC EFFICIENT SEARCH

Santiago Franco

ABSTRACT

This research is on Artificial Intelligence, planning. We are interested on how to most efficiently solve a planning problem given a list of admissible heuristics and a problem description including an initial state and a goal.

Our list of available heuristics is derived from pattern databases abstractions. This heuristic family has been the focus of a lot of research recently due to their very efficient reduction of the search space. However, our research should apply to any group of admissible heuristics.

Heuristic performance is a function of the heuristic accuracy, evaluation time and the solution length. The problem optimal distance (OD) is unknown until solved.

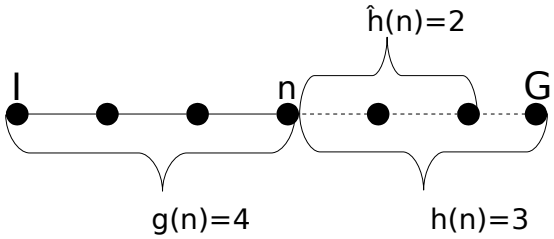
We use Iterative Deepening A* (IDA*) as our search algorithm. IDA* iterative nature allows us to use the information collected from previous iterations to make an informed decision on how to most efficiently use available heuristics on the next IDA* iteration.

1. INTRODUCTION

1.1. Heuristics

Heuristics are created to reduce the time it takes to solve problems with large search spaces. Admissible heuristics are used to perform informed searches(A*,IDA*,etc.) to find optimal solutions to problems.

Fig. 1. Informed Search Diagram



n is the current state.
g(n) is the OD from I to n.
h(n) is the OD from n to G.
$\hat{h}$ is the heuristic estimate OD from n to G.
F(n)=g(n)+h(n)=OD.

Each dot represents a problem state in the optimal path from initial state I to goal state G. Informed search on planning expands all nodes from I to G whose $\hat{F}(n) \leq OD$. If the node is not expanded it is called culled. When no heuristic is used all the nodes whose distance to G is less or equal to the OD would be expanded. This tree is called Brute Force Search Tree (BFST).

$\hat{h}$ is admissible iff: $\forall n \rightarrow \hat{h}(n) \leq h(n)$ Eq. 1. Admissible heuristics are lower bounds on the optimal distance.

$\hat{h}$ is consistent for any choice of problem states n, m iff:

$\forall n, m \rightarrow \hat{h}(n) \leq distance_{min}(n, m) + \hat{h}(m)$ Eq 2.

Informed search algorithms which use admissible heuristics are guaranteed to find an optimal solution (eventually). Consistency guarantees that when node n is expanded it has already found an optimal path to n. All admissible heuristics can be made consistent.

The text book standard for characterizing the effect of admissible,consistent heuristics on search performance is to model the search for an optimal solution as the expansion of a Heuristic Search Tree(HST), from initial state to goal state, on which each node represents a list of successive actions taken from the initial state. HST is a sub-tree of the BFST and thus smaller [1][2]. The quality of the heuristic is defined by how small the HST is.

1.2. IDA*

Among the informed search algorithms IDA* is of particular interest to us. IDA* is a iterative, depth-bounded version of A*. It is a depth-first search algorithm which culls the search path and backtracks when the cost $\hat{F}(n)$ of a node n on the path exceeds a $\hat{F}$ bound C for that iteration. The initial $\hat{F}$ bound C_o is the initial state heuristic value, and is increased in each new iteration to the lowest cost of all the nodes culled on the previous iteration, until a goal node is

expanded. Each successive IDA* iteration expands all the previously expanded nodes plus it expands the previously culled nodes to the new $\hat{F}$ bound. IDA* guarantees an optimal solution if the heuristic function is admissible.[3].
The main difference between IDA* and other informed search algorithms is IDA*'s iterative nature, which is depth bounded.

Each iteration of IDA* generates a depth bounded HST which is a sub-tree of the next IDA* iteration's HST. The effect of each successive iteration is to raise the depth-bound, adding nodes to the HST's until the final iteration. Earlier IDA* iterations can be used to predict the performance of the different heuristic combinations on the next IDA* iteration till the solution is found.

By using IDA* it is not necessary to know the problem optimal distance a priori to choose which combination of heuristics will solve the problem in the smallest possible time.

1.3. PATTERN DATABASES HEURISTICS

Of recent interest to AI planning are automation of pattern database heuristics [4]. Pattern Databases divide the search space into smaller sets (for the N-puzzle these are tile groups) and then build databases solving all instances of the simpler abstracted space. If only the moves concerning the tiles belonging to a set are counted then we can build a group of disjoint databases sets whose admissible solution lengths can be added together forming a new admissible heuristic[5].

Pattern databases work quite well at reducing the search space but their accuracy depends on the unaccounted interactions between elements in different sets.

There is no selection rules for how to best divide the search space as unaccounted interactions between the groups depend on the current problem state. Korf recommends to have as few sets as possible in order to maximize heuristic accuracy but with the constraint of not to keep so many elements on each set that the evaluation time makes it too costly (in evaluation time and memory space) to store a lookup database of all heuristic values[5].

2. PROBLEM DESCRIPTION

Our objective is to minimize the time it takes to solve a planning problem given a list of pattern database heuristics, a problem description including an initial state and a goal and IDA* as a search method.

The time it takes to solve a planning problem depends on several factors but the most important ones are the search method, the accuracy of the heuristics used and their evaluation time.

Combining heuristics can increase accuracy but at the cost of increasing heuristic evaluation time per search node.

Each combination of heuristics is treated as a new heuristic. Unless the component heuristics are known to be completely independent, the only way to guarantee the optimality of combined heuristics is to take the maximum value of the individual heuristics on each search node as the combined heuristic value for that node.

We are interested in the case of how to best combine heuristics which are both more accurate and costly. This is the case for the pattern databases family of heuristics. The smaller the sets the faster is the heuristic evaluation time but the less accurate is the heuristic.

So the problem is: if we have a group of heuristics ordered by decreasing accuracy which is also increasing computational cost, how do we combine them to minimize total search time?

Our approach is to build a formula which will predict the time cost associated to any heuristic combination. We are in the process of creating an automated problem solver which utilizes this formula to minimize the time it takes to solve a given problem instance.

3. COST FORMULA

$$t_{iteration} = N_{\exp} * (t_{prunning\,test} + t_{goaleval} + t_{heuristic\,eval} + t_{node\,expansion}) + N_{culled} * (t_{prunning} + t_{heuristic\,eval}) + N_{prunned} * t_{prunning}$$

The formula above describes the total time it takes to finish an IDA* iteration using a heuristic combination. None of the time parameters are constant or even linear except perhaps $t_{goal\ eval}$. Analysis follows:

N_{exp} is the number of nodes expanded on the search. Every node expanded has been tested for whether it should be expanded.

This requires to test whether is the goal node, whether the state is pruned and finally there is a time cost associated to expanding the node.

Nodes culled are those nodes whose expansion is delayed till the next iteration because their $\hat{F}$ value is less than the current depth bound.

Nodes pruned will never be expanded because of search control rules like cyclic paths, inverse operator check, duplicate check, etc.

All time costs can be assumed to be constant except:

$t_{heurtisitc\ eval}$ depends on the cost of the heuristic. For closed formulas like Manhattan the cost is constant. For abstraction based heuristics it is is a function of how complex the abstracted space is. For pattern database heuristics with large groups the cost can be quite high.

$t_{nodeexpansion}$ depends on the search method. If the search method stores visited states then as soon as the visited states database is bigger than available RAM the node expansion time cost sky-rockets. We should be able to use our the predicted size formulas developed in this research to predict this event. Normally it is a constant cost.

$t_{prunning\ test}$ is constant as long as an efficient hashing function can be built. Otherwise, the cost is linearly dependent of the size of the pruning database to check against. Normally a hashing function can be build for any State of a domain.

4 HEURISTIC COMBINATION EFFECT ON COST FORMULA

Our aim is to combine different pattern database heuristics so we can solve a PDDL problem in the shorter time possible.

Combined heuristics search space savings is the intersection of all the individual heuristic savings as some savings might overlap. Combined heuristic time cost is not the addition of all the individual heuristic evaluation time costs.

The way we combine individual heuristics is to evaluate them on a predetermined order until the current search node is either culled or expanded. For node culling it is sufficient for one of the individual heuristics to cull the node. However, it is necessary to evaluate all individual heuristics on the current node to expand it.

$$t_{comb\ heuristic\ evaluation,\ node\ expanded} = \sum_{i=1}^{i=n} t_i$$

$$t_{comb\ heuristic\ culling} = t_1 + P(\neg h_1)t_2 + P(\neg h_2|\neg h_1)t_3 + \ldots + P(\neg h_{n-1}|\neg h_1 \cup \neg h_{2.} \ldots \cup \neg h_{n-2}) * t_n$$

As most of the nodes in any HST are culled nodes, we need to estimate the average time cost for culling a node when combining heuristics.

5 A NEW WAY TO COMBINE PATTERN DATABASES HEURISTICS

The current approach is to try different sets until the best overall pattern database is found for the domain. So no heuristics are combined, the best one overall is used.

In this research we are trying a new way to minimize the search time by combining different heuristics at different levels.

Korf found out that if he divided the n puzzle for n=15 in 2 halves he got the best time results. But as he increased n the search space exponentially increased until the heuristics search space was computationally too expensive. The workaround was to divide the search space in more sets so the abstracted search space was manageable .

The accuracy of the pattern databases suffered and eventually heuristics like Manhattan did better.

Our idea is to have a dynamic list of possible candidate heuristics. Each of this heuristics is a pattern database which divide the search space in different sets sizes,i.e. dividing the search space in 2, 3,... groups. We would apply them sequentially from cheaper to more expensive. The order is justified by the fact that when culling it is sufficient for the cheaper heuristic to cull the node. As the cost of each heuristic grows exponentially with the set sizes it is a good idea to start with the cheaper heuristic.

The list of candidate heuristics is dynamic because some of the more expensive heuristics would only be available to the lower levels of the search. To explain why consider the case on which a more expensive heuristic h_2 culls a node which cheaper heuristic h_1 does not. The time savings would be:

$$time\ savings(h_{2,}h_1) = P(h_2|(\neg h_1)) * (N_{h1\ expanded} * t_1) - t_{evaluation\ h_2}$$

Time savings is linearly dependent on how many nodes h_1 would expand. So we can afford to use more expensive heuristics if enough nodes are culled.

Our time savings formula is the number of nodes saved from expanding by h_1 times their total expansion time cost minus the time cost of heuristic h_2. Time savings of h_1 is weighted by the probability of h_2 culling the node while h_1 does not. h_2 culling a node which h_1 does not is not a guarantee. The evaluation time of h_2: will be added to the time cost regardless of whether the node is culled or not. So assuming we know the culling probabilities of the heuristics, a decision still has to be made on whether is it worth it to try a more expensive heuristic which could cull the node.

This decision depends on the potential nodes saved, so as the search depth gets closer to the search frontier the potential nodes saved decreases. At some threshold depths it will not be worth even trying some of the available expensive heuristics as the potential savings do not justify it. This threshold can be adjusted empirically.

We need to estimate how many nodes could potentially be saved from expansion. As previously said any part of the HST is a subtree of the BFST. So the maximum size savings would be if h_1 would expand all nodes to the depth bound as the BFST would. This is not going to be the case if h_1 is any good.

Most of our previous research has been on how to predict the sizes of HST[6] . The traditional view is to model both the BFST and the HST as uniform search trees with a reduced Effective Branching Factor(EBF) or more recently reduced Effective Depth(ED). Our research indicates that both models are valid on the limit of large depth.

$$N_T = \frac{EBF^{ED+1} - 1}{EBF - 1}; Uniform\,Search\,Formula$$

But when looking at trees whose depth is not very large we discovered that the initial state and the heuristic used changes the way the EBF and ED converge to their asymptotic values. We have different methods to predict this convergence. The two most successful ones are:

1. modelling the HST EBF and ED as a system of 2 non linear equations which we can solve after two iterations.
2. Calculating the HST EBF as an arithmetic mean of nodes created vs fertile nodes and then solving for ED on the uniform Search formula.

We use the previous iteration of IDA* to acquire statistics to estimate the conditional culling probabilities and also the EBF and ED for each heuristic needed to estimate potential savings.

6 CONCLUSIONS

It is not trivial how to combine more informed heuristics but computationally more expensive.

Disjoint pattern databases heuristics are an example of this behaviour. The current solution of using only one heuristic which divides the space in as few parts as possible as long as it fits on a lookup table is not the best solution as it is not scalable nor its is necessarily the fastest pattern database combination.

We introduce a novel solution. We propose to use different heuristics depending on which search level we are. The higher the potential savings the more worth it is to try more accurate but more expensive heuristics. We derive a statistical model which allows us to divide available heuristics on sets which applied on the right search depths should minimize the search time. We also account for the time costs derived of trying more expensive heuristics and also the time costs of statistics gathering. We are in the process of implementing a solver using this principles. We have done most of our previous research regarding HST size predictions. Time costs is the new element on this research and is a work in progress.

7. REFERENCES

[1] Stuart Russel and Peter Norvig, *Aritificial Intelligence: A modern approach Journal*, Prentice Hall,2003.

[2] Nils J Nilsson. *Artificial intelligence: a new synthesis*. Morgan Kaufmann Publishers Inc, 1998.

[3] Richard E. Korf. Recent Progress in the Design and Analysis of Admissible Heuristic Functions. In *AAAI/IAAI:1165-1170*. 2000.

[4] S. Edelkamp. Planning with pattern databases. In European Conference on Planning (ECP), 2001

[5] A Felner, RE Korf, S Hanan. Additive Pattern Database Heuristics. Journal of Artificial Intelligence Research, 2004.

[6] Franco,Santiago. Topological Analysis of Admissible Heuristics in IDA*.Computer Science Graduate Workshop, Auckland University.2006.

AUTOMATICALLY EXTENDING THE COVERAGE OF HIERARCHICAL TASK NETWORK PLANNERS

Andrew William Hay

ABSTRACT

Hierarchical Task Network (HTN) planning has the widest use in practical planning systems because of its speed in discovering solutions. One of the biggest obstacles in the development of HTN planners is the difficulty in obtaining domain dependent knowledge that guides the planning system. By automatically discovering HTN methods, the main knowledge artifacts in HTN planning, we seek to increase the coverage of an HTN planner without requiring a domain expert to craft the HTN methods.

This paper describes a system that extends the coverage of an HTN planner to include a set of problems currently uncovered, given only the set of problems and the solutions to those problems (the plans). The current progress in implementation of this system and what remains to be done are also discussed.

1. INTRODUCTION

Planning systems have three performance attributes. How many problems in a domain they can plan for (or domain coverage), optimality of solutions found and speed of finding solutions. No planning system can be both optimal, have complete coverage and be tractable [1].

For practical planning applications, speed of finding solutions is essential. Classical planners are optimal and complete but are intractable, and are therefore not practical planning systems.

HTN planners solve this problem by sacrificing complete coverage of the domain for speed of planning. This is done by utilizing domain dependent information to guide the planning process, in the form of HTN methods.

1.1. Overview of HTN planning

HTN planning is similar to STRIPS-style planning in that both use state based representations of the world, and the actions in HTN planning are similar to those of STRIPS-style planning. The difference between HTN planners and STRIPS-style planners is in what they plan for and how they plan for it.

Classical planners, or STRIPS-style planners plan by finding a sequence of actions (operators with preconditions) that satisfies certain conditions or "attainment goals". Planning usually proceeds by finding operators that have the desired effects, and by making the preconditions of those operators into subgoals[2].

HTN planners, on the other hand, search for plans that accomplish task networks, which can include goals other than just attainment goals. They plan via task decomposition, in which tasks are decomposed into smaller and smaller subtasks until primitive tasks (which correspond to actions) are reached that can be executed directly.

An HTN planning problem consists of the following[3]: an *initial state*, the *initial task network* and a *domain description* containing:

- A set of *operators* that describe how to execute primitive tasks by adding and deleting items in the current state of the world. These differ from STRIPS operators in that they have no preconditions. The preconditions are enforced in the decomposition methods.

- A set of *decomposition methods* (or simply: methods) that describe the ways of decomposing non-primitive tasks into task networks containing possibly both non-primitive tasks and primitive tasks. Methods usually have a set of constraints, or preconditions that need to be satisfied in order to be applicable.

- Optionally, various information such as definitions of axioms for inferring conditions not mentioned explicitly in states of the world.

An HTN method contains three items, a non-primitive task to decompose, a set of preconditions which need to be satisfied in the current world in order use the method, and a task network which replaces the decomposed task. To learn an HTN method, all three need to be discovered.

HTN methods contain domain specific knowledge that guides the HTN planner through the planning process. There is currently no way of learning these methods without using plan traces, which contain information about the planners (e.g., a human expert) decisions as well as the final solution[3]. Or a domain expert constructing the methods by hand.

By only using the classical solution (plan) of a problem that is not currently covered by an HTN planner to learn HTN methods, it requires less input from an domain expert. This allows learning of HTN methods to become more automatic than the current state of the art.

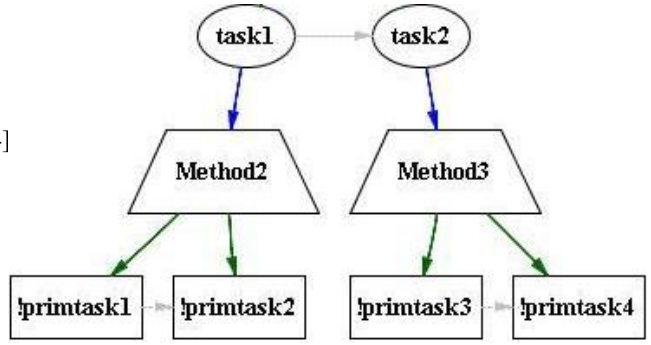
1.2. Current Method Learning Algorithms

HTN method learning algorithms could first only learn the preconditions of methods[3]. More recently now algorithms can learn methods without any prior information on the methods[4]

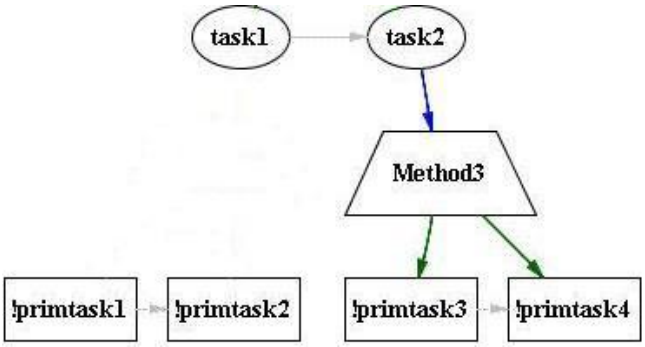
These algorithms all require multiple plan traces to discover the HTN methods. Plan traces contain not only the correct solution to a planning problem but also information about inferences derived and decisions made while the plan was generated. To make these plan traces requires either a domain expert or an HTN planner with the set of HTN methods that are to be learned.

The current state of the art algorithms cannot automatically extend the coverage of an HTN domain, they require inputs that come from domain experts such as plan traces described above.

We propose a system that can extends the coverage of an HTN planner to cover a set of problems currently uncovered, given only the set of problems and the solutions to those problems (the plans). By not using plan traces, this system does not require a domain expert unlike the current state of the art method learning algorithms.



(a) Plan Trace of problem covered by method set.



(b) Plan Trace of problem not covered by method set.

Fig. 1. Figure 1: Example of plan trace with a) all HTN methods, and b) missing one HTN method. Single non-primitive tasks get decomposed by a method into a set of subtasks, in this task 2 primitive tasks.

2. METHOD LEARNING SYSTEM

Consider a set of HTN methods M for solving problems in the HTN domain D, and an HTN planner. When the HTN planner is run using the method set M some problems are in the planners coverage and some are not.

Let P be a currently uncovered problem that is required to be in the coverage of the planning system. To achieve this for an HTN planner, a new method or methods need to be constructed and added to the current method set to expand the coverage of the planner.

Let $M' \supset M$ be the set of methods that when run on the HTN planner covers the problem P (in other words, the planner can find the problems solution).

Looking at the plan trace of the HTN planner using method set M' while solving P will generate a tree of tasks, the top of the tree being the initial task network (as specified in the problem) and the leaves being the primitive tasks (for example: Figure 1a). In between tasks are methods which take a task and decompose it into a task network (a set of tasks).

The methods that need to be constructed are in the set $M' - M$, if those methods are removed from the plan trace, gaps in the plan trace appear (for example: Figure 1b). By finding this plan trace with the gaps using only the method set M, all three items that make up a method can be discovered. Methods in the set $M' - M$ are called missing methods.

With only the methods in the set M, the input to the system, the correct plan trace with gaps cannot be directly computed. But by doing both top-down planning from the initial task network in the problem, and also bottom-up parsing using the solution to the problem, the plan trace with gaps can be constructed.

The top of the gap (or Gap Problem) is the non-primitive task that the missing method decomposes, and the bottom of the gap (or Gap Solution) is the task network that the missing method decomposes the top of the gap into. These constitute two out of the three required items to construct a method, finding the final item (the preconditions) means the method has been learnt. Discovery of preconditions will be discussed later in the paper.

The method learning system consists of three parts:

- Finding the Gap Problem via the Gap Problem algorithm.
- Finding the Gap Solution via the Gap Solution algorithm.
- Finding the preconditions.

2.1. HTN planner and assumptions

The HTN planner that is being used is the JSHOP2 planner[5]. JSHOP2 is an domain-independent HTN planning system that

is based on ordered task decomposition, which involves planning for tasks in the same order as they will be later executed.

There is one assumption that is used throughout the rest of the paper, that there is only one missing method, and thus only one Gap Problem and Gap Solution. However, it is our belief that the Gap Problem and Gap Solution algorithms will work even if there are multiple missing methods but for now the simpler problem will be tackled.

2.2. Gap Problem algorithm

The Gap Problem (or GP) is the task at the top of the gap, the non-primitive task that is decomposed by the missing method. The GP algorithm does not find the GP but a set of candidate GPs. The reason for will be explained below.

The inputs to the GP algorithm are the initial task network for the uncovered problem P, the set of incomplete methods M and the correct problem solution (the plan) S for the uncovered problem.

Consider how the HTN plans for P using method set M, assuming there is only one GP then the HTN will plan as normal until it tries to decompose the GP. A number of things can happen:

- The GP could have no applicable methods.
- The GP could have 1 or more applicable methods.

When there are no applicable methods, the planner will backtrack. Thus when the planner backtracks is a good indication that the non-primitive task thats being backtracked from is the GP.

When there is 1 or more applicable methods, the planner will decompose the GP with one of them and continue planning. However it will be the wrong decomposition, which will result in the planner backtracking to the GP and trying another decomposition if possible.

There is no assurance that the first occurrence of backtracking in the HTN planner is the GP for two reasons. First is the one mentioned above, the GP may have 1 or more applicable methods (but incorrect ones) so the backtracking would not first happen on the GP. Secondly, the planner may backtrack even if there is no GP.

What this means is that every time backtracking occurs in the GP algorithm, the algorithm must both backtrack at the task and also add the task to the set of candidate GP's.

The end result is a set of candidate GP's and each candidate GP has a partial plan trace. The partial plan trace is the plan trace of the planner at the time the candidate GP was the task being backtracked from. The partial plan trace has also a partial plan $S_p \subset S$, this is the partial plan that was found by the HTN planner.

The partial plan trace has a remaining task network t which is the task network that had not been decomposed by the planner at the time, the GP is also inside t.

When these partial plan traces are combined with the Gap Solution algorithms result, this will result in a plan trace with gaps which represents a candidate missing method.

2.3. Gap Solution Algorithm

The Gap Solution (or GS) algorithm finds the task network at the bottom of the gap, the set of primitive and non-primitive tasks that the missing method decomposes the GP into. By then matching the partial plan trace from the GP algorithm and the bottom-up parse of the GS algorithm, a set of candidate missing methods can be constructed. The last part is to select from the set of candidate missing methods, the best missing method.

The inputs to the GS algorithm are the set of candidate GP and partial plan traces from the GP algorithm, along with the set of incomplete methods M and the correct problem solution (the plan) S for the uncovered problem.

For this algorithm, a bottom-up parse of the remainder plan is done to reconstruct the other part of the correct plan trace with gaps. The remainder plan of a partial plan trace is the plan $S - S_p$, in other words the part of the problems solution that the planner could not find before hitting the candidate GP.

The first part of the GS algorithm is to turn each of the methods in M into a grammar rule. The left hand side being the non-primitive task that the method decomposes and the right hand side being the decomposition of the grammar rule.

Then, using this set of grammar rules, a bottom-up parser will generate all the possible parse trees for the remainder plan.

The difference between these grammar rules and normal rules is that since these rules represent methods, so the preconditions have to be checked before the rule can be applied. This is a simple process, as every state is known (the algorithm has the entire plan for the problem). Also, there is no start symbol, the parser simply tries every possible combination of rules.

This parse is run on each candidate GP and its partial plan trace, and for each candidate GP will have a number of bottom-up parse trees.

Before the algorithm can decide which is the best GP and GS, the top-down partial plan trace and the bottom-up parse trees need to be matched. At the end of matching the actual GP and GS merge will result in a correct plan trace with gaps.

Matching is done by matching the bottom of the top-down partial plan trace with the top of the bottom-up parse tree. This is the remaining task network t for the top-down partial plan trace. And the tasks that have no parents in the bottom-up parse tree. These are called the top and bottom remaining lists respectively.

Each task that is in both top and bottom remaining lists and in the correct position, becomes a place where the top-down and bottom-up parses meet. The end result will be 1

or more tasks in the top-down remaining list and some in the bottom-up remaining list (sometimes none) that will not be matched. After matching, the partial plan trace and the parse tree become one single plan trace with gaps in it.

These tasks that are not matched are the GP and GS for the missing methods, GP if its in the top-down list and GS if its in the bottom-up list. Though these are still candidate GP and GS, as there will be more than 1 plan trace with gaps in it.

The last job of the GS algorithm is to decide which of the number of candidate GP and GS pairs is the best one, the heuristic metric for this is complexity. The least complex solution is the best, if the GP is the initial task, and the GS is the entire plan S, that is clearly the worst method to learn for it is very specific.

The primary metric is how many primitive tasks are in the candidate GS, the less primitive tasks the better. Secondary metrics for tiebreakers could include the number of non-primitive tasks in the candidate GS (less is better). The complete heuristic metrics are still being developed.

2.4. Finding Preconditions

After the GP and GS are decided, there still is the third item of a method to deal with, its preconditions. This is a task not yet tackled in detail enough to go over in this paper.

3. CONCLUSIONS

Because planning systems cannot be optimal, have complete coverage and be quick, HTN planners choose to sacrifice coverage for quickness. This means that HTN planners are the most widely used planning system for practical planning applications where speed is essential.

This paper describes a system that extends the coverage of an HTN planner by discovering new HTN methods from uncovered problems and their solutions. It takes method learning a step beyond the current state of the art by not relying on information packed plan traces for input and only requiring the solution or plan of an uncovered problem.

By discovering the correct plan trace with gaps, the Gap Problem and the Gap Solution can be found. The Gap Problem is the non-primitive task that the missing method decomposes into task network that is also the Gap Solution.

At this point in time, the Gap Problem algorithm has been completed and tested to run correctly. The Gap Solution algorithm is currently being implemented, and we have some ingenious ideas in how to tackle the problem of finding the preconditions of the method.

4. REFERENCES

[1] Tom Bylander, "The computational complexity of propositional STRIPS planning," *Artificial Intelligence*, vol. 69, no. 1-2, pp. 165–204, 1994.

[2] Kutluhan Erol, *Hierarchical task network planning: formalization, analysis, and implementation*, Ph.D. thesis, 1996.

[3] Okhtay Ilghami, Dana S. Nau, Hector Muñoz-Avila, and David W. Aha, "Learning preconditions for planning from plan traces and HTN structure," *Computational Intelligence*, vol. 21, no. 4, pp. 388–413, november 2005.

[4] Okhtay Ilghami, Dana S. Nau, and Hector Muñoz-Avila, "Learning to do HTN planning," in *Proceedings of the Sixteenth International Conference on AI Planning and Scheduling*, Cumbria, UK, June 2006, AAAI Press.

[5] Okhtay Ilghami, "Documentation for JSHOP2," Tech. Rep. CS-TR-4694, Department of Computer Science, University of Maryland, February 2005.

GOAL-ORIENTED TESTING OF SEMANTIC WEB SERVICES

M. Shaban Jokhio, Gill Dobbie, Jing Sun

ABSTRACT

Semantic Web services (SWS) has been emerging for past few years. It provides a semantic version of current syntactic Service-Oriented Architecture (SOA). SWS aims at the automation of services process tasks such as web services discovery, selection and invocation etc. This research proposes the methodology of a goal-oriented testing for SWS that shall enable service consumers to test the functionality of services as opposed to the usual service-provider method of software testing. Moreover, the proposed research is based on the emerging Web Services Modelling Ontology (WSMO) framework, which has strong conceptual design basis and is probably to lead the SWS in future.

1. INTRODUCTION

Software testing is one of the main phases of the software development process which consumes an average of 30% to 70% of an organization's resources. [1] Much like common software testing, the web services (WS) testing aims at finding the syntactical or logical errors that may occur potentially in a WS specification or in the backend program to which the web service provides an interface. However, the WS testing is different and more challenging than normal software testing because of following reasons.

1. WS only has application interface and no user interface (UI). The absence of UI causes lack of controllability and difficulty for the testing process.
2. Dynamic nature of WS including dynamic publication, discovery, selection, invocation and monitoring needs the test-process be dynamic.
3. Test-case need to be documented in XML for WS consumption.
4. No internal details of program are known.

WS are an important paradigm in current era of Web computing. A web service defines an interface to a program that is accessible through internet via XML. Although current WS technologies such as WSDL, SOAP and UDDI [2,3,4] have made Web services practical, but the syntactical basis and the lack of machine understandable semantics have caused involvement of great manual work for different Web Services such as manual publication, discovery, invocation and composition. To realize the actual aim of SOA, the SWS adds a layer of semantics to existing web service technologies so that the manual work can be replaced by machine automation. SWS have emerged by combining the semantic web technologies like RDF, RDFS and ontologies etc, with WS technologies as in Fig.1. [5]

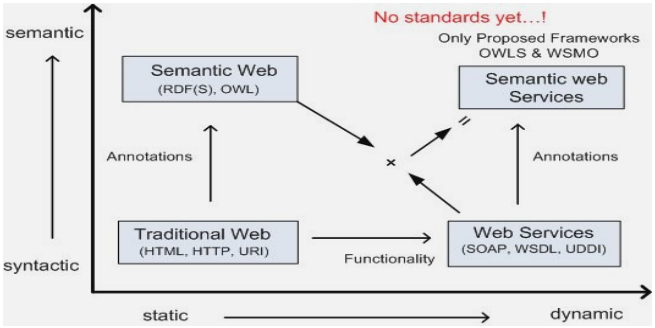


Fig. 1. Evolution of web from syntactic WS to SWS.

Web Services Modelling Ontology (WSMO) is an emerging framework of SWS being developed in DERI research Labs, Innsbruk Austria, since 2004. It is based on strong design principles of scalable mediation and strict decoupling between its elements. WSMO defines four top level elements for Web service modelling, including "*ontology*" to formally specify the terms to be used by other web service elements, "*webservice*" to model the description of computational entities or services,

"*goal*" to formally specify the user objectives that are to be achieved in a web service market place and "*mediator*" which link any of these four components and resolve any potentially occurring mismatches at different levels such as data, process or function levels. Web Services Modelling Language (WSML) provides the base ontology modelling language and Web Services Execution Environment (WSMX) is an execution environment for WSMO as in Fig. [6].

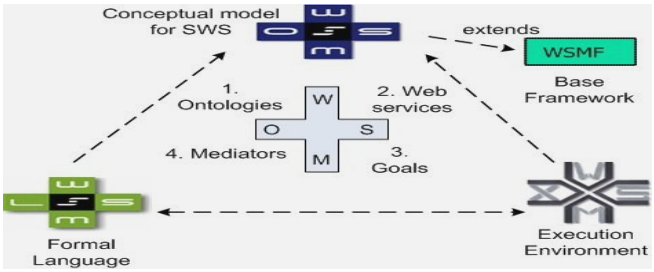


Fig. 2. WSMO overview.

No research has yet been done in WS testing area for testing the semantic web services, especially for WSMO-based semantic web services. The proposed research shall enable the user-oriented testing of semantic web services. Rest of this paper is organized as follows. Section 2 briefly describes related work, Section 3 described the problem definition that is addressed, Section 4 describes the Proposed Solution and Section 5 describes Conclusions and Future Work briefly.

2. RELATED WORK

A lot of researches such as [7, 8, 9, 10, and 11] mainly, were done by the WS testing research community. The advantages of these research methods were the ease of testing procedure with WSDL specification and automated testing tools such as *Coyote* and *WSDLTest* etc, but the disadvantages were that all of these testing methods are limited to the provider's view of testing web services and are restricted to only the current form of web services based on WSDL testing. Very little research has been done so far that aims at testing the SWS.

Wang, Y, et al. [12], however, took a step for SWS testing, proposing a methodology for ontology-based test case generation for WS testing. The service specification in OWLS is given input to the tool and the specification was parsed for the input/output information and operational behaviours of composite service. The operational behaviour was converted to state-machine and the full state-machine ontology was obtained, combining the operational semantics and input/output information. Finally the test-case generation algorithm parses different paths of the state-machine, analyzing the inputs and outputs of every atomic process and generated the test cases. The approach is quite feasible and generates the effective test-cases but it is again provider's view of testing the service that may not be visible to the service user.

3. PROBLEM DEFINITION

Goal-oriented design is considered to be one of the best practices of software designing. [13]. User goals have primary importance throughout the software development process. A good software must identify the user goals properly, fulfil them completely and then test for their fulfilment.

The user goals become more important in an open WS market place where users find a web service, invoke it and consume its functionality. Therefore it is important to have a method which can let the users to test the functionality of a web service from the goal-fulfilment point of view.

For SWS that adds semantics to automate the WS usage such as WS discovery, selection and invocation etc., it becomes more important to have such a testing methodology. The testing of service from user point of view may help in appropriate service selection and invocation. For example, a user goal such as "*Buy a book*" can be fulfilled by several services whereas the user may select only that service which is tested by user and found to be functioning correctly on all user inputs. In other words this can help users find, select or invoke a user-reliable service i.e. a service which the user trusts to operate correctly. An analogy would be the beta-testing of software which is done by the end-

users and reveals more errors than the alpha-testing done by the software developers.

The existing WS testing frameworks test a web service only from a given service specification (WSDL or OWLS) which is only visible to the service provider and not to service client. Unfortunately no research in WS testing research area has been done so far that proposes to test the web services from user point of view.

This research aims to propose such a methodology for WS testing which can let the users test the services from their goal perspectives and let them ensure the fulfilment of their goal before the actual service usage.

4. PROPOSED SOLUTION

To achieve the goal-oriented testing of semantic web services as described above, the WSMO framework is chosen due to following reasons:

1. WSMO is the only framework of SWS which provides the formal specification of the user objectives as the goals. Moreover, these user goals are modelled as first class citizens or top level elements in WSMO.
2. WSMO provides a strong conceptual basis for SWS and is probably to be the W3C recommendation in future when the semantic web services shall be used as in real.

4.1 High level Methodology

An abstract high level methodology for goal-oriented testing is shown in Fig. 4. The goal specification given as input to the tool can be parsed for extracting goal capability and interface information with help of tools such as WSMO4j- (an API for building applications with WSMO). Next, the test-case generator can generate test-cases from capability or interface specification as described above. Ontology-base that contains domain ontologies, sample test data in WSML and Knowledge Base (KB) for test rules, can be established from *importsOntology* part of goal specification. An ontology reasoner for WSML such as MINS/IRIS can be employed to generate the test-data inferred from ontologies for more efficient testing purpose. Thus the test cases generated in this way can be used for WSMO-based semantic web service testing. The test-results can be analyzed and the feedback can be supplied to the ontology-base for updating the knowledge-base for test-case generation rules to generate more efficient test-cases for next time.

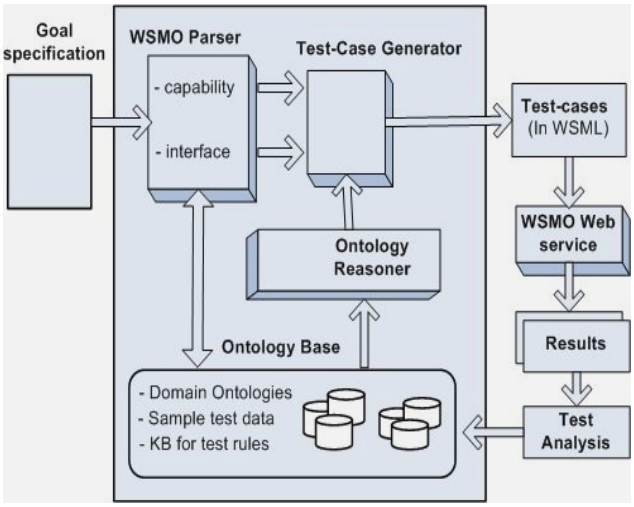


Fig. 4. Overview of High Level Methodology for goal-oriented testing of SWS.

4.2 WSMO goal specification

WSMO goal specifies the requested view of the service. A typical WSMO goal specification is given in listing 1.

```
Class goal
   hasNonFunctionalProperties type
     nonFunctionalProperties
   importsOntology type ontology
   usesMediator type {ooMediator,
       ggMediator}
   requestsCapability type
     capability
   requestsInterface type interface
```

Listing. 1. WSMO goal specification.

Within above goal-specification two elements "*requestsCapability*" and "*requestsInterface*" elements can be used to generate the test-cases. We discuss how these two elements can be used for the purpose of generating test cases. The capability element is further specified as the set of four sub-elements:

1. *precondition:* defines the constraint on the information that must be available for successful fulfillment of the goal.
2. *assumption:* defines the condition on real world that holds true for successful fulfillment of the goal.
3. *postcondition:* defines the information produced after the goal is achieved.
4. *effect:* define the real world changes after the successful achievement of this goal.

The "*requestsInterface"* element defines the requested view of the interface to achieve this goal in the form of the requested choreography- (a desired format of interaction with the service that fulfills the particular goal) and a requested orchestration- (a desired format of how the service should compose other services to achieve the particular goal). Each of the choreography or orchestration further defines two sub-elements: i.e. state or state-signature- (defining the states of concepts which are used to interact with the requested service) and the transition-rules (dictate how the states of the concepts are changed based on different condition). A pseudo code example of the BuyBookGoal is given in Listing 2.

```
goal BuyBookGoal
importsOntology
      BuyBookOntology,
CreditCardOntology
capability
  precondition
     - purchaseRequest
     - creditiCard
   postcondition
           -  bookReceipt
   assumption
      - credit card is valid
   effect
     - credit card charged with the
     price of the book
interface
   choreography
     state
        in
          purchaseRequest
          creditCard
        out
          failureNotice
          bookReceipt
        controlled
          tempReceipt
transitionRules
 if (purchaseRequestInstance is
      received)
 then
   create(new tempReceiptInstance)
 endIf
 if (tempReceiptInstance  exists)
   and creditCardInstance valid)
 then
   create new BookReceiptInstance)
 endIf
 if (purchaseInstance is received)
     and (creditCard is not valid)
 then
    (create new FailureInstance)
   endIf
```

Listing. 2. A BuyBook Goal Example.

4.3 Generating Test-Cases from goals

The critical analysis reveals that WSMO goal specification is capable to be used for generating test-cases to test the service functionality that fulfils the specified goal. Test cases from goals can be generated in two ways: From requested capability specification and from requested interface specification. This is shown in figure. 3 below

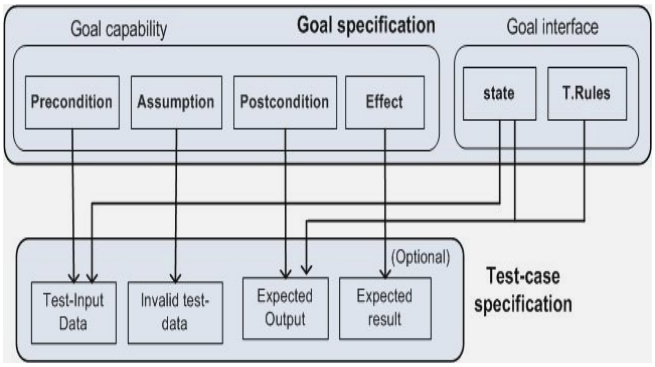


Fig. 3. Test case generation from goals.

In the requested capability, "*preconditions"* provides an input space that can be used to generate the test-input data, "*postconditions"* specifies the constraints on information space after successful achievement of goal and can be used to record the expected output, "*assumptions"* can be used to generate a negative test-case that violates the important real world condition and the effect can

optionally be used to record the expected result after the successful achievement of the goal.

In *"requested interface"* specification, the state signature specifies the "*in*" state and can be used to generate the test-input data and the "*out*" state together with "*transition rules*" can be used to record the expected outputs in a test-case.

4.3 Evaluation of Test-cases

Effectiveness of test-cases generated can be measured on the basis of capability coverage and interface coverage. Capability coverage specifies the number of precondition, assumption, postcondition and effect parts covered; where as interface coverage specifies the state signature coverage (number of *in* and *out* states covered) and transition rules coverage (number of transition rules covered).

5. CONCLUSION & FUTURE WORK

The proposed research "Goal-Oriented Testing of semantic web services" is entirely a new concept of web service testing, which shall let the service users ensure the functionality before selecting and actually consuming the service functionality. It can be implemented by WSMO framework. We have yet worked out only the high level solution for the simplified case however we intend to do following in future.

1. Design a concrete high level methodology.
2. More effective way of test-case effectiveness measurement.
3. Solutions for complex issues like mediators.
4. Implementing prototype system to evaluate proposed research.

6. REFERECES

[1] Phadke, M.S, "Planning Efficient Software Tests",1997 http://www.stsc.hill.af.mil/crosstalk/1997/10/planning.asp

[2] Chinnici, R, Moreau, J.J., Ryman, S. "Web Services Description Language (WSDL)" http://www.w3.org/TR/wsdl20/wsdl20-z.html, 2006

[3] Mitra, N, "SOAP V1.2 Part-0: Primer", http://www.w3.org/TR/2002/WD-soap12-part0-20020626/, June 2002

[4] Rogers, T, Clement, L, Kumar, R, "OASIS UDDI Specification", http://www.oasis-open.org/,

[5] Stollberg, M "Semantic Web Services, Realizing SOA vision", SWS tutorial Industry workshop, DERI Innsbruk, February 2007, page. No. 2, 3

[6] Raman D, Lausen, H, Keller, U, "Web Services Modelling Ontology", D2V1.3, http://www.wsmo.org/TR/d2/v1.3/, 2006

[7] Tsai, W. T, Paul, R, Wang, Y, Fan, C, Wang, D, "Extending WSDL to Facilitate Web Service Testing", pp 1-2, 2002.

[8] Tsai, W. T, Paul, R, Song, W, Cao, Z, "Coyote: An XML-based Framework for Web Services Testing", pp 1-2, 2002

[9] Siblini, R, Mansour, N, " Testing Web Services", pp1-6, 2005

[10] Bai, X, Dong, W, "WSDL-Based Automated Test Case Generation for Web Services Testing", pp 1-6, 2005

[11] Sneed, M, H, Huang, S, "WSDLTest, Tool for Testing Web Services", pp 1-8, 2006

[12] Wang, Y, Bai, X, Li, J, Haung, R, "Ontology-based Test Case Generation for Testing Web Services", pp 1-8, 2007

[13] Schnabel, I, Pizka M, "Goal-Driven Software Development", pp 1-6

VERIFYING NORMALIZATION ALGORITHMS FOR SEMISTRUCTURED DATA

Scott Uk-Jin Lee

ABSTRACT

The rapid increase in semistructured data usage has resulted in various developments in database systems for semistructured data. Many web services and applications that utilize large amounts of semistructured data require the developed database systems to minimize redundancies and update anomalies. Several normalization algorithms for semistructured database systems have been proposed to satisfy the demands. However, currently proposed normalization algorithms lack verification to ensure the preservation of the data and the constraints on the data. It is critical for the correctness of these normalization algorithms to be verified. Otherwise the data and the constraints between the data could be lost or corrupted during the normalization process. In this paper, we propose a declarative methodology to verify the correctness of normalization algorithms developed for semistructured data in the ORA-SS data modeling language. The verification methodology is developed by adapting the concept of dependency preserving and lossless join properties from the relational database systems into the semistructured data context. The methodology verifies the correctness of the normalization algorithms by checking whether the functional dependencies are preserved, data is not lost, and spurious data is not created during the application of normalization algorithms.

1. INTRODUCTION

The rapid growth of the World Wide Web and its technologies has resulted in enormous amounts of data being used over the Internet by Web Services and Web-based applications. The increase in semistructured data usage is not limited to Web applications but expands into various other applications such as digital libraries, biological databases and multimedia data management systems. This expansion of semistructured data usage creates the need for effective and efficient utilization of semistructured data [1].

With such a rapid increase in its usage semistructured data needs to be stored, manipulated, and queried to be utilized properly by various applications and tools. For these purposes, many researchers have proposed to design and develop adequate database systems for semistructured data. As a result, several database systems have already been developed for eXtensible Markup Language (XML) [2], which is a common representation for semistructured data, while traditional database companies, such as Oracle, have provided XML support for their existing database systems.

As with widely used database systems such as relational database systems, redundant data stored in XML database systems must be minimized otherwise it can cause data inconsistency and anomalies [3]. Several normalization algorithms have been proposed and used to minimize redundancies in semistructured database systems by transforming the schema of the semistructured data. It is possible for the schema transformation in the normalization process to lose or corrupt the semistructured data or the dependencies between the data if normalization algorithms are designed incorrectly. The consequences of losing or corrupting data or the constraints between the data are devastating especially for databases used for online banking applications, credit card transactions, government's legal applications, and any other applications dealing with critical information.

However, currently developed normalization algorithms for semistructured data lack verification that would ensure preservation of the data and the constraints on the data. The normal form for semistructured data (NF-SS) developed by Wu et al. [4], XML normal form (XNF) developed by Embley and Mok [5], and normal form for XML documents developed by Arenas and Libkin [6] are examples of the algorithms and rules for normalizing semistructured data. These normalization algorithms use different methods to minimize redundancies in semistructured database systems. Without adequate verification, these algorithms could cause more damage than the promised benefits since the process of minimizing redundancies is useless if it corrupts the data.

In widely adopted database systems, one of the features that is used to prove the correctness of the normalization algorithms is the mathematical foundation. For example, in relational database systems, a mathematical foundation has been extraordinarily useful in the definition of normalization, to prove that lossless and dependency preserving algorithms [7] can be defined. Such verification support for normalization and its algorithms ensures the consistency of the data and the constraints on the data.

Therefore, this research proposes an adequate verification methodology that proves the correctness of normalization algorithms for semistructured data. The verification will be provided through an establishment of the mathematical foundation for the semistructured data. The derived mathematical foundation will verify whether normalization algorithms that transform the schema of semistructured data preserves the data and the constraints on the data. Furthermore, the well defined mathematical foundation will provide various addi-

tional benefits to the database for semistructured data.

2. PROPOSED RESEARCH

The main objective of this research is to provide verification methodology that proves the correctness of normalization algorithms used for semistructured data. A well defined mathematical foundation will be establish to satisfy this objective since it is proven to provides effective verification for the normalization algorithms in widely used database systems.

The mathematical foundation for semistructured data requires two main components to provides an adequate verification methodology for proving the correctness of normalization algorithms. A formally defined and verified semantics of an adequate data modeling language for semistructured data is one component and a formally defined and verified schema transformation operators with its verification criteria for normalization process and algorithms is the other.

The formal semantics for semistructured data must be defined first to provide a standard representation of schemas. Then the formal definition for transformation operators with its verification criteria for normalization processes and algorithms can be defined on top of the formal semantics. Also to provide the correct verification methodology for the normalization algorithms, the formally defined mathematical foundation must be supported with adequate formal verification tools for its verification. Hence, to define a mathematical foundation for semistructured data, a standard representation of schemas and the normalization of schema for semistructured data must be formally specified and verified using adequate data modeling languages and formal languages.

2.1. Formal Semantics for Semistructured Data

In mathematical foundation for semistructured data, formal semantics for semistructured data must be defined to provide standard representation for schema and the data of semistructured data. Also it is essential requirement for defining schema transformation operators to represent and prove normalization process and algorithms.

There has been many other researches which proposed a formal semantics for semistructured data. For example, the formalization of DTD (Document Type Definition) and XML declarative description documents using expressive description logic has been presented by Calvanese et al. [8]. Anutariya et al. presented the same formalization using a theoretical framework developed using declarative description theory [9]. Also spatial tree logics have been used to formalize semistructured data by Conforti and Ghelli [10]. More recently, hybrid multimodal logic was used to formalize semistructured data by Bidoit et al. [11]. While these works have helped us develop a better understanding of the semantics of semistructured data, none of them have applied adequate and automated verification. Furthermore, none of these researchers have considered providing specification and verification for operations and algorithms that transforms semistructured data schema.

As a result of examining related work and intensive background research, Object Relationship Attribute model for semistructured data (ORA-SS) data modeling language [12, 3] will be adopted as a data modeling language. The ORA-SS data modeling language is used because it not only captures the constraints that are represented in textual languages such as XML Schema [13] but also it is a diagrammatic notation which can be used for conceptual modeling.

With ORA-SS, we also applied a similar approach to formalize semistructured data using Z/EVEs [14] and Alloy [15]. But the approach using Z/EVEs had problems with complicated and time consuming verification and the approach using Alloy had a scalability problem. Considering these problems, the research will use Prototype Verification System (PVS) [16] and its theorem prover as the formal specification language and verification tool. Also PVS has proven its effectiveness by providing precise formal definitions and powerful automated verification support in various other research projects [16].

With ORA-SS data modeling language and PVS formal language the following tasks has been conducted to provide formal semantics of semistructured data.

- Specifying formal definition of schema representation in ORA-SS data model using PVS formal specification language
- Verifying the formally defined ORA-SS schema diagram
- Specifying formal definition of data representation in ORA-SS data model using PVS formal specification language
- Verifying the formally defined ORA-SS instance diagram
- Specifying formal definition for relationship between schema and data in ORA-SS data model using PVS formal specification language
- Verifying the formally defined relationship between the schema diagram and the instance diagram in ORA-SS

At the completion of the above formal definition and verification, the formally defined and verified semantics for semistructured data is represented. It provides a standard representation for schema and data of semistructured data. Additionally, the above formal definition provides a verification for the schema instance and the data instance of semistructured data against the semantics of ORA-SS data model language. The verification for the data instance of semistructured data against its schema is provided as well through the above definitions.

The formally defined ORA-SS semantics also enables the formal definition of verification methodology for normalization processes and algorithms. On top of the above definitions, the verification criteria for the correct normalization can be formally defined and verified. The formally defined verification criteria can then be used to formally define transformation operators to complete the methodology for verifying normalization process as well as its algorithms.

2.2. Verification of Normalization Algorithms

The verification methodology for normalization algorithms of semistructured data will be defined on top of the ORA-SS formal semantics defined in PVS formal language. The methodology will consists of verification criteria for the normalization process and verified schema transformation operators for normalization algorithms.

The verification criteria adapts the concepts and rules for dependency preserving and lossless property [7] from relational database systems. However, the dependency preserving and lossless properties of relational database systems are defined specifically for relational database systems. In order to utilize these properties for semistructured data systems, dependency preserving and lossless properties specific to semistructured data context need to be defined. The dependency preserving property ensures that the business logics of the data represented in functional dependencies are preserved. The lossless properties ensure that the transformed schema does not lose data nor create spurious data. The combination of the two properties provides a definition for data equivalence that can be used to verify the correctness of normalization algorithms for semistructured data.

A declarative definition of dependency preserving and lossless properties provides an appropriate and effective way to verify the correctness of normalization processes for semistructured data. These properties provide reliable verification since the dependency preserving and lossless properties are proven to ensure the equivalence of the data and verify the correctness of normalization algorithms in relational database systems.

With the declarative definition of dependency preserving and lossless properties the following tasks will be conducted to provide verification methodology for normalization of semistructured data.

- Specifying formal definition of dependency preserving and lossless properties for semistructured data according to the specification of ORA-SS formal semantics
- Verifying the formally defined dependency preserving and lossless properties for semistructured data
- Specifying formal definition of transformation operators according to the specification of ORA-SS formal semantics
- Applying formally defined dependency preserving and lossless properties for semistructured data to the transformation operators as constraints
- Verifying the correctness of the normalization and its algorithms using transformation operators (whether they maintain lossless and dependency preserving property)

For the verification methodology described in several tasks above, verification criteria for normalization process will be applied as constraints for every schema transformation operators defined. With the constraints applied, the series of schema transformation operators will be used to represent the normalization algorithms where it will be proved using PVS verification support. The algorithm is proved to be correct, if every constraints of the transformation operator series, representing the algorithm, is satisfied. Hence, the completion of the formal definitions and verification described above will provide an adequate verification methodology for normalization algorithms. Additionally, the research can compare different normalization algorithms derived for each database concept based on its performance to find the best algorithms using the verified representations of the database concepts.

This part of the research extends the defined mathematical foundation even further by enabling its practical applications on utilization of semistructured data in various applications and in its database systems. Also by representing the normalization algorithms and verifying their correctness, the research will demonstrate the correctness and applicability of the mathematical foundation.

At the completion of all these tasks, the research will have provided an adequate verification methodology for proving the correctness for normalization algorithms through the definition of a mathematical foundation. The defined mathematical foundation of the research, that consists of verified formal specification of ORA-SS data model semantics, incorporated schema transformation operators and the verified representation of the best algorithms for each database operations, will be powerful enough to ensure the correct, effective, and efficient use of semistructured data in various applications as well as its database systems. In addition, the research will also help the ORA-SS data model to evolve and provide a possibility for real Web-based applications to be developed from the ORA-SS data model.

3. CONCLUSIONS

This proposed research will provide a verification methodology for semistructured data to prove the correctness of normalization algorithms. The verification methodology will be provided through the establishment of mathematical foundation for semistructured data. The advantages of having such a mathematical foundation includes providing formal semantics for semistructured data design, enhancing the discovery

of inconsistencies in the data, providing verification for correctness of database operations such as normalization and view definitions, and maintaining lossless and dependency preserving properties of algorithms for database systems. Also the generic nature of the defined mathematical foundation allows it to be applied to any applications or database systems that use semistructured data.

Currently, the formal semantics of ORA-SS data model language has been specified and verified using PVS and its verification support. Also lossless and dependency preserving properties for semistructured data has been formally defined on top of the ORA-SS formal semantics and used as correctness criteria for normalization algorithms. Using the formally specified and verified ORA-SS semantics and normalization correctness criteria, basic transformation operators will be defined and verified. Furthermore, based on the defined semantics of the ORA-SS data model and basic transformation operators, various normalization algorithms developed for semistructured data will be defined and verified. Then the verification methodology defined through mathematical foundation will be evaluated through several case studies.

For the future works of the research, other various database concepts such as view creation can be defined and verified based on the defined semantics of the ORA-SS data model and basic transformation operators. Also this addition to the mathematical foundation can be evaluated through several case studies of conducting verification for view definitions and their algorithms.

4. REFERENCES

[1] Xiaoying Wu, Tok Wang Ling, Mong Li Lee, and Gillian Dobbie, "Designing Semistructured Databases Using the ORA-SS Model," in *WISE '01: Proceedings of 2nd International Conference on Web Information Systems Engineering*, Kyoto, Japan, 2001, IEEE Computer Society.

[2] Elliotte Rusty Harold and W. Scott Means, *XML in a Nutshell*, O'Reilly, Sebastopol, 3rd edition, 2004.

[3] Tok Wang Ling, Mong Li Lee, and Gillian Dobbie, *Semistructured Database Design*, vol. 1, Springer-Verlag, 2005.

[4] Xiaoying Wu, Tok Wang Ling, Mong Li Lee, Sin Yeung Lee, and Gillian Dobbie, "NF-SS: A Normal Form for Semistructured Schemata," in *In Proceedings of International Workshop on Data Semantics in Web Information Systems (DASWIS-2001)*, Yokohama, Japan, November 2001, Springer-Verlag.

[5] David W. Embley and Wai Yin Mok, "Developing XML Documents with Guaranteed "Good" Properties," in *ER '01: Proceedings of the 20th International Conference on Conceptual Modeling*, London, UK, 2001, pp. 426–441, Springer-Verlag.

[6] Marcelo Arenas and Leonid Libkin, "A normal form for XML documents," *ACM Trans. Database Syst.*, vol. 29, no. 1, pp. 195–232, 2004.

[7] Ramez Elmasri and Shamkant B. Navathes, *Fundamentals of Database Systems*, Addison-Wesley, 4th edition, 2004.

[8] Diego Calvanese, Giuseppe De Giacomo, and Maurizio Lenzerini, "Representing and Reasoning on XML Documents: A Description Logic Approach," *Journal of Logic and Computation*, vol. 9, no. 3, pp. 295–318, 1999.

[9] Chutiporn Anutariya, Vilas Wuwongse, Ekawit Nantajeewarawat, and Kiyoshi Akama, "Towards a Foundation for XML Document Databases," in *EC-Web*, 2000, pp. 324–333.

[10] Giovanni Conforti and Giorgio Ghelli, "Spatial Tree Logics to reason about Semistructured Data," in *SEBD*, 2003, pp. 37–48.

[11] Nicole Bidoit, Serenella Cerrito, and V. Thion, "A First Step towards Modeling Semistructured Data in Hybrid Multimodal Logic.," *Journal of Applied Non-Classical Logics*, vol. 14, no. 4, pp. 447–475, 2004.

[12] G. Dobbie, X. Wu, T. Ling, and M. Lee, "ORA-SS: Object-Relationship-Attribute Model for Semistructured Data," Tech. Rep. TR 21/00, School of Computing, National University of Singapore, 2001.

[13] Henry S. Thompson, C.M. Sperberg-McQueen, Noah Mendelsohn, David Beech, and Murray Maloney, "XML Schema 1.1 Part 1: Structures," http://www.w3.org/TR/xmlschema11-1/.

[14] Scott U. Lee, Jing Sun, Gillian Dobbie, and Yuan Fang Li, "A Z Approach in Validating ORA-SS Data Models," in *3rd International Workshop on Software Verification and Validation*, Manchester, United Kingdom, 2005.

[15] Lin Wang, Gillian Dobbie, Jing Sun, and Lindsay Groves, "Validating ORA-SS Data Models using Alloy," in *17th Australian Software Engineering Conference (ASWEC 2006)*, Sydney, Australia, 2006.

[16] S. Owre and J. M. Rushby and and N. Shankar, "PVS: A Prototype Verification System," in *11th International Conference on Automated Deduction (CADE)*, Deepak Kapur, Ed., Saratoga, NY, jun 1992, vol. 607 of *Lecture Notes in Artificial Intelligence*, pp. 748–752, Springer-Verlag.

DESIGN OF A MODULAR GPU-BASED DIRECT VOLUME RENDERING FRAMEWORK FOR SCALAR AND MULTIVARIATE DATA SETS

*Felix Manke (supervised by Burkhard Wünsche)**

ABSTRACT

Direct Volume Rendering (DVR) is a technique for displaying volumetric data sets without generating intermediate representations. Traditionally, DVR has only been applied to scalar data sets, such as Computed Tomography (CT) scans. However, dramatic advances in biomedical imaging and other research areas have resulted in ever increasingly complex data. So far, no DVR techniques have been developed that support the interactive exploration of arbitrary volumetric data sets.

We discuss the design of an interactive rendering framework that is capable of dealing with arbitrary volume data sets and visualisation methods. We utilise advanced features of programmable consumer graphics cards to gain both flexibility and performance.

1. INTRODUCTION

In many scientific research areas three-dimensional (3D) discrete data arrays need to be analysed. The visualisation of the internal structures of the data helps researchers to investigate and understand the data.

In DVR, the input volume data set is considered to consist of a "virtual" gaseous material which interacts with the incoming light [1]. The optical properties of the virtual material are modelled by a *transfer function* that maps data values to colours and opacities. In this way internal structures and objects of interest can be classified and emphasised.

Modern graphics processing units (GPUs) make interactive DVR on consumer hardware possible. Current research mainly concentrates on improving efficiency and perception of DVR [2, 3, 4]. This is not surprising, as visual quality and performance is always an important issue. For the purpose of demonstrating their achievements, researchers usually implement specialised renderers which are not flexible enough to interactively explore arbitrary volumetric data.

Especially when dealing with higher-dimensional data, new information has often to be derived before the volume can be displayed (for example gradient vectors or eigenvalues). The ability to combine several data sets is also demanded, since insights can often be gained by comparing the difference or correlation between data sets.

In this paper, we discuss the design of an interactive and flexible GPU-based direct volume renderer. The aim is to support an easy integration of arbitrary data into the rendering framework — regardless of its dimensionality or representation (for example scalar, vector, or tensor data). Moreover, user-defined visualisations of the internal structures have to be possible. Here, our main focus is to develop mechanisms that easily allow the derivation of new values out of existing data and the combination of data sets. Generally, as much of the computation as possible is to be performed on the GPU to benefit from its enormous computational power.

*Graphics Group, Department of Computer Science.

2. THE DVR PROCESS

Figure 1 illustrates the steps that are performed in direct volume rendering. At first, input data sets are loaded. When dealing with multivariate data, meaningful entities often have to be derived or data sets have to be combined. Note that conventional rendering applications usually only support to load scalar volume data and to derive the gradient vector from it.

In the next step a transfer function is evaluated. A selection of data values and derived entities serves as input for the classification. The transfer function maps this input to colours and opacities. As explained above, objects are differentiated by assigning different colours and opacities, which happens at exactly this stage.

During the rendering, shading and illumination effects can be applied to achieve a better perception of the three-dimensional structures. Finally, the volume is projected onto the image plane and displayed.

The actual realisation of this general DVR pipeline depends on the selected algorithm. In the last 20 years many different DVR approaches have been proposed [5, 6, 2]. However, in the end they all follow the steps shown in figure 1.

3. REALISATION OF A MODULAR RENDERING FRAMEWORK

We concentrate on utilising the computational power of modern GPUs. Thus, DVR algorithms, the derivation of entities, and visualisation techniques are all executed on GPU. However, GPU programming is very restricted, because GPUs are highly specialised stream processors. They do not offer the same flexibility as general purpose central processing units (CPUs). A developer can write short micro-programs called *vertex* and *pixel shader*. Nowadays, three high level shading languages are in use, which are generally very similar in syntax and functionality. However, to realise our goals we make

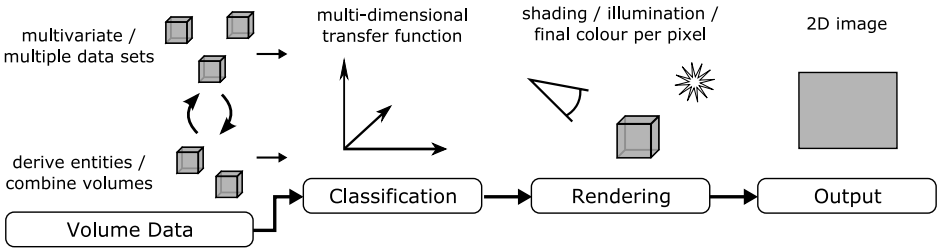


Fig. 1. The stages of the DVR process. For multivariate data sets especially the pre-processing and classification is usually more complex than in the case of scalar data.

use of advanced features that only the language *C for Graphics (Cg)* provides [7].

When comparing our research objectives with the DVR process (figure 1) it becomes clear that flexibility is needed at each stage of the process:

Data initialisation stage: It has to be possible to integrate and load arbitrary volume data sets and file formats. Additionally, the user must be able to derive whatever entities are needed and to freely combine different data.

Classification stage: The framework must be flexible enough to support any classification that is suitable for the current domain and data set. The design of a transfer function largely depends on the data values and entities selected as input. Additionally, a reasonable classification of structures is only possible with knowledge of the data.

Rendering stage: New DVR algorithms must be integratable into the framework. More importantly, it must be possible to define and switch between visualisation techniques so that the user can emphasise different aspects of the data and improve the visual perception.

3.1. Flexibility for Developers

We can observe that both the integration of new data formats and DVR algorithms are tasks for developers, since they have to be done *before* a user can work with the framework. From this observation we derived a unified scheme that makes extending the framework as straightforward as possible.

First of all, a generic template factory lets developers register new implementations in a single line of code. After registration, a sub-class is instantiable throughout the application using a unique identifier, without the need of knowing the concrete data type.

In order to easily initialise objects at start-up, we developed a unified design for initialising all possible state variables. Two fundamental problems arise when dealing with state variables of unknown objects (as they may be present in the framework due to sub-classing by other developers): The state variables themselves are unknown (that is, their "name" or signature) and their data type may differ.

To be able to initialise the state variables of the unknown objects we introduce a design which we call *parameter design pattern*: Every pair of a `Get...` and a `Set...` method is encapsulated by a "parameter" object that hides the data type of the state variable using a standardised string representation, which is used by the application during initialisation.

With these two concepts, the generic factory and the parameter design pattern, we are able to automatically instantiate and initialise new implementations.

3.2. Flexibility for Users

The most important user-specific aspects of the rendering framework are the specification of data to load, the processing of the data, and finally the specification of visualisation techniques and shading effects.

As already mentioned, the derivation of new entities is to be done on the GPU. We call the Cg code blocks or modules that implement this derivation *operators*. The main input of an operator are volume data sets and the output is a texture object that holds the derived values. Note that, by specifying operators in the CgFX syntax, the full capabilities of the graphics hardware is available for the execution (vertex and fragment shaders, multiple render targets, etc.).

Note that loaded data is exclusively used by operators and visualisation effects, which both run on the GPU and therefore will be implemented by the user as Cg shader programs. Hence, to facilitate a practicable work with the framework, we keep the specification of resources and the pixel-shader definition at the same location.

Further on, we can observe that the Cg shader code executed during rendering can be separated into code that is

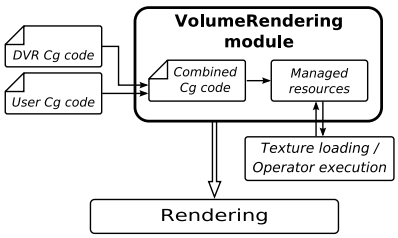


Fig. 2. The volume rendering module that manages the Cg code and resources.

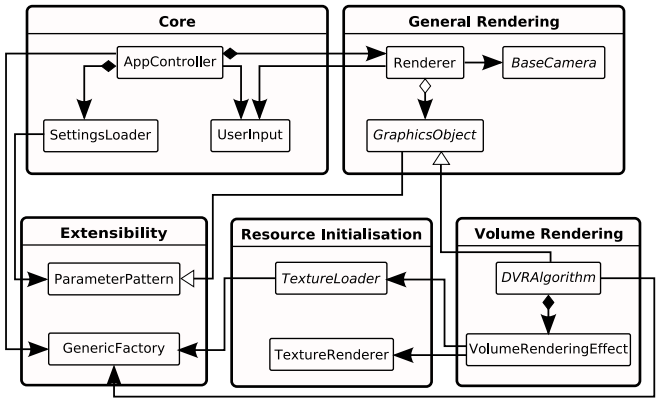


Fig. 3. The main components of the volume rendering application. Note that `VolumeRenderingEffect` combines Cg shaders and manages the specified resources.

specific to the DVR algorithm on the one hand (written by developers) and code that is specific to a user-defined visualisation technique on the other hand (written by the user). To make arbitrary combinations possible, the two types of code are physically and conceptually separated.

As shown in figure 2, our framework contains a module that assembles the Cg code of the DVR algorithm and the user's Cg code and manages the specified resources (volume data sets or textures). During the initialisation, the compiled Cg code is analysed using the Cg Core Runtime API, data sets are loaded and operators executed.

By making use of Cg interfaces, the implementation of a visualisation technique (which we call *evaluator*) is held abstract and the user can define as many different evaluators as desired using several implementations of the abstract interface. The volume rendering module makes interactive switching of evaluators possible.

3.3. Components of the framework

The main components of our rendering framework are shown in figure 3. Besides the concepts and modules discussed so far, the framework contains a controller object that controls the entire program execution (initialisation, rendering and termination). During the start, a configuration file is parsed. It contains settings that specify global states of the application. Further on, a renderer is introduced that renders all graphical objects and updates the camera according to the user input.

4. RESULTS

To demonstrate the applicability of the discussed concepts, our prototype implements two different DVR algorithms as well as support for different data formats. In three case studies we show how to define resources, operators and evaluators. At first, we used a scalar CT scan of the head of the *Visible Male* (acquired by [8], downloaded from [9]). Renderings of different evaluators are shown in figure 4. Then, we used a CT and a Positron Emission Tomography (PET) data set of a monkey (acquired by [10], downloaded from [9]) to demonstrate how to combine several volumes (see figure 5). In the third case study, we procedurally generated a 3D vector field with an operator and visualised it using an interactive 3D Line Integral Convolution (LIC) technique (see figure 6).

Note that, besides the multitude of implemented evaluators and besides implementing several different operators, the source code is very well structured and short (between two and 16 lines per evaluator), because the Cg code for the DVR algorithm is separated.

For a detailed discussion of the design of our framework and the results we outlined in this paper, please see [11].

5. CONCLUSION AND FUTURE WORK

We have presented a direct volume rendering framework that is GPU-based and, at the same time, flexible enough to integrate arbitrary data sets and DVR algorithms and to implement customised visualisation techniques. In contrast to existing visualisation tools our solution enables the user to interactively process, display and investigate complex data sets. We use advanced mechanisms of the Cg language to provide a flexibility that is usually difficult to achieve on the specialised and restrictive graphics hardware.

In the future, our implementation could be further extended to provide more functionality (for example, volume clipping). Additionally, the usage of the framework could be simplified by creating a graphical user interface which offers menus and dialogues for loading data and for deriving entities using operators. Cg source code could be created on-the-fly according to the user's demands.

6. REFERENCES

[1] P. Sabella, "A rendering algorithm for visualizing 3d scalar fields," in *SIGGRAPH '88: Proceedings of the 15th annual conference on Computer graphics and interactive techniques*. New York, NY, USA: ACM Press, 1988, pp. 51–58.

[2] C. Rezk-Salama, K. Engel, M. Bauer, G. Greiner, and T. Ertl, "Interactive volume on standard pc graphics hardware using multi-textures and multi-stage rasterization," in *HWWS '00: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS workshop on Graphics hardware*. New York, NY, USA: ACM Press, 2000, pp. 109–118.

[3] J. Kniss, S. Premoze, C. Hansen, and D. Ebert, "Interactive translucent volume rendering and procedural modeling," in *VIS '02: Proceedings of the conference on Vi-*

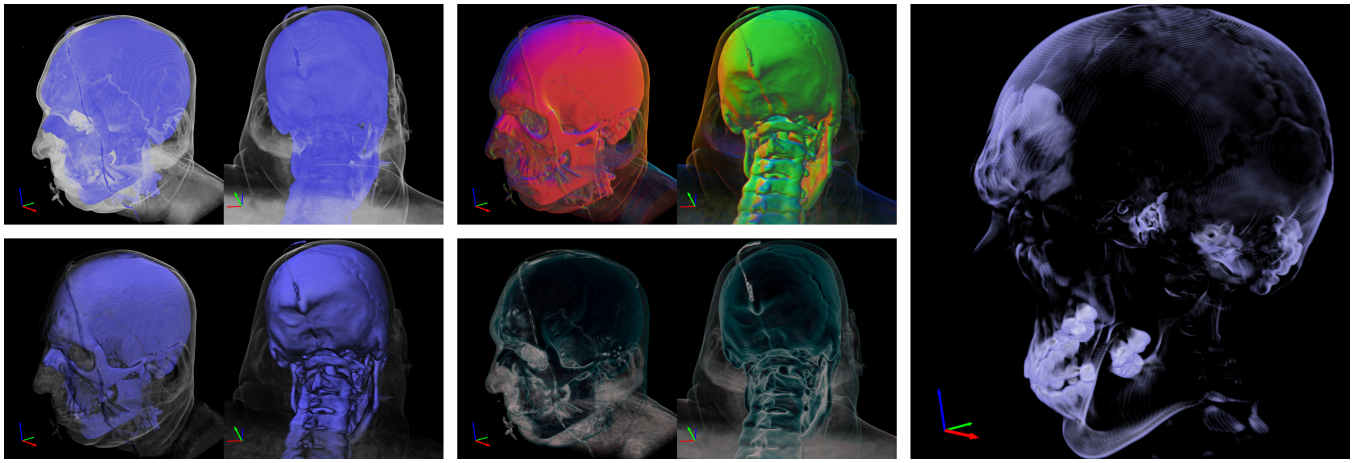


Fig. 4. Different renderings of a CT data set. Top-left: Use of a basic 1D transfer function. Bottom-left: Additional diffuse lighting. Top-centre: Gradient shading that shows the direction of the gradient vectors. Bottom-centre: Artistic shading that enhances the silhouette of rendered structures. Right: 2D transfer function using the scalar data value and gradient magnitude.

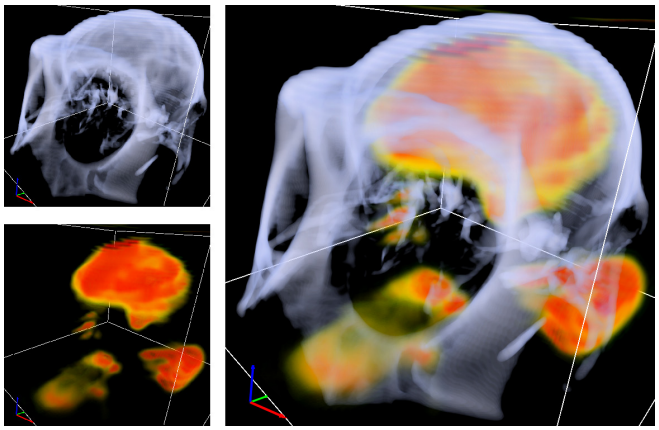


Fig. 5. Combined rendering of a CT (top-left) and a PET (bottom-left) data set of a monkey.

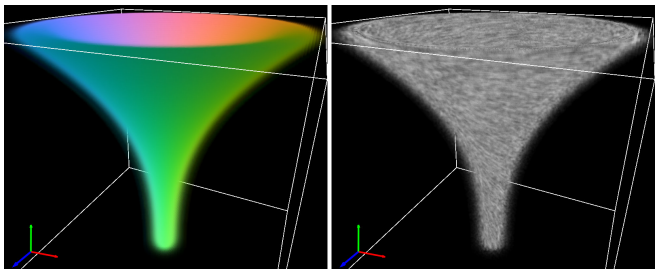


Fig. 6. Renderings of a 3D vector field. Left: Color-encoded procedural vector field (normalized vectors are mapped into RGB range $[0, 1]^3$). Right: Interactive LIC rendering. The opacity is proportional to the vector length.

sualization '02. Washington, DC, USA: IEEE Computer Society, 2002, pp. 109–116.

[4] S. Röttger, S. Guthe, D. Weiskopf, T. Ertl, and W. Strasser, "Smart hardware-accelerated volume rendering," in *VISSYM '03: Proceedings of the symposium on Data visualisation 2003*. Aire-la-Ville, Switzerland: Eurographics Association, 2003, pp. 231–238.

[5] M. Levoy, "Efficient ray tracing of volume data," *ACM Trans. Graph.*, vol. 9, no. 3, pp. 245–261, 1990.

[6] P. Lacroute and M. Levoy, "Fast volume rendering using a shear-warp factorization of the viewing transformation," in *SIGGRAPH '94: Proceedings of the 21st annual conference on Computer graphics and interactive techniques*. New York, NY, USA: ACM Press, 1994, pp. 451–458.

[7] NVIDIA® Corporation. (2005) Cg language specification. [Online]. URL: http://developer.download.nvidia.com/cg/Cg_1.5/1.5.0/0019/Cg_Specification.pdf

[8] N. I. o. H. National Library of Medicine. (2007) The visible human project®. [Online]. URL: http://www.nlm.nih.gov/research/visible/visible_human.html

[9] S. Röttger. (2006) The volume library. [Online]. URL: http://www9.informatik.uni-erlangen.de/External/vollib/

[10] U. S. o. M. Laboratory of Neuro Imaging. (2007) Monkey atlas. [Online]. URL: http://www.loni.ucla.edu/

[11] F. Manke, "A Modular GPU-based Direct Volume Renderer for Visualising Scalar and Multi-dimensional Data," Oct. 2007, CompSci 780 project report. [To be published].

USING GAME ENGINES AS A BASIS FOR VIRTUAL REALITY SURGERY SIMULATORS

*Stefan Marks**

ABSTRACT

The increase of complexity and costs of surgical training and the constant development of new surgical procedures has made virtual surgical training an increasingly important tool in medical education. Unfortunately, commercial tools are very expensive and mainly offer training only for individuals. Game engines offer unique advantages for the creation of inexpensive, highly interactive and collaborative environments.

This paper examines the suitability of currently available game engines for developing applications for medical education and simulated surgical training. We formally evaluate available game engines for stability, availability, the possibility of custom content creation and the interaction of multiple users via a network.

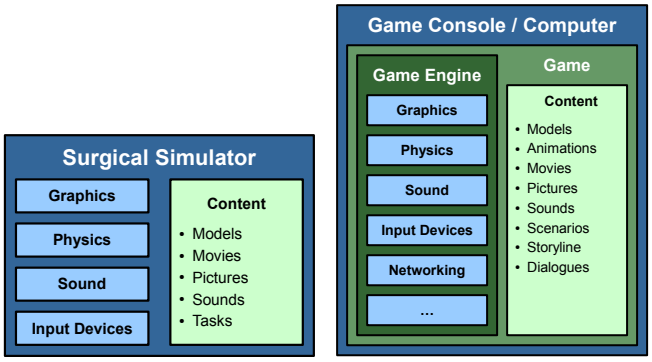
1. INTRODUCTION

Training tools using virtual reality (VR) simulations are becoming increasingly important in healthcare, in particular for applications which involve complex procedures and tasks, such as surgical training (a list of the most common simulators is given in [1]). The drivers for this include the improvement of quality in medical care and training, the need to reduce errors and costs [2], the requirement of quality standards and assessment methods for the performance of medical staff, and the desire to increase patient safety.

We identified four major drawbacks for commercial simulators:

1. Although teamwork and cooperation is considered an important dimension of simulation [3], most commercial products only train individuals.
2. Training focuses mainly on technical aspects, which only form a part of the total range of skills of a well trained and experienced surgeon [4].
3. Simulators are expensive, mainly due to the high-end hardware and specialised input and output devices (e.g. for force feedback).
4. For every newly developed simulator, the vendors "reinvent the wheel" constantly, because all simulators require at least the basic components depicted in figure 1(a).

There have been attempts to create extensible frameworks for building surgical simulators upon (SPRING [5], GiPSi [6], SOFA [7], ESQUi [8], [9]). They all incorporate the above mentioned components and a variety of mathematical models for the physical simulation and interaction. But except for SPRING (ironically the oldest project in the list) they all lack the capability of networking with other simulators to build collaborative scenarios. Even worse, sound support is not built into one of them.



(a) Surgical Simulator (b) Game Engine

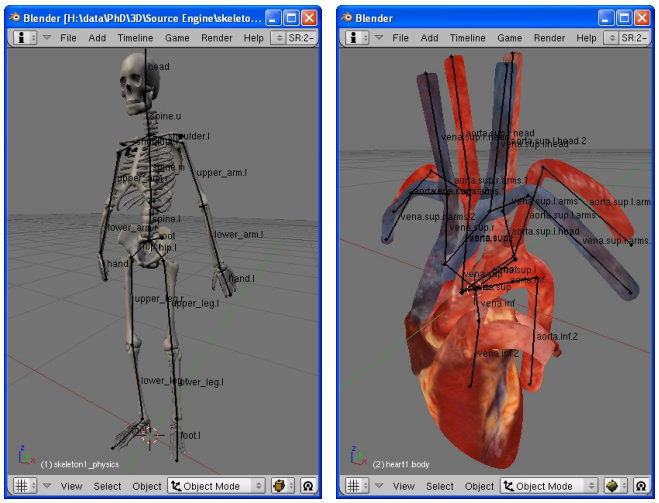
Fig. 1. Functional components of a surgical simulator and a game engine.

Modern game engines are structured similar to surgical simulators (see figure 1(b)) and also incorporate the networking and sound component. Game engines are well tested, robust and usually available at low costs.

The use of games or game engines for medical education is a little explored research subject with many areas still undiscovered. Projects like the "Serious Game Initiative" [10] focus on offering help to "organize and accelerate the adoption of computer games for a variety of challenges facing the world today." The subproject "Games for Health" focuses mainly on games used in various health care sectors.

Previous authors have so far concentrated on applications where the game content was about learning facts, rather than tasks and procedures. For example, Wünsche et al. have examined how game engines can be used for visualising medical datasets [11], and Mackenzie et al. utilise a game engine for anatomical education [12]. However, so far nobody has focused on the cooperative aspect in the simulation of complex medical procedures.

*Division for Biomedical Imaging and Visualization, Department of Computer Science



(a) Human Skeleton (b) Human Heart

Fig. 2. Custom medical models in the 3D editor. The black lines show the "bone" structure used to allow movement of parts of the model.

2. GOALS

The goal of our work is the evaluation of commercially available game engines for their suitability as a basis for surgical simulators.

To achieve this, we have to identify the necessary technological prerequisites for surgical simulation, select the most suitable engines for further evaluation and explore their capabilities by constructing test scenarios.

3. METHODOLOGY

We started our selection of suitable game engines with an evaluation of an internet game engine database [13]. We disregarded engines still in an early development state, those that were not developed or maintained any more, engines without sound or other essential components, and engines without inbuilt means of creating new game environments (maps).

After the reduction of the formerly 278 engines (as of June 2007) by this selection process, we chose from the remaining engines those which were inexpensive and in our opinion most popular and widely distributed:

- Unreal Engine 2 [14]
- id Tech 4 [15]
- Source Engine [16]

A first step in our evaluation process was the creation of multiplayer maps with simple physical objects to test the basic physical interaction with several users.

In a second step, we created mass-spring systems, and imported articulated medical models (human skeleton and heart, see figure 2) into the test maps to further evaluate the stability and precision of the physical simulation, as well as the suitability for more complicated physical models.

(a) View of the user on the server. (b) View of user on the client.

Fig. 3. The Unreal Engine 2 is unable to synchronise the position and the state of the skeleton model on the server and the client.

Fig. 4. Manipulation of the mass-spring model in the multiplayer map.

4. RESULTS

The first evaluation step resulted in the discarding of the Unreal Engine 2 and the id Tech 4 engine due to major restrictions in their physical simulation. Despite both engines were capable of simulating physical objects in *single* user mode, they were not able to reliably represent the state and movements of the objects simultaneously for all connected users in *multi* user scenarios (see figure 3 for an example).

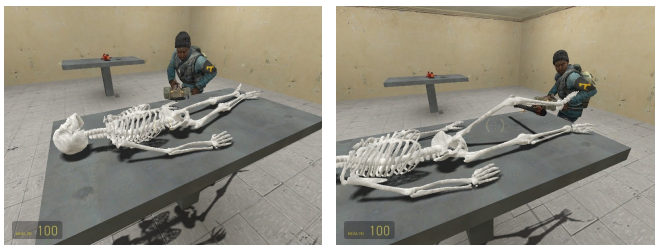
Only the Source Engine was capable of simulating simple physical objects and the cooperative interaction with them by multiple users over the network, e.g. pushing, pulling, rotating. We thus continued the evaluation with only this engine.

The more complicated models were also simulated without difficulty. A 4x4 mass-spring model was simulated without signs of instability and could be moved and deformed (see figure 4). In addition, we utilised the extensive material system of the engine to create a custom texture of a rough, shiny surface to give the mass-spring system an organic look.

The skeleton model reacts realistically to applied forces (see figure 5). Both users can move the legs, arms or other parts of the skeleton with the other user directly being able to see the result of that operation.

The vessels of the heart model can be moved and bent with the scalpel and deform in an intuitive manner (see fig-

(a) View of the acting user.



(b) View of the observing user.

Fig. 5. Manipulation of the skeleton model in the multiplayer map.

(a) View of the acting user.

(b) View of the observing user.

Fig. 6. Manipulation of the heart model in the multiplayer map.

ure 6). We discovered an instability in the physical simulation when the heart model is moved too quickly. It starts to oscillate in a chaotic manner and can only be stopped when its degrees of freedom get restricted, e.g. by pushing it into a corner or onto the table.

5. CONCLUSION

Our results show that, in general, game engines can be used for collaborative virtual reality surgical simulation, although the physical simulation capabilities do not yet allow for the complicated simulation of, e.g., soft bodies like organs, and cutting and suturing tissue.

Mass-spring models can act as a basic model for soft bodies, and their simulation is stable and fast enough for real-time interactivity. Articulated structures built with the help of bones (see figure 2) allow for a certain amount of deformability, though only restricted to rotating, bending and twisting. Other deformations like stretching or pressing are not possible.

Nevertheless, game engines could compensate this lack of simulation features with their means of playing back predefined animations. These could be put together to form a scenario where actions have to be triggered by the users by pointing on the right locations and then choosing from a range of optional actions. Artificial intelligence, already built into game engines, could enhance the immersion by controlling additional characters that react to the users actions, e.g. giving advise or creating additional stress by distraction.

6. FUTURE WORK

In our upcoming research, we will investigate into the extensibility of game engines for the incorporation of enhanced simulation models, e.g. [17, 18, 19].

We also will keep an eye on next generation game engines like CryEngine 2 [20] or Unreal Engine 3 [21], to see if their physical capabilities have improved and allow for new simulation features.

Furthermore, We will evaluate other aspects of modern game engines that could be used to enhance the educational impact, e.g. the use of pre-animated models, or computer controlled characters that react to the users actions and decisions.

7. ACKNOWLEDGEMENTS

We would like to thank the company Go Virtual Medical Ltd. (`http://www.govirtualmedical.com`) for supplying us with the 3D model and the texture of the heart.

8. REFERENCES

[1] B. Dunkin, G. L. Adrales, K. Apelgren, and J. D. Mellinger, "Surgical simulation: a current review," *Surgical Endoscopy*, vol. 21, no. 3, pp. 357–366, Mar. 2007.

[2] L. T. Kohn, J. M. Corrigan, and M. S. Donaldson, Eds., *To Err is Human: Building a Safer Health System*. Washington, DC, USA: National Academy Press, Nov. 2000. [Online]. Available: http://www.nap.edu/catalog.php?record_id=9728

[3] D. M. Gaba, "The future vision of simulation in health care," *Quality and Safety in Health Care*, vol. 13, no. Suppl 1, pp. i2–i10, Oct. 2004.

[4] F. C. Spencer, "Observations on the teaching of operative technique," *Bulletin of the American College of Surgeons*, vol. 3, pp. 3–6, 1983.

[5] K. Montgomery, C. Bruyns, J. Brown, S. Sorkin, F. Mazzella, G. Thonier, A. Tellier, B. Lerman, and A. Menon, "Spring: A General Framework for Collaborative, Real-time Surgical Simulation," *Studies in Health Technology and Informatics*, vol. 85, pp. 296–303, 2002.

[6] M. C. Çavuşoğlu, T. G. Göktekin, and F. Tendick, "GiPSi: A Framework for Open Source/Open Architecture Software Development for Organ Level Surgical Simulation," *IEEE Transactions on Information Technology in Biomedicine*, vol. 10, no. 2, pp. 312–322, Apr. 2006.

[7] J. Allard, S. Cotin, F. Faure, P.-J. Bensoussan, F. Poyer, C. Duriez, H. Delingette, and L. Grisoni, "SOFA – an Open Source Framework for Medical Simulation," in *Medicine Meets Virtual Reality (MMVR 15)*, Long Beach, USA, February 2007.

[8] M. A. Rodriguez-Florido, N. Sánchez Escobar, R. Santana, and J. Ruiz-Alzola, "An Open Source Framework for Surgical Simulation," *Insight Journal*, Jul. 2006. [Online]. Available: http://hdl.handle.net/1926/219

[9] S. Tuchschmid, M. Grassi, D. Bachofen, P. Früh, M. Thaler, G. Székely, and M. Harders, "A Flexible Framework for Highly-Modular Surgical Simulation Systems," in *Biomedical Simulation: Third International Symposium, ISBMS 2006, Zurich, Switzerland, July 10-11, 2006*, ser. Lecture Notes in Computer Science, vol. 4072. Heidelberg: Springer Berlin, Jul. 2006, pp. 84–92.

[10] Serious Games Initiative. (2007) Serious Games Initiative. [Online]. Available: http://www.seriousgames.org

[11] B. C. Wünsche, B. Kot, A. Gits, R. Amor, J. Hosking, and J. Grundy, "A Framework for Game Engine Based Visualisations," in *Proceedings of Image and Vision Computing New Zealand 2005*, Nov. 2005. [Online]. Available: http://www.cs.auckland.ac.nz/~burkhard/Publications/IVCNZ05_WuenscheKotEtAl.pdf

[12] J. Mackenzie, G. Baily, M. Nitsche, and J. Rashbass, "Gaming Technologies for Anatomy Education," Online, May 2003. [Online]. Available: http://www.virtools.com/news/pdf/2004/CARET.pdf

[13] DevMaster.net. (2007) 3D Game Engines Database. [Online]. Available: http://www.devmaster.net/engines/

[14] Epic Games. (2004) Unreal Engine 2. [Online]. Available: http://www.unrealtechnology.com/html/technology/ue2.shtml

[15] Wikipedia. (2007) id Tech 4 — Wikipedia, The Free Encyclopedia. [Online]. Available: http://en.wikipedia.org/wiki/Doom_3_engine

[16] Valve Corporation. (2004) Valve Source Engine Features. [Online]. Available: http://www.valvesoftware.com/sourcelicense/enginefeatures.htm

[17] A. Henriques, B. Wünsche, and S. Marks, "An investigation of meshless deformation for fast soft tissue simulation in virtual surgery applications," *International Journal of Computer Assisted Radiology and Surgery*, vol. 2, no. Suppl 1, pp. S169–S171, June 2007.

[18] M. Müller, B. Heidelberger, M. Teschner, and M. Gross, "Meshless deformations based on shape matching," *ACM Transactions on Graphics*, vol. 24, no. 3, pp. 471–478, Jul. 2005.

[19] A. R. Rivers and D. L. James, "FastLSM: Fast Lattice Shape Matching for Robust Real-Time Deformation," in *SIGGRAPH '07: ACM SIGGRAPH 2007 papers*, vol. 26, no. 3. New York, NY, USA: ACM Press, Jul. 2007, p. 82.

[20] Crytek. (2002) CryEngine 2 Specifications. [Online]. Available: http://www.crytek.com/technology/index.php?sx=eng2

[21] Epic Games. (2006) Unreal Engine 3. [Online]. Available: http://www.unrealtechnology.com/html/technology/ue30.shtml

SURFACE MANIPULATION USING A PAPER SCULPTURE METAPHOR

Glenn McCord, Beryl Plimmer

ABSTRACT

The creation of 3D computer models is an essential tool for many applications in science, engineering and arts and is frequently performed by untrained users. However, creating an intuitive mapping between 2D input and 3D models is a non-trivial task and is reflected in the difficulty novices have in using current 3D modelling software. Using metaphors of paper sculpture and pen sketching, our gesture based modelling tool simplifies this interaction mapping. More intuitive object manipulation means that an otherwise complex model can be rapidly created by an inexperienced, non-artistic user. To demonstrate this, we have chosen to model orchid flowers as they offer considerable challenges to the artist due to their complexity of shape and detail, especially the petal surfaces which vary a great deal in curvature.

1. INTRODUCTION

Traditional 3D modeling applications offer tools powerful enough to model a diverse range of creations but, unfortunately, many potential users can be overwhelmed by the enormous flexibility associated with them. One of the difficulties of these tools for novice users is that they are not based on any real world metaphor. Pencil and paper sketching, for example, is one of the most simplest yet effective ways to exercise some artistry, yet few modelling tools support digital pens (stylus) to any significant degree. Other metaphors, such as paper sculpting, can provide an interaction that makes it easier for users to predict the results of an action. We are exploring a blend of paper sculpture and sketching (where sketched lines represent paper cutouts) using a stylus, as an aid to novice 3D modelling interaction.

The proposed interface combining sketching and paper sculpting has the goal of easing the transition from the initial conceptual design into the final 3D model. This proposal is supported by two observations: firstly, many users find it hard to create 3D shapes which correspond to multiple 2D views and secondly, they find it difficult to interact with the complex and powerful mathematical surface descriptions used in traditional 3D modelling tools.

2D sketching is a quick, intuitive and easy way to indicate 3D shapes. Using this input and mapping it automatically into a 3D shape will help users who do not have sufficient artistic skills or find it difficult to mentally conceptualise 3D shapes. In order to provide a similar intuitive way for modifying the resulting 3D shape we extend the metaphor of sketching on paper and allow the user to interact with it as if sculpting paper. This is a natural process children perform from an early age and it facilitates the mental transition from a 2D to a 3D object since the paper used in this metaphor is a 2D object.

A mixture of drawing and sculpting metaphors allows the user to intuitively interact with the model because they subconsciously predict the effect their actions would have on the model based on their real world experiences. To realise this idea we have chosen to model orchid flowers as they offer great challenges to the artist due to their complexity of shape and detail, especially the petal surfaces which vary significantly in curvature.

2. RELATED WORK

Our orchid modeller draws on work from three main areas of research: sketch based modelling, flower modelling and surface deformation.

Sketch-based tools have been explored for a number of 3D modelling domains such as transformation from sketch to structured CAD projects. SKETCH [1] was an early research project that turned a conceptualised sketch into a digitised 3D scene. It exploits the ease of design afforded by sketching and the ability to change viewing angles with the 3D digital medium. 3D primitives are constructed with basic pen strokes which are then extended to basic 3D objects. Complex objects are constructed using a combination of primitives.

A common strategy for creating free form 3D objects from sketch input is to create simple objects and then either combine or deform them into other shapes. Igarishi's Teddy application created a 3D object by inflating 2D sketches based on the width of the 2D object [2]. With a combination of cutting gestures and combining objects together, it is possible to create complex, inflated (blobby) shapes. Further research projects using similar inflation metaphors join objects smoothly together and afford shape alterations by re-sketching parts of the silhouette [3] or by inferring 3D geometry by interpreting overlapping sketch lines [4].

Other authors [5], [6] have shown that complex 3D objects can be edited using stylus strokes that retrace an object's silhouette. The modification of a model's silhouette subsequently rescales it so that it remaps itself to the new silhouette.

An algorithmic approach to modelling plants is suggested by Prusinkiewicz and Lindenmayer whereby plant structures are constructed using rule based logic [7]. This abstract, bot-

tom up is, however, non-intuitive and difficult to control without extensive experience, so alternative sketch-based modelling of plants has been explored [8], [9]. Constraints can help with automatically creating a 3D structure, such as the assumption that branches seek to be as far away from their neighbour branches as possible [10]. Ijiri et. al. have shown that an effective way of creating a realistic flower is to sketch and then edit each individual flower component (petals, flower head etc), and then combine them together to form the complete flower model [11].

In order to make the flower modelling process more an artistic exercise, it has also been shown that a user can sketch a plant in its entirety, and then have each of its sketched components replaced with 3D equivalents [12].

Although petal like surfaces can be created from the Teddy 'blobs' by creating the blob and then cutting it like a potato chip, it deviates too far from what would be intuitive to a user. The flower modellers by Ijiri et. al. offer a much better alternative but their petals are restricted to a silhouette that doesn't form large concavities; ideally a user can sketch a petal of arbitrary shape. Another limitation is that petal curvature can only be altered by a series of modifying strokes that displace the vertices. We believe that there are more intuitive ways of modifying the curvature of petal-like surfaces.

3. OUR APPROACH

Paper is thin, which makes it an ideal metaphor for 3D modelling of objects that consist of thin surfaces such as flowers. Our approach uses metaphors of paper sculpture techniques whereby the user's sketch is a paper cut out that gets sculpted by folding, crimpling and indenting it.

Paper is a widely used artistic medium, not just because of its prevalence but also because of its flexibility as a modelling medium. One of the most well known paper crafts is origami, but there is more to paper sculpting than just folding hard edges.

Paper can be cut, torn either with or against the grain, creased along either a straight or curved line, coiled/rolled, cut to form textured patterns by utilizing light sources, joined together using tabbing, layered in relief, crimped (forming curves by cutting the paper and then folding it in on itself), impressing the paper and by curling edges [13]. All of these can serve as metaphors for virtual modelling.

3.1. Interaction

Many of these aforementioned paper sculpting techniques can be applied to surface manipulation to facilitate predictable user interaction. The primary techniques are the ability to cut, curve and crease paper so we have explored how these metaphors can be used for manipulating surfaces on the computer.

Besides the inherent difficulties with managing 3D objects in a 2D space, there is also the problem of working with a single mouse cursor. We are essentially paper sculpting with one hand. With one hand, we have to take a piece of paper, cut it to shape and then sculpt it by adjusting its curvature. By blending the sketching and paper sculpture metaphors together, simple interaction is achieved.

Here is an example of how the user could create a petal of an orchid. First the user draws the outline of the petal. The region enclosed by the sketch can be interpreted as a flat object cut out of a sheet of paper. Immediately the software generates and displays possible fold lines. The user selects part of the cut out with the mouse cursor and drags/pulls at it. As a result, the selected subpart will fold about an axis formed depending on the geometry of the cut out.

Since mouse input only gives 2D coordinates we must find a way to specify movements orthogonal to the screen during folding. Since a fold backwards often leads to visibility problems due to hidden surfaces our applications always interprets mouse drags as a pulling operation out from the screen rather than deeper into it.

3.2. Geometry

The user requires an intuitive way to curve their original shape. There are two generalized ways of achieving this:

1. Define a sub part of a surface to be curved and then curve it about the rest of the surface.

2. Have the program infer areas that can be curved and then curve these areas about the rest of the surface.

Both of these ideas are perfectly valid for paper sculpture. We can achieve the first technique by creasing the paper, thus creating an artificial axis from which the two areas about this axis rotate about. Creasing and folding don't have to be restricted to a straight axis either.

The second technique takes advantage of a 2D geometric property that defines these foldable axes automatically. Halverston [14] determined that silhouettes are especially important in determining objects. This is made evident when children draw the most salient silhouette of objects such as animals. By taking such a silhouette, Marr and Nishihara [15] noted that the concave sections of objects define the subparts of an object. As can be seen in figure 3, it is these subparts that define the foldable areas. The axes about which they fold is defined by the path that joins one concave curve to the other.

These geometric properties can be applied to paper sculpture. By picking up the paper from one of the subparts of the object, the subpart then naturally curves about the axis defined by the concave points of the silhouette that define that subpart (figure 1).

The ability for the user to relate to this object subdivision and paper style curvature is the basis of this interaction. Using

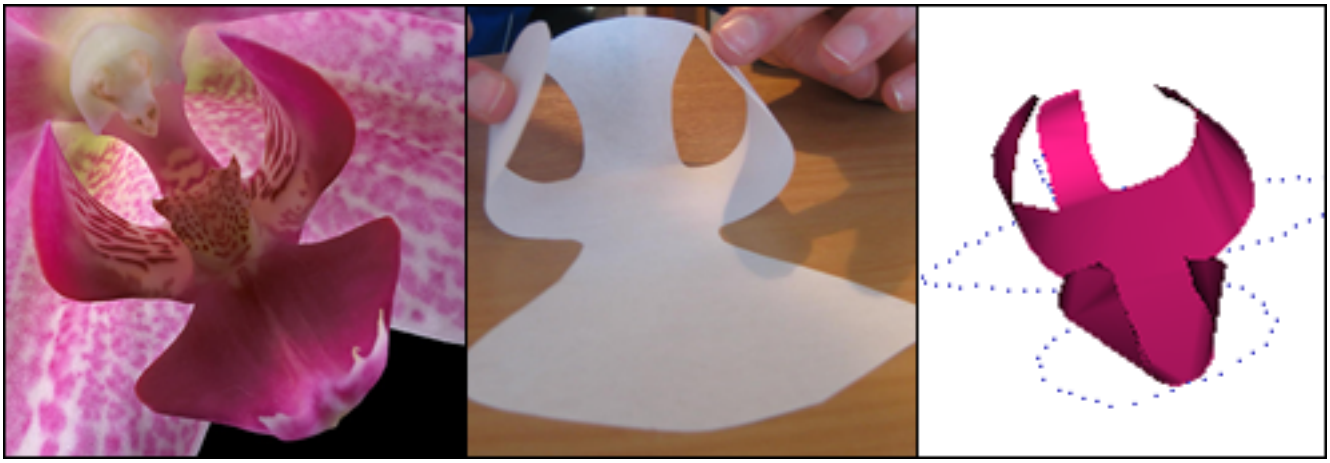


Fig. 1. This selection of images shows how the curvature of a real orhid petal (left) can be represented with paper (centre) and with a digital model that uses paper folding properties.

Fig. 2. The triangulation strategy used by "Teddy" defines different triangles as Terminal (T), Sleeve (S) or Junction (J) depending on how many sides are shared with the silhouette [2]

these geometric properties a predictable response will occur when the user interacts with the surfaces.

3.3. Implementation

Realisation of these subparts was achieved with Delaunay triangulation and some of the skeletonisation strategies used in the "Teddy" application [2]. The "Teddy" tool defines 3D shapes from 2D sketches by triangulating a closed sketch curve and computing a skeleton from it. The vertices of the sketch curve are then rotated around the skeleton resulting in a 3D shape whose projection is the original sketch. During the process of constructing the skeleton the triangles that make up the triangulation are defined as either a terminal, sleeve or junction triangle where each triangle type has either one, two or no shared edges to the silhouette respectively (see figure 2).

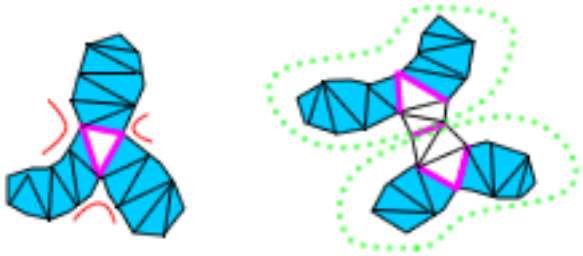
The edges that make up the junction triangle define the folding axes of a simple surface, thus isolating the sub parts of the overall surface (see figure 3). Some of the subparts become impractically small so the pruning method used in Teddy for eliminating insignificant parts of the skeleton was applied here.

More complex shapes create complications but the solutions still draw on the geometric properties:

- If the triangulated surface contains multiple junction triangles then consider a skeleton between pairs of junction triangles. The fold axis will be the shortest line to the silhouette that is orthogonal to the skeleton (figure 3(b)).
- Sometimes the pruning algorithm eliminates all the junction triangles. However, there is usually at least one pre-pruning junction triangle that is larger and shaped closer to an equilateral. The shortest edge of this triangle is the significant fold axis. Note that size and closeness to being equilateral is a property of most of the junction triangles.
- There is a special case when a surface is a consistent width and results in no junction triangles. If the surface has no concave areas then it won't be able to utlise the geometric properties discussed in section 3.2. However, a surface in the shape of a letter S or U does have such concavities and should be able to fold about certain points. Because such shapes don't exist in orchid flowers, it has been ignored for now.

3.4. Discussion

By adopting a paper sculpture metaphor, we believe we can facilitate the task of creating and manipulating the flat surfaces required by 3D flower models. However, there are still some points that should be addressed with regards to expanding the functionality and highlighting some of the issues with the interaction strategies.



(a) (b)

Fig. 3. 3(a) shows a junction triangle (purple) that stretches between concave sections (red), defining the subparts (blue). For more complex surfaces a fold axis between two junction triangles is idendified as the shortest distance across the surface. The green dashed lines of 3(b) represent encapsulating subparts.

The interaction becomes less clear when the user wishes to fold a large subpart that encapsulates smaller subparts (figure 3(b)). If the user selects one of the smaller subparts with the intention of folding the larger subpart, the application must be able to realise this. The only way to differentiate between the two areas is by the overlapping and non-overlapping regions. Expecting the user to select the non overlapping region in order to get the desired result may be optimistic. Another possible solution is to have the larger encapsulating subpart begin to fold once the smaller subpart has been folded to some critical angle.

Real orchid petals can form a curvature that is very difficult to sculpt with paper unless one was to 'collapse' the paper by crumpling it or using multiple tiny zigzag-like folds. Modelling such orchid surfaces would therefore break the paper scultping metaphor because simple paper folds maintain surface area.

We have implemented the cutting and folding metaphors but there are a multitude of other paper sculpting techniques that can enrich the user interaction possibilities. Our next step is to conduct usability testing in order to ascertain the effectiveness of the metaphor. The eventual goal is to utilize these techniques for the rapid creation of orchid flowers, as these represent a good example of high curvature surfaces.

4. CONCLUSION

Paper sculpture is a promising metaphor to assist in the creation and manipulation of complex surfaces such as those used in flower modelling. By utilizing geometric properties and the sculptural qualities of paper, it is possible to make a more intuitive 2D to 3D mapping. Traditional 3D modelling tools are complex so such techniques go some way to assist both novice and expert users to rapidly create complex models.

5. REFERENCES

[1] Robert C. Zeleznik, Kenneth P. Herndon, and John F. Hughes, "Sketch: an interface for sketching 3d scenes," in *SIGGRAPH '06: ACM SIGGRAPH 2006 Courses*, New York, NY, USA, 2006, p. 9, ACM Press.

[2] Takeo Igarashi, Satoshi Matsuoka, and Hidehiko Tanaka, "Teddy: a sketching interface for 3d freeform design," in *SIGGRAPH '99: Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, New York, NY, USA, 1999, pp. 409–416, ACM Press/Addison-Wesley Publishing Co.

[3] O. Karpenko, J. Hughes, and R. Raskar, "Free-form sketching with variational implicit surfaces," *Computer Graphics Forum*, vol. 21, no. 3, pp. 585–594, 2002.

[4] Olga A. Karpenko and John F. Hughes, "Smoothsketch: 3d free-form shapes from complex sketches," *ACM Transactions on Graphics*, vol. 25, no. 3, pp. 589–598, July 2006.

[5] Andrew Nealen, Olga Sorkine, Marc Alexa, and Daniel Cohen-Or, "A sketch-based interface for detail-preserving mesh editing," in *SIGGRAPH '05: ACM SIGGRAPH 2005 Papers*, New York, NY, USA, 2005, pp. 1142–1147, ACM Press.

[6] Jing Hua and Hong Qin, "Free-form deformations via sketching and manipulating scalar fields," in *SM '03: Proceedings of the eighth ACM symposium on Solid modeling and applications*, New York, NY, USA, 2003, pp. 328–333, ACM Press.

[7] P. Prusinkiewicz and A. Lindenmayer, *The algorithmic beauty of plants*, Springer-Verlag, New York, 1990.

[8] Lars Mundermann, Peter MacMurchy, Juraj Pivovarov, and Przemyslaw Prusinkiewicz, "Modeling lobed leaves," *Computer Graphics International*, pp. 60–65, 2003.

[9] Fabricio Anastacio, Mario Costa Sousa, Faramarz Samavati, and Joaquim A. Jorge, "Modeling plant structures using concept sketches," in *NPAR '06: Proceedings of the 4th international symposium on Non-photorealistic animation and rendering*, New York, NY, USA, 2006, pp. 105–113, ACM Press.

[10] Makoto Okabe, Shigeru Owada, and Takeo Igarashi, "Interactive design of botanical trees using freehand sketches and example-based editing," in *SIGGRAPH '06: ACM SIGGRAPH 2006 Courses*, New York, NY, USA, 2006, p. 18, ACM Press.

[11] Takashi Ijiri, Shigeru Owada, Makoto Okabe, and Takeo Igarashi, "Floral diagrams and inflorescences: interactive flower modeling using botanical structural constraints," *ACM Trans. Graph.*, vol. 24, no. 3, pp. 720–726, 2005.

[12] T. Ijiri, S. Owada, and T. Igarashi, "Seamless integration of initial sketching and subsequent detail editing in flower modeling," *Computer Graphics Forum*, vol. 25, no. 3, pp. 617–624, 2005.

[13] Paul Jackson, *The Art and Craft of Paper Sculpture*, Apple Press, 1996, Quarto Publishing Plc.

[14] J. Halverston, "The first pictures, perceptual foundations of paleolithic art," *Perception*, vol. 21, pp. 389–404, 1992.

[15] D. Marr and H. K. Nishihara, "Representation and Recognition of the Spatial Organization of Three-Dimensional Images," *Proceedings of the Royal Society of London, Series B*, vol. 200, pp. 269–294, 1978.

RETHINKING THE GENETIC ALGORITHM

Cameron Skinner

ABSTRACT

The Genetic Algorithm (GA) promises to be able to solve many classes of problems, particularly those where we do not fully understand the structure of the optimal solution. Unfortunately, using a GA in practice requires much trial-and-error in order to determine which combination of operators, selection and other parameters will result in good solutions in a reasonable time. This paper describes a known weakness in the GA and suggests an alternative approach to designing the GA to eliminate this weakness.

1. INTRODUCTION

The Genetic Algorithm (GA) [1, 2] is a machine learning algorithm inspired by the processes of natural selection and biological evolution. To solve a given problem the GA works by maintaining a *population* of candidate solutions. These solutions are evaluated to determine their *fitness* and some subset of the population is *selected* for reproduction. A new population is generated by applying various operators to the set of selected parents, such as *crossover*, *mutation* and *cloning*. The process is repeated for some number of generations and the best solution found is returned to the user.

The GA's power comes from the fact that it maintains a population of solutions rather than a single "best-so-far" solution. If the problem contains seperable sub-problems then these sub-problems can be solved in different individuals and then combined, via crossover, into a single, highly-fit individual. Unfortunately, the interactions between different components of the GA are complex and subtle making it difficult to predict how the algorithm will perform given a particular problem, operator set and fitness function.

This paper describes a new way of thinking about the GA that moves away from the traditional, biologically inspired ideas of crossover and mutation in favour of concepts that directly relate to how the GA behaves in practice. It is suggested that this change of paradigm may lead to a better understanding of the GA and to variations of the algorithm that require fewer parameters and therefore less tuning.

Section 2 gives a brief overview of the *Canonical Genetic Algorithm* (CGA), followed by a description of the classical GA model known as the *building block hypothesis* in Section 3 and a demonstration of a flaw in the standard GA in Section 4. Section 5 describes an alternative approach to GA implementation and proposes a solution to one aspect of the new model. Preliminary experimental results are discussed in Section 6 and Section 7 concludes with a number of possible future research directions.

```
let P = random_population();
while (not finished) do
    Evaluate fitness of each individual P_i in P
    let P' = ∅
    while |P'| < |P| do
        Select two parents A and B from P based on fitness
        With probability c perform crossover on A and B
                to produce children C and D
                otherwise let C=A and D=B
        With probability m perform mutation on C and D
        Add C and D to P'
    let P = P'
```

Fig. 1. The standard GA

2. THE CANONICAL GENETIC ALGORITHM

The standard pseudocode for a GA is shown in Figure 1.

The algorithm starts with a randomly generated population and repeatedly evaluates fitness, selects suitable parents and applies genetic operators to those parents to produce a new generation of individuals. The genetic operators are usually crossover and mutation. These are designed to combine promising candidate solutions and to perform local search respectively. Crossover generally applies to two parents[1] whereas mutation is a unary operation.

Crossover is generally implemented by slicing the two strings at a *crossover point* and combining the first part of the first parent with the second part of the second parent, and vice versa. For example, Figure 2 shows what happens if we cross at the third index into the string. There are many variations on this basic crossover operator, including the use of multiple crossover points, but the general idea remains the same: crossover is designed to take chunks of each parent in the hope that the combination of these chunks will produce a highly fit child.

Mutation is generally a simple bit-flip operation with each bit mutated independently with some low probability m, usually $m < 0.1$. Mutation is intended to provide *genetic diversity*, i.e. to prevent the situation where every individual in the

[1] Although there is no reason why it cannot be applied to multiple parents.

Parent 1	**1**	**1**	**0**	1	0	0	1	0
Parent 2	1	0	0	**0**	**1**	**1**	**1**	**1**
Child 1	1	1	0	0	1	1	1	1
Child 2	1	0	0	1	0	0	1	0

Fig. 2. An example of crossover. Bold entries indicate the values that are inherited by child 1.

population has the same value for some gene, and to allow for local search. Mutation also ensures there is some probability (albeit an exponentially small one) of escaping local optima.

3. BUILDING BLOCKS

The *building block hypothesis* [2] describes how the GA is supposed to work: suppose we have a problem defined over fixed-length binary strings [2]. We can define a *schema* as a fixed-length binary string with some defined values and some "don't care" bits. A fully specified string is then defined to be an *instance* of a schema if the schema's defined bits exactly match the candidate string's values.

For example, the string 1101 is an instance of the schemas *10*, ***1 and **** (amongst others). The building block hypothesis states that the GA will identify short, highly fit schemata and combine them to form longer, highly fit schemata.

The classic problem devised to illustrate this hypothesis is called the *Royal Road* problem [3, 4]. The Royal Road problem consists of n blocks each of b bits. These blocks are concatenated to form a string of length $n * b$ and each block defines a single schema with b ones. Individuals in the population are binary strings of length $n * b$ and they get one point of fitness for each schema that they are an instance of.

Figure 3 shows the schemata defined by the 4x4-bit Royal Road problem.

s_1	1111	****	****	****
s_2	****	1111	****	****
s_3	****	****	1111	****
s_4	****	****	****	1111

Fig. 3. Schemata defined by the 4x4-bit Royal Road problem

On this problem the string 1101 0000 1111 1111 would have a fitness of 2 as it is an instance of two schemata (s_3 and s_4). The optimal string is 1111 1111 1111 1111.

4. PROBLEMS WITH THE STANDARD GA

Figure 4 shows a typical plot of fitness vs generation number for a GA running on the 8x8-bit Royal Road problem (average over 100 runs). We can see that initially the best fitness grows very quickly, followed by a long tail where there is very little fitness gain. What this means is that in the initial population each individual has at most one building block, but they are combined within a few generations to form individuals with several blocks correct. The average fitness slightly lags the best fitness due to the destructive effects of mutation.

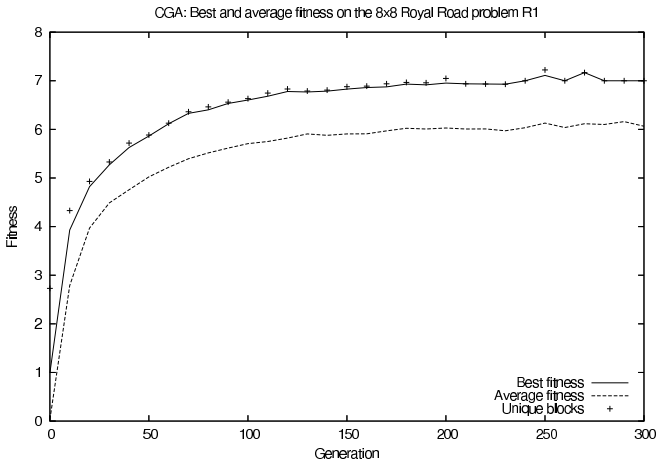


Fig. 4. Typical GA performance on the 8x8-bit Royal Road problem.

This result highlights a deficiency in the algorithm: after the initial rapid progress it spends a large amount of time achieving very little. Unless the population is sufficiently large to start with it can spend hundreds of generations without discovering any new building blocks.

Figure 4 shows, along with the best and average fitness in the population, the total number of distinct building blocks that exist anywhere in the population, not just in a single individual. We can see that after the first 10 generations or so that the number of blocks in the population exactly matches the best fitness, that is to say that the best individual in the population contains all known blocks.

Further investigation shows that the algorithm fails to proceed quickly because it lacks the ability to efficiently discover new building blocks and incorporate them into the population. This realisation leads to the following proposed alteration to the standard model of the genetic algorithm.

5. RETHINKING THE ROLES OF GENETIC OPERATORS

Much GA research revolves around debating whether mutation is more important than crossover, or which crossover scheme to use, and so on. It is proposed that instead of focussing on mutation and crossover, GA researchers should be looking at two abstract operators: *discovery* and *combination*.

A building block is said to have been *discovered* by a genetic operator if a child resulting from that operation has the block correct but none of the parents do. A set of blocks are

[2]This assumption is not as restrictive as it seems. We can easily encode many standard problems, such as the Travelling Salesman Problem, as fixed-length binary strings.

said to have been *combined* if the child has all those blocks correct and each block exists in exactly one parent. We can now examine the GA in terms of its ability to perform discovery and combination.

The standard GA fails because it is highly inefficient at performing discovery. In fact, it can be shown that both mutation and crossover are orders of magnitude less efficient at discovery than random search. It is generally very difficult to balance the explorative power of high mutation and crossover rates with the corresponding increase in building block destruction.

For this reason it is suggested that the standard GA pseudocode be modified to that shown in Figure 5. We have replaced crossover and mutation with combination and discovery, but this gives us a powerful advantage: our discovery operator can be highly destructive without impacting the performance of the rest of the GA. We also need to define "suitable" parents for combination. Intuitively we can consider this to mean parents that contain different sets of building blocks. If they contain exactly the same set of blocks then crossover can achieve nothing, so the parents are "unsuitable".

let P = random_population();
while (not finished) do
    Evaluate fitness of each individual P_i in P
    let $P' = \emptyset$
    while $|P'| < |P|$ do
        Select two parents A and B from P based on fitness
        If A and B are suitable perform combination to
                produce children C and D
        Otherwise perform discovery to replace A or B
                and perform combination on the replacement
                to obtain C and D
        Add C and D to P'
    let $P = P'$

Fig. 5. Modified GA pseudocode

6. EXPERIMENTS

In order to test whether this approach shows any promise a simple experiment was conducted. A standard GA was adapted to the new architecture, with the widely-used 2-point crossover operator used as the combination component. Rather than using mutation as the discovery operator (since mutation is known to be inefficient at discovery) random search was used to generate an estimate of the average fitness across the whole search space. The top few individuals from this search were kept in a *seed pool*. Application of the discovery operator simply randomly chooses an individual (with replacement) from the seed pool.

Parameter	**Value**
Number of blocks	8
Block size	8
Population style	Generational
Population size	100
Crossover rate	0.9
Crossover mode	2 point
Mutation rate	$\frac{1}{64}$
Selection	5 tournament
Seed probability (s)	$\frac{1}{3}$
Seed pool size	50
Presample size	1000

Table 1. Experiment parameters

Algorithm	Generations	
Canonical GA	147.32	(75.36)
Seed pool	81.17	(97.03)

Table 2. Results for the canonical GA and seed pool algorithm. Standard deviations are shown in parenthesis

Identifying when two parents are "suitable" for crossover is a difficult problem. In general, it is impossible to tell whether two individuals contain different sets of building blocks [3], so for this experiment a random process was used: parents are unsuitable with probability s.

Table 1 shows the parameters that were used for the experiment. Preliminary results using this new approach are encouraging. Table 2 shows the number of generations required to find a solution averaged over 100 repeats. There is nearly a two-fold performance improvement when the seed pool is used, and this improvement is statistically significant[5] even when the cost of performing the seed pool presampling step is taken into account.

Figure 6 shows the best and average fitness and the number of building blocks in the population for the new algorithm. We can see why it performs so much better than the canonical GA: the number of blocks that exist is almost always around 8, the maximum possible. This means that the proposed algorithm is maintaining instances of all building blocks all the time, allowing crossover to do combine these blocks and quickly find the optimal solution. There is still a long tail in the fitness plot so the algorithm can obviously be improved further.

Obviously these results only apply to the Royal Road problem and should be regarded purely as a proof of concept, but further encouraging results have been obtained from the Deceptive Trap and Heirarchical If-and-only-if problems[6].

[3]Recall that "suitable" parents are those that contain different sets of building blocks

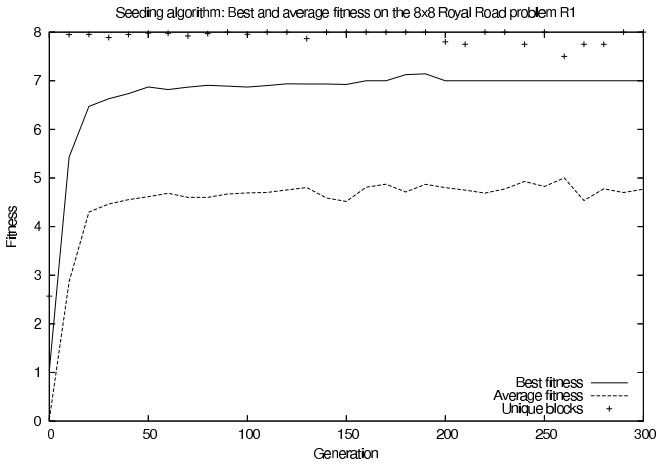


Fig. 6. Typical GA with seed pool performance on the 8x8-bit Royal Road problem.

7. CONCLUSIONS AND FUTURE WORK

The Genetic Algorithm has a fundamental design flaw: it is not suited to performing building block discovery during the run. An alternative way of approaching the design of the algorithm is proposed that explicitly splits the algorithm's discovery and combination aspects.

Preliminary experiments using this approach are highly encouraging, however there a number of difficulties that must be addressed before this approach can be used in general. Effective discovery and combination operators must be identified. Although random search has been suggested as a good discovery operator it is not yet clear which combination operators will be most effective on any given class of problems. The problem of detecting when parents are "suitable for combination" has not yet been addressed and only a naive, randomised process has been investigated to date.

Even with a naive implementation, however, this approach shows great promise and can result in a statistically significant performance improvement (in terms of number of fitness evaluations required to find the solution). This approach may lead to a deeper understanding of how the GA behaves and to improved performance across a variety of difficult real-world problem domains.

8. REFERENCES

[1] John H. Holland, *Adaptation in Natural and Artificial Systems*, The University of Michigan Press, 1975.

[2] David E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison-Wesley, 1989.

[3] Melanie Mitchell, Stephanie Forrest, and John H. Holland, "The royal road for genetic algorithms: Fitness landscapes and GA performance," in *Towards a Practice of Autonomous Systems: Proceedings of the First European Conference on Artificial Life, 1991*, Francisco J. Varela and Paul Bourgine, Eds. December 1992, pp. 245–254, MIT Press.

[4] Stephanie Forrest and Melanie Mitchell, "Relative building-block fitness and the building-block hypothesis," in *Foundations of Genetic Algorithms 2*, L. Darrell Whitley, Ed., pp. 109–126. Morgan Kaufmann, 1993.

[5] Cameron Skinner and Patricia Riddle, "Random search can outperform mutation," in *Proceedings of the IEEE Congress on Evolutionary Computation*, 2005.

[6] Richard A. Watson and Jordan B. Pollack, "Hierarchically consistent test problems for genetic algorithms," in *Proceedings of the Congress on Evolutionary Computation*, Peter J. Angeline, Zbyszek Michalewicz, Marc Schoenauer, Xin Yao, and Ali Zalzala, Eds. 1999, vol. 2, pp. 1406–1413, IEEE Press.