Automated Generation of Graphic Sketches by Example

Michelle X. Zhou **Min Chen**

IBM T. J. Watson Research Center
19 Skyline Drive
Hawthorne, NY 10532
Phone: (914)784-7000
{mzhou, minchen}@us.ibm.com

Abstract

Hand-crafting effective visual presentations is time-consuming and requires design skills. Here we present a case-based graphic sketch generation algorithm, which uses a database of existing graphic examples (cases) to automatically create a sketch of a presentation for a new user request. As the first case-based learning approach to graphics generation, our work offers three unique contributions. First, we augment a similarity metric with a set of adequacy evaluation criteria to retrieve a case that is most similar to the request and is also usable in sketch synthesis. To facilitate the retrieval of case fragments, we develop a systematic approach to case/request decomposition when a usable case cannot be found. Second, we improve case retrieval speed by organizing cases into hierarchical clusters based on their similarity distances and by using dynamically selected cluster representatives. Third, we develop a general case composition method to synthesize a new sketch from multiple retrieved cases. Furthermore, we have implemented our case-based sketch generation algorithm in a user-system cooperative graphics design system called IMPROVISE+, which helps users to generate creative and tailored presentations.

1 Introduction

Automated graphics generation systems promise to aid users in creating effective visual presentations (e.g., charts and diagrams) [Mackinlay, 1986; Zhou, 1999]. Upon a user request (e.g., displaying sales data), these systems directly provide users with the final presentation (e.g., a barchart). To better tailor a presentation to user preferences, we are building a user-system cooperative graphics generation system, called IMPROVISE+. IMPROVISE+ generates a presentation in two steps: sketch generation and sketch refinement. Here a *sketch* is an outline of a presentation without all visual details (Figure 1). A generated sketch is first presented to users for their feedback. Depending on the feedback, IMPROVISE+ may redesign the sketch or refine it to create a final illustration (e.g., setting the exact layout of Figure 1). By allowing users to critique a sketch *first*, IMPROVISE+ can save the cost of fine-tuning an undesirable design.

Our focus here is on a case-based learning approach to sketch generation. Given a user request, from a database of existing graphic examples (cases) our approach uses a similarity metric to retrieve the case that is most similar to the request. The retrieved case is then directly reused for or adapted to the new situation (e.g., new data). Instead of using a rule-based approach as most graphics generation systems do [Mackinlay, 1986; Zhou, 1999], our decision of using case-based learning is two-fold. First, it is difficult to hand-extract a complete and consistent set of graphics design rules, while existing graphic examples are abundant [Zhou et al., 2002a]. Second, case-based learning is efficient for sketch generation, where we focus on learning overall visual structures instead of precise visual arrangements (e.g., exact positions and scales) [1].

Although case-based learning has been applied to various design problems [Borner, 2001], it has never been applied to graphics design. As the first case-based learning

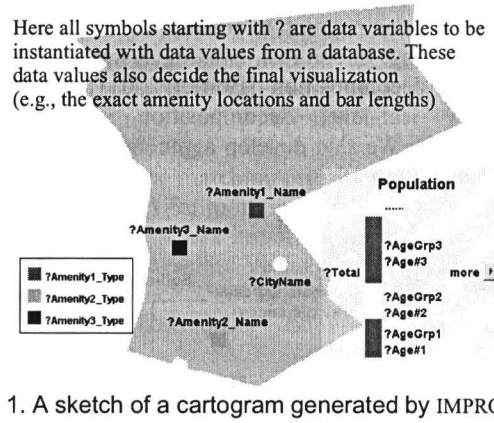


Figure 1. A sketch of a cartogram generated by IMPROVISE+.

1 Due to the complexity in graphics rendering, it is impractical to learn the values of visual parameters, such as scales and positions, from existing cases. We compute the parameters during the sketch refinement stage.

approach to graphics design, our work offers three unique contributions. 1) In case retrieval, we apply a set of adequacy evaluation criteria in addition to a similarity metric to ensure that the retrieved case is usable in sketch synthesis. To handle situations where a usable case may not be found, we decompose cases/requests into sub-cases/sub-requests to facilitate the retrieval of case fragments. 2) We improve case retrieval speed by organizing cases into hierarchical clusters based on their similarity distances and by using dynamically selected cluster representatives. 3) We synthesize a new sketch from multiple retrieved cases through case generalization and visual decoration inference (e.g., inferring decorations such as coordinate axes and legends).

Starting with a brief discussion of related work, we then present our case-based learning algorithm, highlighting the three unique features mentioned above. Finally we use a concrete example to demonstrate how a new sketch is created.

2 Related Work

Unlike rule-based graphics systems [Mackinlay, 1986; Andre and Rist, 1993; Chuah et al., 1997; Zhou, 1999], our work is the first to apply a general case-based learning technique to graphics design. Although one system, SAGE, has an example-based generation component, it only reuses examples created by its own rule engine [Chuah et al., 1997]. Compared to SAGE, IMPROVISE+ uses a much more fine-grained representation to capture the semantic and syntactic features of each graphic example [Zhou et al., 2002b]. As a result, without using any rules IMPROVISE+ can create new graphic sketches by directly learning from a wide variety of graphic examples.

Differing from a programming by demonstration system [Myers et al., 1994], where users must supply the desired examples, our approach uses the graphic examples stored in a database.

There are many case-based systems developed for other domains, e.g., engineering design [Sycara et al., 1992]. The closest to ours are known as case-based structure generation systems [Borner, 2001]. However we have gone beyond existing approaches to address specific challenges in graphics design. In particular, we support a systematic, multi-level case/request decomposition to achieve a more accurate case-request matching. In contrast, existing systems either ignore case/request decomposition [Borner, 2001] or simplify it (e.g., only leaf-level decomposition [Michelena and Sycara, 1994]). We also develop a general case composition method to synthesize a new sketch from multiple cases, while existing systems only allow limited case composition (e.g., combining only highly similar cases [Borner, 2001]).

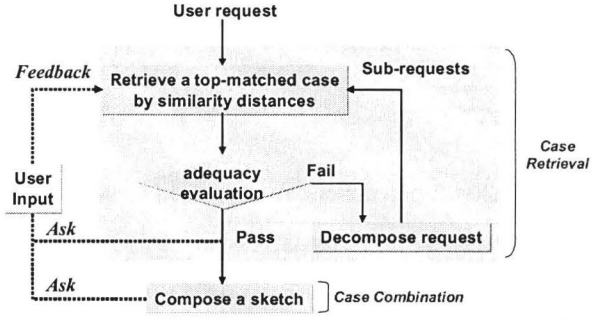


Figure 2. Flow diagram of our sketch generation algorithm.

3 Example-based Sketch Generation

Figure 2 gives an overview of our case-based sketch generation algorithm. Our algorithm uses a database of existing information graphics (cases) to suggest a visual design for a new user request. Here each stored case is described by its data content D and visual representation V. Each request is presented by specifying the data D' and its desired visualization V', which may be partially or not specified at all. Given such a request, our algorithm first uses a similarity metric to retrieve the top-matched case by computing the similarity distances between the request and existing cases. A *top-matched case* is the case that has the shortest similarity distance to the request. If the top-matched case fails our adequacy test, the current request is decomposed into a set of sub-requests. The whole process is recursively called to find the top-matched case for each sub-request. Depending on the retrieval result, a new sketch (a fully specified V') may be constructed from the visual encoding (V) of a single matched case or composed from multiple cases. Upon completion, the user is presented with a rendered graphic sketch (e.g., Figure 1). We also involve users at different design stages (Figure 2). For example, our studies have shown that users may express their preferences to retrieve more desired cases [Zhou et al., 2002b] or propose new visual compositions during sketch synthesis.

Before discussing our algorithm, we first briefly describe the representations of our cases and user requests.

3.1 Representation of Cases and User Requests

We employ a labeled graph structure to represent our cases and requests. Since we have described in detail how to model and represent various semantic and syntactic features of an existing graphic example previously [Zhou et al., 2002b], we summarize the representation here.

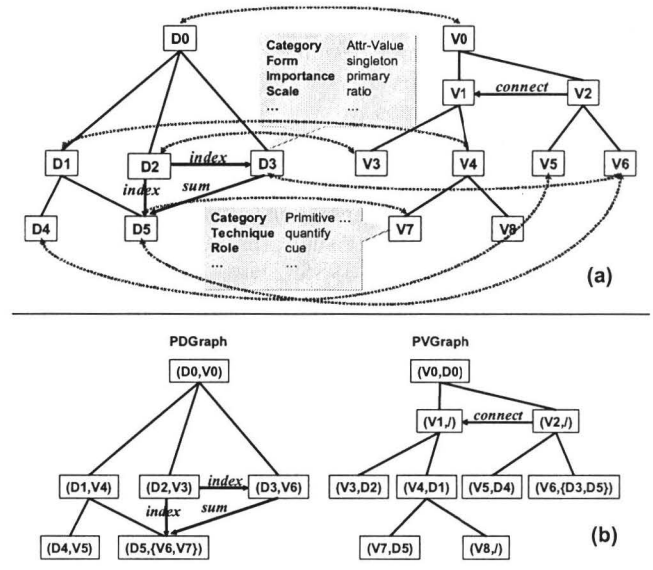


Figure 3. A labeled graph representation of a case.

Visual Database of Cases

Our case base contains an assortment of graphic designs collected from a wide variety of sources [Zhou et al., 2002a]. Using a labeled graph structure [Zhou et al., 2002b], each case is described in XML as a graph (Figure 3a), which expresses a complex mapping between a data hierarchy (D0–D5) and a visual hierarchy (V0–V8). Within each hierarchy, a data/visual node is described by a set of features (e.g., D3 and V7). There are two types of links in the graph: intra-hierarchy links for data/visual node relationships (e.g., D3 is indexed by D2 and all undirected edges implying parent-child relations), and inter-hierarchy links (red dotted lines) for data-visual mappings (e.g., D5 to V7). To facilitate case retrieval, we also index each case using two graphs (Figure 3b): PDGraph organizes data-visual mapping pairs along the data hierarchy, while PVGraph arranges the mapping pairs by the visual hierarchy. Note that a data/visual node may be mapped to multiple visual/data nodes (e.g., D5 and V6) or none (denoted by "/"). Our bi-graph indexing not only preserves all the information captured in the original complex graph (Figure 3a), but also partitions one complex graph into two simpler sub-graphs (Figure 3b), which can greatly simplify the similarity measuring process.

When loading all cases from the database to memory, IMPROVISE+ parses the XML document of each case and automatically builds all case indices (PDGraphs and PVGraphs).

User Request

A user request submitted in XML format is also described as a graph similar to the case representation. Figure 4 outlines a request for presenting the information of a city. Specifically, it asks to display the relevant county (CountyBoundary), basic information of the city (Name and Location), the city gazetteer (Population, AgeInfo for 6 age groups, and an arrow button indicating MoreInfo is available upon request), and 3 city amenities (Name, Location, and Type) such as community golf courses and swimming pools. Since users may not know every presentation aspect, the representation of a user request is often a partially specified graph with the majority of the visual elements left unspecified. For example, Figure 4 does not specify a visual encoding for any data nodes except MoreInfo.

Based on the characteristics of data relations, our algorithm automatically assigns *matching priorities* to different data relations in a user request to indicate that matching certain data relations well (e.g., index in Figure 5) is more important than matching others (e.g., has-a). As described below, matching priorities aid us in evaluating the usability of a retrieved case and in selecting visual candidates. Currently, we assign priorities by relation type. For example, a higher priority is given to presentation relations like index than to semantic relations like has-a.

3.2 Case Retrieval

The success of an interactive case-based system like ours, depends largely on the quality and speed of the case retrieval process. To ensure retrieval quality, we augment similarity measuring with *adequacy evaluation* that tests whether a retrieved case is usable in sketch synthesis. If a usable case cannot be found, we decompose cases and requests into sub-cases and sub-requests to facilitate the retrieval of case fragments. In addition, we use a hierarchically structured case base to improve our retrieval speed.

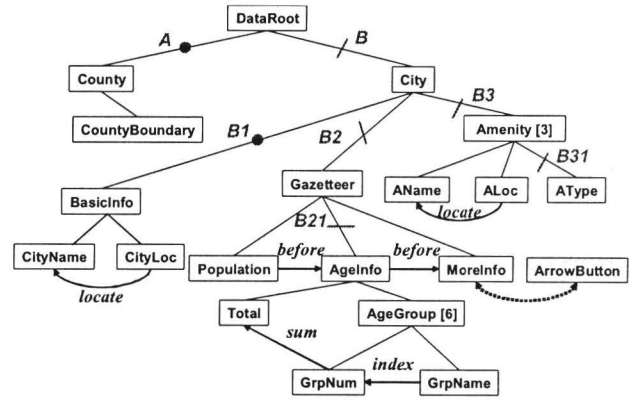


Figure 4. A sample user request.

Adequacy-Guided Retrieval

Using a quantitative similarity metric that we have developed [Zhou et al., 2002b], our algorithm first retrieves the top-matched case for a request. As a result, each data node and data relation in the request are associated with a computed similarity distance in the range of [0, 1]. Since our similarity model stresses the overall structural similarity between the request and the existing examples, a data node D^r in the request may or may not acquire a match (D, V) from the top-matched case. Here D is the data that matches D^r, and V (the visual mapping of D) is a potential *visual candidate* for encoding D^r in the new sketch. If no match is found for D^r, the recorded distance is 1.0. However, the top-matched case may be inadequate for creating an effective sketch for the following three reasons. Accordingly, we formulate three adequacy criteria to evaluate whether a retrieved case is usable in sketch synthesis.

First, a top-matched case may produce a good overall match but poor or no matches for certain data nodes in the request. This implies that certain data or a sub-set of data expressed in the request may not be well visualized in the new sketch as they could be. To ensure a good match for every data node (i.e., a short similarity distance), our first criterion requires that the similarity distance for every data node of a request be below a threshold. After conducting a series of case retrieval experiments, we currently set the threshold to 0.3 on a [0, 1] scale, which proves to be a good indicator for creating a quality new sketch.

Second, suppose that a top-matched case passes the above criterion. By the matching priorities set in the request, however the distances for more important data relations may be larger than those of less important ones. Since matching priorities are used to select visual candidates during case composition, using such a matching result may alter the original intention of the request and produce an undesired

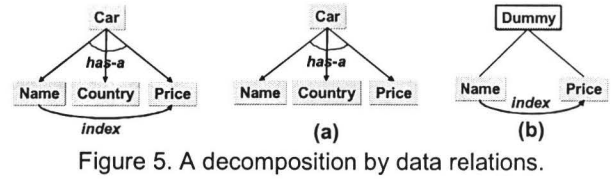


Figure 5. A decomposition by data relations.

sketch. Therefore our second criterion states that for every data relation in a request, the higher its matching priority is, the shorter its associated similarity distance must be.

Third, even though a top-matched case passes the above two criteria, it may still not be adequate for synthesizing a new sketch. Our synthesis starts with the visual candidates of data leaf nodes in a request and composes the higher-level visual nodes from bottom up using the lower-level visual nodes. However our top-down graph matching is a partial matching, which may not guarantee that every data leaf node in a request acquire a visual candidate [Zhou et al., 2002b]. Our synthesis would then fail due to a lack of basic building blocks. Thus our third criterion requires that every data leaf node in a request obtain a visual candidate. It is not required to find visual candidates for intermediate data nodes from case retrieval, since theirs can be composed from those of their children.

Case and Request Decomposition

It is rare to find an exact match for a user request from the case base, but it is quite common that a fragment of the user request matches well with a fragment of a case [Mitchell, 1997]. To facilitate the retrieval of case fragments, we support case/request decomposition.

Case decomposition. To avoid creating a large search space, our challenge is to determine the granularity of sub-cases. Based on the data and visual characteristics of a graphic example [Zhou et al., 2002b], we develop four heuristics to guide our case decomposition.

1). We extract independent and meaningful *visual structures*, which are schematic illustrations for conveying concepts and processes [Winn, 1987]. For example, the table showing the city information (Figure 6a) is considered as a sub-case. Within a case, such a structure can be easily identified by the value (VisualStructure) of its feature Category.

2). We turn a case (e.g., Figure 6b) into a sub-case by trimming all its decorations (e.g., coordinate axes and legends). These simplified sub-cases are good matching candidates for most user requests, which normally do not specify the data for creating visual decorations.

3). We extract all visual leaf objects along with their data mappings from cases (e.g., V7 and D5 in Figure 3b) to form a visual dictionary [Zhou and Feiner, 1997]. The dictionary is used to find matches for user requests that contain a single node.

4). We decompose a case by data relations. Figure 5 shows two data relations (has-a and index) encoded in Figure 6(b), which lead to two sub-cases (Figure 5a–b). In Figure 5(b), a Dummy node is also added to preserve a rooted graph structure for a fast graph matching (see below).

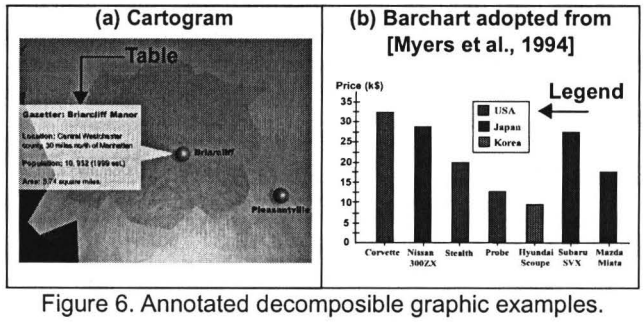


Figure 6. Annotated decomposible graphic examples.

For the sake of performance, we automatically extract all sub-cases during the case loading stage.

Request decomposition. Unlike case decomposition, which is done once during the case loading stage, request decomposition occurs whenever a retrieved top-matched case fails our adequacy evaluation. To ensure the retrieval quality without incurring the expense of rematching, we extract only the failed fragments as sub-requests, while retaining the results for succeeded ones. Here a *failed fragment* is the biggest possible rooted sub-graph that contains failed nodes excluding the root of the current request. Suppose that every node in fragment A (Figure 4) passes our evaluation, but a node (e.g., AName) in fragment B fails. We then create a sub-request containing the whole fragment B with City as the root, while keeping the matching results of fragment A. Depending on the matching results, the decomposition may be repeated until sub-requests contain a single node. The matched cases for these single-node requests can be found from our visual dictionary.

Since sub-requests break up the original structure of a request, it is always desirable *not* to produce too many sub-requests. Request decomposition however enables us to find desired case fragments from a set of heterogeneous cases as we have, where finding a maximal common subgraph may fail [Borner, 2001]. Unlike a static request decomposition used by other researchers [Michelena and Sycara, 1994], our decomposition is dynamically performed based on our adequacy evaluation results.

Improve Retrieval Speed

As the number of cases grows, the cost of searching for a desired case increases. Moreover, finding a graphic example that matches a user request best is a computationally intensive graph-matching process itself. Specifically, we need to match two pairs of PDGraphs and PVGraphs between a case and a user request. By exploiting the rooted hierarchical structure of PD/PVGraph, we perform an ordered, top-town graph matching [Zhou et al., 2002b] to accelerate otherwise an arbitrary graph matching process (e.g., matching two complex graphs similar to Figure 3a). To further improve case retrieval speed, we reduce the search space by using a hierarchically structured case base.

Instead of searching the case base linearly, we exploit the organization of the cases. We use a hierarchical clustering algorithm [Duda and Hart, 1973] to arrange all cases by their pair-wise distances computed using a similarity metric [Zhou et al., 2002b]. Figure 7 shows such a cluster hierarchy. Starting with the two outmost clusters in the hierarchy (e.g., clusters 1–2 in Figure 7), at each level we search only one

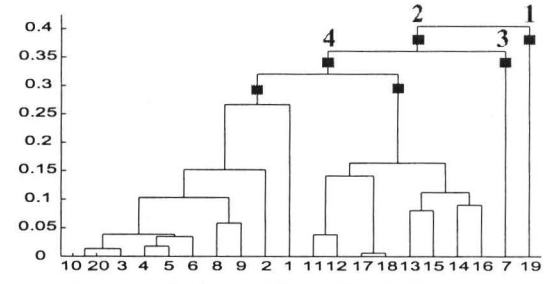


Figure 7. A cluster hierarchy of 20 cases.

cluster that is most likely to contain the top-matched case.

To find a cluster to follow, we first select a representative for each cluster using a quick approximation. A *representative* is a case closest to the request by three meta properties of its PDGraph: the total number of nodes, the average number of levels, and the average number of nodes at each level. We then select the cluster whose representative produces the shortest similarity distance to the request by our similarity metric. The rationale here is that the top-matched case for a request is *likely* to be in the same cluster of cases that match the request well by both meta properties and our similarity metric.

Following the selected cluster (e.g., cluster 2 in Figure 7) down the hierarchy (e.g., clusters 3–4), our algorithm repeats the above process until it explores a leaf cluster. In our experiments, this cluster-guided search improves the performance over a linear search by a factor of 3. Due to the approximation used, our method however is not guaranteed to find the top-matched case. Unlike other structured case search, where cluster representatives are pre-selected in advance [Borner, 2001], we dynamically compute representatives for each request to achieve a more accurate retrieval.

3.3 Case Composition: Sketch Synthesis

As the result of a successful retrieval, each data leaf node of the request acquires at least one visual candidate. Starting with the visual candidates of the leaves, our algorithm synthesizes a sketch from bottom up by creating visual candidates for higher-level data nodes and finally for the root. In this section, we address three challenges arising in sketch synthesis. First, we resolve visual candidate conflicts when a data node in the request acquires multiple candidates. Second, to compose multiple retrieved cases, we use decision-tree learning to generalize existing visual compositions and to verify new compositions. Third, we automatically infer visual decorations from existing cases, such as the coordinate axes and legends, to complete a visual sketch design.

Conflict Resolution of Visual Candidates

Within a user request, a data node may acquire multiple visual candidates from different case matching. For example, data nodes Name and Price appear in two sub-requests (Figure 5a–b). It is most likely for both items to obtain multiple visual candidates as the two sub-requests are matched to different cases. To select a proper visual candidate, we currently use both the matching priority set in the user request and the distance calculated during similarity measuring. Specifically, we first retain candidates that are acquired through matching the data relations with a higher priority. In the above example, we will keep the candidates for Name and Price acquired through matching the sub-request in Figure 5(b), since the priority for matching relation index precedes that of relation has-a (Figure 5a). If there are multiple candidates by the same matching priority, we then select candidates that have produced the shortest distance during the match. Should there still be multiple candidates left, our algorithm would choose a candidate randomly.

Generalizing and Verifying Visual Composition

Our sketch synthesis uses visual candidates acquired for the lower-level data nodes to create visual candidates for the higher-level nodes in the request. Since visual candidates may be retrieved from different cases, their compositions may never exist before and new compositions are needed to piece them together. Here we denote a visual composition ($\Rightarrow$) of N visual elements as:

$$E_1, ..., E_N \Rightarrow rC,$$

where E_i is the ith element, r is the composition relation, and C is the composed visual object. Below is an example of a cartogram (a map-based presentation) composition:

Position, Map $\Rightarrow$ *overlay*Cartogram.

It composes a Cartogram by overlaying a Position element on top of a Map element. In our approach, a new composition for an intended data node (e.g., Amenity in Figure 4) is proposed using the composition information recorded in its children (e.g., AName) during the case retrieval. Specifically, in each data node that acquires a visual candidate V from the retrieved case (e.g., V7 from Figure 3a), we record the composition relation and category specified in V's parent (e.g., V4). As a result, for each retrieved visual candidate our algorithm records the possible compositions that the visual candidate has participated in.

However a proposed composition may not always be valid. Here *validity* means that a visual composition must produce an effective visual design [Mackinlay, 1986]. For example, the above cartogram composition is a valid composition, but composing a visual object by juxtaposing two horizontal position elements is not. Without hand-crafting rules or pre-defining connectors [Michelena and Sycara, 1994], we use a decision-tree learning technique to automatically induce a set of classification rules from 200 visual composition samples extracted from our cases. By our visual composition definition, we describe each sample using N input features specifying N visual element categories (e.g., Position and Map) and one target specifying a combined composition relation and category (e.g., overlay_Cartogram). Currently N is set to 3 since most of our samples contain 3 elements. If a sample contains fewer than 3 elements, the remaining features are set to null. We train C4.5 [Quinlan, 1993], a decision-tree learning algorithm, on the 200 samples using 5-fold cross-validation, a standard procedure when the amount of data is limited. We obtain 14 generalized composition rules with an overall classification error of 22%. We then use these rules to verify a new composition.

To compensate for the situation where there is a lack of visual composition samples, we introduce *negative samples* that are known invalid compositions to help identify *invalid* new compositions. Similar to the above process, we use classification rules induced from the negative samples to verify invalid new compositions to be eliminated. After both positive and negative verifications, if there are still multiple proposed compositions we use the confidence factors generated in the decision-tree learning to select the most probably valid composition. Without a sufficient number of composition samples, our approach may not always verify a composition correctly. That is why we involve users in the design process to help IMPROVISE+ in its decision-making (Figure 2).

Inferring Visual Decoration

A sketch is incomplete without the necessary *visual decorations*, such as coordinate axes and legends, which can guide users to comprehend the information encoded in the

graphic [Wilkinson, 1999]. However in a request a user rarely specifies the data for creating such decorations, which our algorithm must infer.

Currently our inference is based on an assumption that visual decorations can be created using only data leaves in a request. According to visual psychology studies, a person tends to first perceive the global structure of a graphic (e.g., a spatial map) then examine the details (a particular color or a position) [Goldsmith, 1984]. Only when a person attempts to interpret the meanings of visual primitives (e.g., the color code in Figure 6b), visual decorations are needed (e.g., the legend in Figure 6b). Hence visual decorations normally encode leaf data or their transformations.

Specifically, for each leaf L and its match (D, V) from a retrieved case C, we trace the visual mappings of the matched data node D in C. Within case C, if data D is used to create visual decorations, our algorithm would create a visual decoration for L in the user request. More precisely, if there exists a data-visual mapping pair (D', V') in case C, where D' is D or a transformation of D, and V' is a visual decoration; V' becomes a visual decoration candidate for the new sketch. However, L may need to be transformed for creating V', since D' is often a transformation of D. Suppose that a user asks to present the depth of different lakes (LakeDepth), which happens to acquire a match (Price, VertLength) from a retrieved case as in Figure 6(b). Here Price (a *domain*) has been transformed into a *range* [minPrice, maxPrice] in creating the Y axis. Similarly, in the new sketch a Y axis will be created for encoding the range of LakeDepth converted from the LakeDepth domain.

4 An Example

Here we demonstrate how a specific user request (Figure 4) is fulfilled. For this request, our algorithm first retrieves the top-matched case, which fails our adequacy test, as the similarity distance (0.5803) for node Gazetteer in fragment B is above the required threshold (0.3). While keeping the matches for fragment A, we extract the whole fragment B as a sub-request with node City as its root, since we always attempt to find an overall good match for the bigger sub-graph (e.g., the sub-graph covering B instead of B2). Next, the top-matched case for fragment B is retrieved, where nodes in B1 pass the evaluation, but those in B2–B3 do not. Subsequently, our algorithm treats fragments B2–B3 as two new sub-requests and finds matches for them. Except nodes under fragments B21 and B31, all other data nodes have now acquired acceptable matches. Eventually desired matches are found for the last two sub-requests generated for fragments B21 and B31. Since B31 contains only one node, its match is retrieved from our visual dictionary.

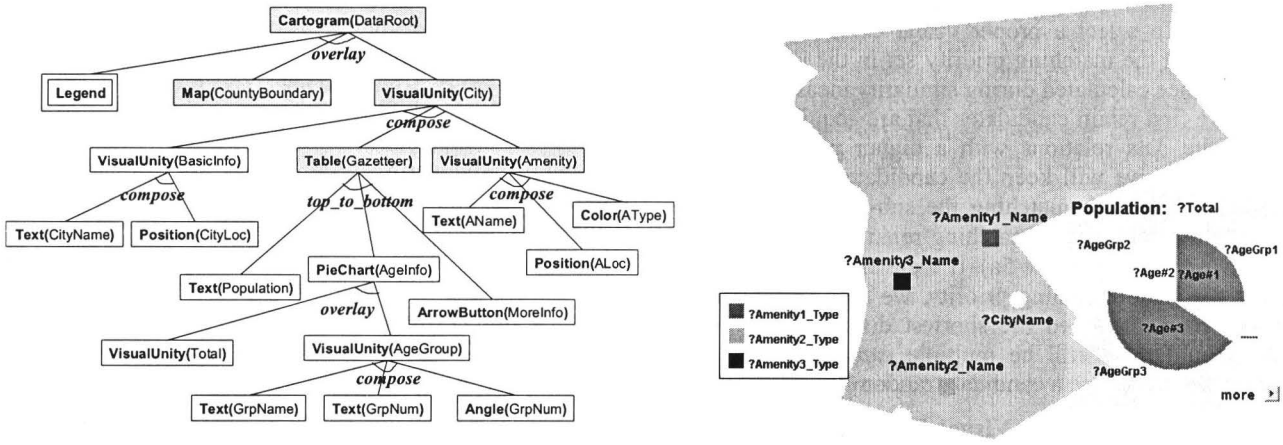
Using the retrieved results, our algorithm synthesizes a new sketch from bottom up (Figure 8a). It proposes new compositions to create visual candidates for shaded data nodes, while reusing retrieved candidates for others. Specifically, a new composition is proposed for Gazetteer, since the visual candidates for its children Population, AgeInfo, and MoreInfo, are retrieved from different cases. In this case, a top-to-bottom_Table composition is recorded for Population during the retrieval. According to C4.5, this is a valid composition, thus it is used to create a Table for conveying Gazetteer with 3 items arranged from top to bottom: a text, a pie chart, and an arrow button. Similarly, visual candidates are created for Amenity, then for City, at last for the root. As a result, a new sketch is created for this request with an inferred legend for interpreting the type of amenities (Figure 8b).

If the user chooses to use a different set of cases, an alternative can be created for the same request (Figure 1). Once a new sketch is created, it can be directly stored back in our case base as a new addition, since the generated sketch has a representation similar to other existing cases.

5 System Implementation and Performance

IMPROVISE+ is implemented using Java and C++. We use Java to implement all design components, including our case-based sketch generation algorithm. We have implemented two rendering components, one in Java2D for 2D graphics and the other in C++/Open Inventor/OpenGL for 3D graphics. IMPROVISE+ currently runs on Win32/Linux/SGI. On a PC with a 1.13 GHz Pentium III processor, it takes about 5 seconds to create the sketch shown in Figure 8(b) from about 100 cases, each of which contains about 30 data nodes and 30 visual nodes on average.

We have also conducted several experiments to test the effectiveness of our approach. According to our user feed-



(a) Graph structure of a composed sketch **(b) Rendered graphic sketch**

Figure 8. A composed sketch and its rendering fort the user request in Figure 5.

back, our case-based approach can provide more versatile and "creative" design suggestions (e.g., Figure 1) than a rule-based approach can [Zhou, 1999]. Moreover, involving users in proper design stages helps create a more tailored visual presentation. For example, by specifying different retrieval preferences [Zhou et al., 2002b], users can choose among different design alternatives (Figure 1 vs. Figure 8b).

6 Conclusions and Future Work

In this paper, we have presented a case-based graphic sketch generation algorithm with an emphasis on its three unique features and how they facilitate an efficient and effective sketch generation. First, we present an adequacy-guided case retrieval method, which augments a similarity metric with a set of adequacy evaluation criteria to retrieve a top-matched case that is also usable in sketch synthesis. To facilitate the retrieval of case fragments, we also describe how to systematically decompose a case/request when a usable case cannot be found. Second, we explain how to enhance case retrieval speed by organizing cases into hierarchical clusters based on their similarity distances and by using dynamically selected cluster representatives. Third, we show how to construct a new sketch through case composition, including case generalization and visual decoration inference (e.g., inferring coordinate axes and legends).

Currently, we are working in two areas to improve IMPROVISE+. First, we are creating a GUI that allows users to easily specify a request by "drawing" a graph similar to Figure 4 without writing an XML document. Second, we are developing more sophisticated interaction support, where IMPROVISE+ can automatically engage users in interaction based on context. For example, if IMPROVISE+ cannot find a valid visual composition using decision-tree learning, it may decide to solicit user inputs.

Acknowledgments

We would like to thank Keith Houck and Alison Lee for proofreading the earlier version of this paper. We would also like to thank Shimei Pan for useful discussions on case-based learning.

References

Andre, E. and Rist, T. (1993). The design of illustrated documents as a planning task. In Maybury, M., editor, *Intelligent Multimedia Interfaces*, chapter 4, pages 94–116. AAAI Press/The MIT Press, Menlo Park, CA.

Borner, K. (2001). Efficient case-based structure generation for design support. *AI Review*, 16(2):87–118.

Chuah, M., Roth, S., and Kerpedjiev, S. (1997). Sketching, searching, and customizing visualizations: A content-based approach to design retrieval. In Maybury, M., editor, *Intelligent Multimedia Information Retrieval*, pages 83–111. AAAI Press/The MIT Press.

Duda, R. and Hart, P. (1973). *Pattern Classification and Scene Analysis*. Wiley.

Goldsmith, E. (1984). *Research into Illustration: An Approach and a Review*. Cambridge University Press, Cambridge, England.

Mackinlay, J. (1986). Automating the design of graphical presentations of relational information. *ACM Trans. on Graphics*, 5(2):110–141.

Michelena, N. and Sycara, K. (1994). Physical synthesis in case-based design. In *Proc. of ASME DTM '94*, pages 273–284.

Mitchell, T. (1997). *Machine Learning*. McGraw-Hill.

Myers, B., Goldstein, J., and Goldberg, M. (1994). Creating charts by demonstration. In *CHI '94*, pages 106–111.

Quinlan, J. (1993). *C4.5: Programs for Machine Learning*. Morgan Kaufman.

Sycara, K., Guttal, R., Koning, J., Narasimhan, S., and Chandra, D. (1992). Cadet: A case-based synthesis tool for engineering design. *Intl. J. of Expert Systems*, 4:157–188.

Wilkinson, L. (1999). *The Grammar of Graphics*. Springer.

Winn, W. (1987). Charts, graphs, and diagrams in educational materials. In Willows, D. and Houghton, H., editors, *The Psychology of Illustration: Basic Research*, volume 1, chapter 5, pages 152–198. Springer-Verlag.

Zhou, M. (1999). Visual planning: A practical approach to automated visual presentation. In *Proc. IJCAI '99*, pages 634–641.

Zhou, M., Chen, M., and Feng, Y. (2002a). Building a visual database for example-based graphics generation. In *Proc. IEEE InfoVis '02*, pages 23–30.

Zhou, M. and Feiner, S. (1997). The representation and use of a visual lexicon for automated graphics generation. In *Proc. IJCAI '97*, pages 1056–1062, Nagoya, Japan.

Zhou, M., Ma, S., and Feng, Y. (2002b). Applying machine learning to automated information graphics generation. *IBM Sys. J.*, 41(3):504–523.