[50] M. M. Zloof, "Office-by-example: A business language that unifies data and word processing and electronic mail," *IBM Systems Journal*, vol. 21, no. 3, pp. 272-304, 1982.

[34] J. L. Sussman and S. H. Kim, "Three dimensional structure of a transfer RNA in two crystal forms," *Science*, vol. 192, pp. 853, 1976.

[35] K. C. Tai, "The tree-to-tree correction problem," *J. Ass. Comput. Mach.*, vol. 26, no. 3, pp. 422-433, July 1979.

[36] A. U. Tansel, M. E. Arkun, and G. Ozsoyoglu, "Time-by-example query language for historical databases," *IEEE Trans. Software Eng.*, vol. 15, no. 4, pp. 464-478, Apr. 1989.

[37] E. Ukkonen, "Finding approximate pattern in strings," *J. Algorithms*, vol. 6, pp. 132-137, 1985.

[38] P. D. Vaidya, L. G. Shapiro, R. M. Haralick, and G. J. Minden, "Design and architectural implications of a spatial information system," *IEEE Trans. Comput.*, vol. 31, pp. 1025-1031, 1982.

[39] T. L. Wang and D. Shasha, "Query processing for distance metrics," in *Proc. 16th Int'l Conf. on Very Large Data Bases*, Brisbane, Australia, Aug. 1990, pp. 602-613.

[40] T. L. Wang, "Query optimization in database and information retrieval systems," Ph.D. dissertation, Dep. Comput. Sci., Courant Institute of Math. Sci., New York Univ., New York, 1991.

[41] J. T. L. Wang, K. Jeong, K. Zhang, and D. Shasha, *Reference manual for ATBE: A tool for approximate tree matching*, Tech. Rep. TR-551, Courant Institute of Math. Sci., New York Univ., New York, 1991, pp. 1-57.

[42] J. T. L. Wang, K. Zhang, K. Jeong, and D. Shasha, "A tool for tree pattern matching," in *Proc. 3rd IEEE Int'l Conf. on Tools for Artificial Intell.*, San Jose, CA, Nov. 1991, pp. 436-444.

[43] C. Wetherell and A. Shannon, "Tidy drawings of trees," *IEEE Trans. Software Eng.*, vol. 5, pp. 514-520, 1979.

[44] *X Toolkit Intrinsics Programming Manual*, O'Reilly & Associates, Inc., CA, 1990.

[45] K. Zhang, "The editing distance between trees: algorithms and applications," Ph.D. dissertation, Dep. Comput. Sci., Courant Institute of Math. Sci., New York Univ., New York, 1989.

[46] K. Zhang and D. Shasha, "Simple fast algorithms for the editing distance between trees and related problems," *SIAM J. Comput.*, vol. 18, pp. 1245-1262, Dec. 1989.

[47] K. Zhang, D. Shasha, and J. T. L. Wang, "Fast serial and parallel algorithms for approximate tree matching with VLDC's," in *Proc. 3rd Int'l Conf. on Combinatorial Pattern Matching*, Tucson, Arizona, April/May, 1992.

[48] K. Zhang, R. Statman, and D. Shasha, "On the editing distance between unordered labeled trees," *Information Processing Letters*, in press.

[49] M. M. Zloof, "Query-by-example," in *Proc. Nat. Comput. Conf.*, May 1975, pp. 431-438.

[19] S. Y. Lu, "A tree-matching algorithm based on node splitting and merging," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 6, pp. 249-256, Mar. 1984.

[20] B. Moayer and K. S. Fu, "A tree system approach for fingerprint pattern recognition," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 8, pp. 376-387, May 1986.

[21] E. W. Myers and W. Miller, "Approximate matching of regular expressions," *Bulletin of Mathematical Biology*, vol. 51, no. 1, pp. 5-37, 1989.

[22] M. Neff, R. Byrd, and O. Rizk, "Creating and querying hierarchical lexical data bases," in *Proc. 2nd Conf. Applied Natural Language Processing*, 1988, pp. 84-93.

[23] M. Neff and B. K. Boguraev, "Dictionaries, dictionary grammars and dictionary entry parsing," in *Proc. 27th Annual Meeting of the Association for Computational Linguistics*, June 1989.

[24] J. A. Orenstein and F. A. Manola, "PROBE spatial data modeling and query processing in an image database application," *IEEE Trans. Software Eng.*, vol. 14, no. 5, pp. 611-629, May 1988.

[25] G. Ozsoyoglu, V. Matos, and Z. M. Ozsoyoglu, "Query processing techniques in the summary-table-by-example database query language," *ACM Trans. Database Syst.*, vol. 14, no. 4, pp. 526-573, Dec. 1989.

[26] J. Pustejovsky, "The semantic representation of lexical knowledge," Lexical Acquisition: Using On-Line Resources to Build a Lexicon, MIT Press, Uri Zernik (Ed.), 1989.

[27] E. M. Reingold and J. S. Tilford, "Tidier drawings of trees," *IEEE Trans. Software Eng.*, vol. 7, pp. 223-228, 1981.

[28] N. Roussopoulos, C. Faloutsos, and T. Sellis, "An efficient pictorial database system for PSQL," *IEEE Trans. Software Eng.*, vol. 14, no. 5, pp. 639-650, May 1988.

[29] H. Samet, "Distance transform for images represented by quadtrees," *IEEE Trans. on Pattern Anal. Machine Intell.*, vol. 4, no. 3, pp. 298-303, May 1982.

[30] B. A. Shapiro and K. Zhang, "Comparing multiple RNA secondary structures using tree comparisons," *Comput. Appl. Biosci.*, vol. 6, no. 4, pp. 309-318, 1990.

[31] L. G. Shapiro and R. M. Haralick, "Structural descriptions and inexact matching," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 3, no. 5, pp. 504-519, Sep. 1981.

[32] D. Shasha and T. L. Wang, "New techniques for best-match retrieval," *ACM Transactions on Information Systems*, vol. 8, no. 2, pp. 140-158, Apr. 1990.

[33] M. Stonebraker, E. Wong, P. Kreps, and G. Held, "The design and implementation of INGRES," *ACM Trans. Database Syst.*, vol. 1, no. 3, pp. 189-222, 1976.

[4] R. S. Boyer and J. S. Moore, "A fast string matching algorithm," *Comm. ACM*, vol. 20, pp. 262-272, 1977.

[5] R. Byrd, *LQL User Notes: An Informal Guide to the Lexical Query Language*, Tech. Rep., IBM T. J. Watson Research Center, Yorktown Heights, New York, 1990.

[6] S. K. Chang and T. L. Kunii, "Pictorial data-base systems," *Computer*, vol. 14, no. 11, pp. 13-21, 1981.

[7] Y. C. Cheng and S. Y. Lu, "Waveform correlation by tree matching," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 7, pp. 299-305, May 1985.

[8] M. Chock, A. F. Cardenas, and A. Klinger, "Database structure manipulation capabilities of the picture database management system (PICDMS)," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 6, no. 4, pp. 484-492, July 1984.

[9] M. S. Chodorow, R. J. Byrd, and G. E. Heidorn, "Extracting semantic hierarchies from a large online dictionary," in *Proc. Annual Meetings of the Association for Computational Linguistics*, 1985, pp. 299-304.

[10] M. Chodorow and J. L. Klavans, "Locating syntactic patterns in text corpora," Manuscript, Lexical Systems project, IBM T. J. Watson Research Center, Yorktown Heights, New York, 1990.

[11] L. S. Davis and N. Roussopoulos, "Approximate pattern matching in a pattern database system," *Information Systems*, vol. 5, pp. 107-119, 1980.

[12] R. O. Duda and P. E. Hart, *Pattern Classification and Scene Analysis*, Wiley, New York, 1973.

[13] R. W. Ehrich and J. P. Foith, "Representation of random waveforms by relational trees," *IEEE Trans. Comput.*, vol. 25, pp. 725-736, 1976.

[14] C. M. Hoffmann and M. J. O'Donnell, "Pattern matching in trees," *J. Ass. Comput. Mach.*, vol. 29, pp. 68-95, 1982.

[15] B. E. Jacobs and C. A. Walczak, "A generalized query-by-example data manipulation language based on database logic," *IEEE Trans. Software Eng.*, vol. 9, no. 1, pp. 40-57, Jan. 1983.

[16] R. L. Kashyap and B. J. Oommen, "The noisy substring matching problem," *IEEE Trans. Software Eng.*, vol. 9, no. 3, pp. 365-370, May 1983.

[17] S. Kosaraju, "Efficient Tree Pattern Matching," in *Proc. 30th Annual IEEE Symp. on Foundations of Computer Science*, Oct. 1989, pp. 178-183.

[18] G. M. Landau and U. Vishkin, "Introducing efficient parallelism into approximate string matching and a new serial algorithm," in *Proc. 18th Annual ACM Symp. on Theory of Computing*, ACM, New York, 1986, pp. 220-230.

We use the editing distance to measure the dissimilarity between two trees. In some applications, researchers have proposed different measurements for tree matching (see, e.g., [7], [19]). To adapt our system and develop it into a custom environment for specific distance measures, we have designed the system in a very modular way, cleanly separating routines for tree comparison from all other routines (i.e., those for query optimization, graphical interface, etc.). Thus, the user can modify those routines without changing the rest of the system. The modular design also facilitates augmenting the system with additional functions.

Work on ATBE is continuing. We have two main goals.

1. Our system, as well as our algorithms, deal with ordered, labeled trees. In [48], it was shown that the problem of finding the editing distance between unordered, labeled trees (i.e., trees in which the left-to-right order of each node's children is unimportant) is NP-complete. We are investigating heuristics for comparing these trees, and plan to extend our system to handle them.

2. The present lack of techniques for optimizing queries containing variables (or umbrellas) may degrade the system performance seriously. We are currently working on optimization strategies for such queries.

ATBE is used in several universities. We would be pleased to share ATBE software and experiences with other groups pursuing relevant research. Readers interested in obtaining the software should send a written request to any one of the authors.

# 7 Acknowledgements

We would like to thank members of the lexical systems project at IBM T. J. Watson Research Center, in particular B. Boguraev, R. Byrd, M. Chodorow, J. Klavans, M. Neff, and Y. Ravin for helpful discussions concerning this work. The systems of R. Byrd and M. Chodorow were very inspirational. We would also like to thank: P. Kilpelainen of the University of Helsinki, for providing valuable comments in using ATBE; our colleagues A. Howell, M. Smosna and K. Snyder, for assisting us in implementing ATBE; and the anonymous referees and the editor D. Spooner, for their constructive suggestions that have greatly improved the readability of this paper.

# References

[1] A. V. Aho, M. Ganapathi, and S. W. K. Tjiang, "Code generation using tree matching and dynamic programming," *ACM Transactions on Programming Languages and Systems*, vol. 11, no. 4, pp. 491-516, Oct. 1989.

[2] A. M. Alashqur, S. Y. W. Su, and H. Lam, "OQL: A query language for manipulating object-oriented databases," in *Proc. 15th Int'l Conf. on Very Large Data Bases*, Aug. 1989, pp. 433-442.

[3] M. M. Astrahan, et al., "System R: A relational approach to data management," *ACM Trans. Database Syst.*, vol. 1, no. 2, pp. 97-137, 1976.

However, in very special cases, namely, those in which the pattern contains only bars, we can derive bounding procedures similar to those used in the mark-free case. Consider a simple case where the pattern $pa$ contains one bar. Suppose $dist(pa,t) = k_1$ and $dist(t,t') = k_2$ for two arbitrary trees $t$ and $t'$ in $F$. We claim that

$$dist(pa,t') \leq k_1 + k_2.$$

To show this, consider the set $E$ of edit operations that transform $pa$ to $t$, and the set $E'$ of edit operations that transform $t$ to $t'$. Let $P$ be the path in $t$ that is matched with the bar. Let $E'' \subseteq E'$ be the set of edit operations applied to $P$, and let $P'$ be the resulting path in $t'$. (The length of $P'$ may be zero.) Thus, the distance between $pa$ and $t'$ is bounded by $k_1 + k_2$ because at worst, we can match the bar with $P'$ (at zero cost) and apply edit operations in $E \cup E'$ to transform the remaining part of $pa$ to $t'$.

The upper bound offers a useful cut-off criterion to eliminate trees from consideration when searching for farther trees from the pattern. For example, consider retrieving trees in file $F$ that are most dissimilar to the pattern $pa$. Let $t_w$ represent the current worst match. Using our previous arguments, if $dist(pa,t_i) \leq dist(pa,t_w)$, we can filter out trees $t$ where

$$dist(t,t_i) < dist(pa,t_w) - dist(pa,t_i).$$

On the other hand, if $dist(pa,t_i) > dist(pa,t_w)$, then $t_i$ becomes the current worst match, and we can eliminate trees $t$ from consideration where

$$dist(t,t_j) < dist(pa,t_i) - dist(pa,t_j) \qquad 1 \leq j \leq i-1.$$

This example illustrates how to use the triangle inequality to eliminate irrelevant trees for the worst-match retrieval. Unfortunately, developing complete cut-off procedures for queries containing all types of marks is a non-trivial problem. Techniques for optimizing such queries remain to be explored.

# 6 Conclusions and Future Work

This paper presents an overview of the ATBE system. The system makes several contributions.

1. It is the first system to support approximate tree matching.

2. It supports many useful optional functions such as the ability, while computing the distance between the pattern and a data tree, to

   - cut (and prune) those subtrees from the data tree that will minimize the distance;
   - substitute for those variable length don't care nodes of the data tree that will minimize the distance; and
   - instantiate variables placed as leaves in the pattern tree.

3. It provides a query language that allows users to combine a variety of constraints in flexible ways.

or closer to $t_j$ than $dist(pa,t_j) - dist(pa,t_i)$ (if these trees haven't been eliminated from consideration yet).

Thus, by exploiting the triangle inequality and precomputed intra-file distances, we can filter out trees that could not possibly satisfy the query, given the distances known so far.[11] To expedite the query evaluation, we also developed a heuristic, called *pick least lower bound*, for picking trees. The heuristic works by picking its first tree randomly, and then in subsequent stages, it selects a tree $t$ such that the lower bound of the distance between $t$ and the given pattern $pa$ is minimized based on all previously computed trees. (These lower bounds are obtained by $dist(pa,t) \geq |dist(pa,t') - dist(t',t)|$ where $t'$ represents trees picked previously.) Intuitively this heuristic uses the lower bound to estimate the exact distance. Thus the tree having the least lower bound is expected to be (potentially) the closest tree to $pa$. If several trees have the same lower bound, the heuristic selects one that has the least upper bound. (The upper bound is obtained by $dist(pa,t) \leq dist(pa,t') + dist(t',t)$.) The reason for doing so is that we expect the smaller the difference between the lower and upper bounds, the more precise the estimated distance is. Ties on the difference are broken arbitrarily.[12]

## 5.2 Algorithms for Processing Queries Containing Bars

In general, the triangle inequality does not hold when the pattern contains marks. Figure 13 illustrates this case.

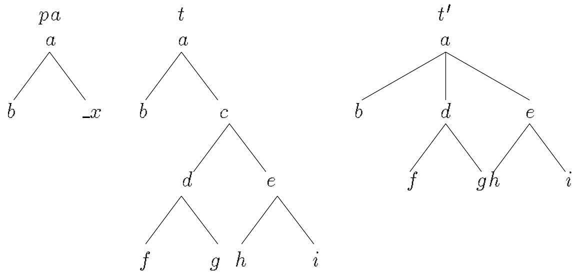


Fig. 13. Assume that all edit operations cost one.

- In computing $dist(pa,t)$, $a, b$ in $pa$ would match their corresponding nodes in $t$, and _$x$ would match the subtree rooted at $c$. The resulting distance would be 0.
- In computing $dist(t,t')$, $c$ would be deleted and all other nodes in $t$ would match their corresponding nodes in $t'$. The resulting distance would be 1 (representing the cost of deleting $c$).
- In computing $dist(pa,t')$, $a, b$ in $pa$ would match their corresponding nodes in $t'$, and _$x$ would match either the subtree rooted at $d$ or the subtree rooted at $e$. In both cases, the resulting distance would be 3 (representing the cost of deleting the three nodes in the subtree not used for the instantiation).

Thus, $dist(pa,t') = 3 \not\leq dist(pa,t) + dist(t,t') = 0 + 1 = 1$.

[11] We put data trees and intra-file distances in UNIX files. The distance information is stored separately from the trees, and is read into the query processor when needed. Distances between newly inserted trees and all other trees already in the file are added periodically. Deleted trees are marked; these trees, together with their distances, are erased manually in a periodical manner as well.

[12] We have conducted experiments to evaluate the behavior of the algorithms. The results indicate that the performance of these algorithms is slightly influenced by file sizes, but is strongly dependent on distance distribution [32], [39]. The algorithms work well if distance distribution is not seriously skewed. For instance, in processing best-match queries, our algorithms can eliminate nearly 80% trees, on the average, from consideration for uniformly distributed distances.

The second set of algorithms was given in [47]. The algorithms compute the distance between $T_1$ and $T_2$, assuming one of them contains variables (bars, umbrellas). The algorithms also find the best mapping yielding the distance, and locate appropriate subtrees (or nodes) used for instantiating the marks. The time and space complexities of these algorithms are the same as those for the first set of algorithms.

The third set of algorithms essentially employs the triangle inequality to reduce computational efforts during query evaluation. Illustration of these algorithms might be instructive to researchers who develop query optimizers for other advanced information systems, such as pictorial [6], [8], [28], or spatial database systems [24], [38]. Below, we describe these algorithms, first focusing on patterns without marks, and then those with.

## 5.1 Algorithms for Processing Queries without Marks

For exposition purpose, we use the best-match query as a running example. More general cases can be found in [32], [39], [40].

Given a file $F$ of $n$ trees and a pattern $pa$, one straightforward way of finding trees in $F$ that are closest to $pa$ is to compute the distance between each tree of $F$ and $pa$, and then search for the trees with minimum distance. The major problem with this approach is its computational expense, particularly when trees or files are large.

Our approach instead is to first precompute pairwise distances between trees in $F$. We then proceed in stages, picking one tree at a time, comparing it against the pattern, updating the current best-matches (if necessary), and eliminating certain trees from consideration. Specifically, suppose at some point, we just computed $dist(pa, t_i)$ for a tree $t_i$ in $F$, and the current best-matching tree is $t_b$. Two cases may arise:

Case 1. $dist(pa, t_i) \geq dist(pa, t_b)$. In this case, we can eliminate trees $t$ that are farther away from $t_i$ than $dist(pa, t_i) + dist(pa, t_b)$ or closer to $t_i$ than $dist(pa, t_i) - dist(pa, t_b)$ from consideration, because they cannot contribute to solutions. To see this, notice that

$$\begin{array}{lcll}
dist(pa, t) & \geq & |dist(t, t_i) - dist(pa, t_i)| & \text{(by the triangle inequality)} \\
 & > & dist(pa, t_i) + dist(pa, t_b) - dist(pa, t_i) & \text{(if } dist(t, t_i) > dist(pa, t_i) + dist(pa, t_b)\text{)} \\
 & & \text{or} & \\
 & & dist(pa, t_i) - (dist(pa, t_i) - dist(pa, t_b)) & \text{(if } dist(t, t_i) < dist(pa, t_i) - dist(pa, t_b)\text{)} \\
 & = & dist(pa, t_b) &
\end{array}$$

which implies that $t$ would never become the best-matching tree.

Case 2. $dist(pa, t_i) < dist(pa, t_b)$. In this case $t_i$ becomes the current best-matching tree, and we can eliminate trees $t$ that are farther away from $t_i$ than $2 \times dist(pa, t_i)$ from consideration, because

$$\begin{array}{lcll}
dist(pa, t) & \geq & |dist(t, t_i) - dist(pa, t_i)| & \text{(by the triangle inequality)} \\
 & > & 2 \times dist(pa, t_i) - dist(pa, t_i) & \\
 & = & dist(pa, t_i) &
\end{array}$$

Moreover, let $t_1 \ldots t_{i-1}$ be the $i-1$ trees whose distances to $pa$ have been computed in previous stages. We can eliminate trees $t$ from consideration where $t$ is farther away from $t_j$, $1 \leq j \leq i-1$, than $dist(pa, t_j) + dist(pa, t_i)$

### 4.3 Customizing the ATBE System

The ATBE system is customizable. By providing node definitions and cost functions for trees, users can tailor the system to meet the need for different applications. We briefly describe the procedure here. For further details about the use of the system, the reader is referred to [41], [42].

Recall that each node in an ordered, labeled tree has a label and possibly some node contents. A simple language (written in C) is provided to specify these node contents. As an example, consider the molecular structure shown in Figure 5. Each node in the structure is associated with a label and an integer, which represents the size property of the node. The node definition for this type of tree thus looks as follows:

```
Node_L
{
int size;
}
```

Here, Node_L is node's identification. size refers to the information the integer represents. Notice that the node's label (of string type) is not specified in the above definition. This is so because we assume that every node has a label and it is treated as an implicit field.

Users feed various node definitions as described above into the system. A set of I/O programs (which are used to read trees with specified node formats) as well as cost function programs specific to the user-defined node formats are generated. Users may then modify the cost function programs to meet their application's requirements.[10] The resulting programs are then compiled and linked with other modules (i.e., those for tree comparison, query optimization and graphical interface) to produce a custom system.

In using the generated system, users may input queries that refer to trees with different node formats. To compute the distance between these trees, users must provide correct format information and the corresponding cost functions for edit operations. (This is done by clicking appropriate pop-up menu items and filling in the information in dialogue windows.) If the information does not match the trees being compared, the system shows error messages and ignores the corresponding query.

## 5 Underlying Algorithms

Major algorithms used to implement ATBE can be classified into three categories: (1) those for computing distances between trees without marks (i.e., variables, bars, or umbrellas), (2) those for computing distances between trees with marks (and also instantiating them), and (3) those for query optimization.

The first set of algorithms was presented in [46]. Given two trees $T_1$ and $T_2$, the algorithms compute their distance (with or without cuttings or prunings) in $O(|T_1| \times |T_2| \times \min(depth(T_1), leaves(T_1)) \times \min(depth(T_2), leaves(T_2)))$ time and using $O(|T_1| \times |T_2|)$ space. The computation also produces (as a by-product) the best mapping that yields the distance, as well as distances between all subtrees of the two trees.

[10] For example, insert or delete a node of format Node_L may cost 5, and relabeling cost 3. When users do not specify cost functions, the system provides some default cost functions.

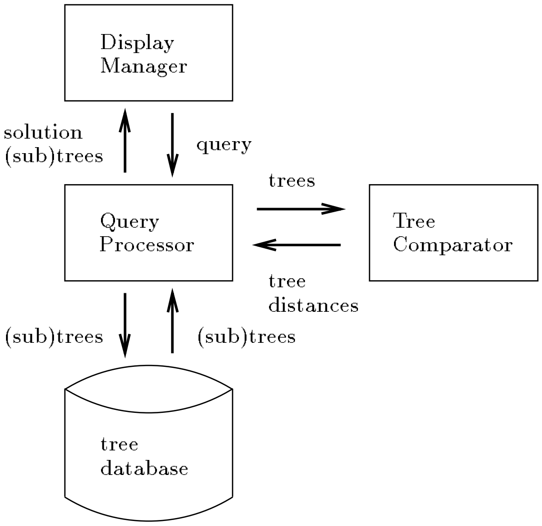


Fig. 12. ATBE system organization.

The display manager is menu-driven, with a number of available functions at each state. Each state is associated with a set of commands (displayed via pop-up menus) which causes a state transition when executed. When a command is entered, the display manager either updates the current window (if the command is recognized), or prints an error message and goes back to the state from which the command was entered (if the command is erroneous). After the user finishes constructing the query, he types a carriage return and the display manager goes through a series of consistency checks on the query. If errors appear in the query, the display manager shows error messages on the screen, and goes back to the initial state, waiting for another query. (Typical errors may include inserting unmatched parentheses or braces when the pattern is keyed in, or giving wrong information for trees being compared, as we will explain later.) If no errors are found, the display manager transforms the pattern into its linear form (if the query was not keyed in), and stores the query into a file. It then passes the file name as well as control to the query processor.

## 4.2 Query Processor

When the query processor gets control, it reads the query from the file specified by the passed file name. It then parses the query, and accesses files where relevant data trees are stored. (The data trees are stored in their linear forms; cf. Section 3.1.) Whenever encountering tree comparison operators such as dist, distwithcut, distwithprune, the processor encodes corresponding trees into a file and invokes the tree comparator to execute these operators. Since computing distances between trees is usually time-consuming, we have developed algorithms to prevent the query processor from exhaustively searching data trees in a file (see Section 5). In addition, the query processor is responsible for choosing appropriate heuristics from a pool of heuristics tailored to different queries. After finding solution trees, the query processor stores them in a file; it then passes the file name and control back to the display manager.

Thus, by attaching variables and bars in the pattern, users can extract information (nodes, subtrees) from the database.

### 3.5 Updating Operations

Having described ATBE's retrieval/extraction operations, we now turn to its updating operations. ATBE provides *insert* and *delete* operations to maintain a database. For example, if the user wishes to erase the pattern (with name, say, foo) shown in Figure 5 from file F, he would type the statement

```
delete foo from F
```

Modifying a tree can be achieved by first retrieving it from the file, e.g.,

```
retrieve tree foo from F
```

editing it, erasing the original copy, and then storing the new copy by typing

```
insert foo into F
```

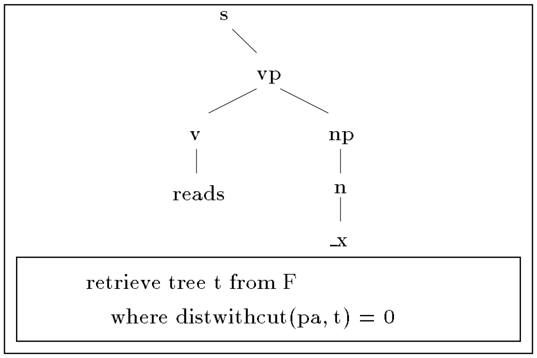
## 4 System Organization and Implementation

ATBE is implemented in C and X-windows. It currently runs on SPARC workstations under the Sun operating system version 4.1.1. Figure 12 shows the organization of the system. The display manager displays trees, their mapping information, and assists the user to form an ATBE query. The query processor parses the query and performs query optimization. The tree comparator is responsible for computing distances between trees.

When the system is first started, the display manager is activated. It accepts a query, performs syntax checking on the pattern and the input statement. If there is no error, the query is passed to the query processor. When the processor is parsing the query, it recognizes files to be accessed, retrieves trees from them, and invokes the tree comparator whenever necessary to perform tree comparison. The query processor produces the output of the query. The control then returns to the display manager, which displays the answers on the screen. These modules communicate by writing data into files; file names are passed as parameters.

### 4.1 Display Manager

The display manager consists of two components, one that provides a user interface for editing queries, and one that displays solution trees. In editing a pattern, ATBE allows users to use the text editor (for keying in the pattern), or alternatively a tree editor. The tree editor enables users to insert, delete, remove, copy subtrees, modify node labels and contents, and attach bars and umbrellas. There is no restriction on the order of these operations. The display manager also helps users format their output (in vertical or horizontal normal forms), and shows the mapping information.

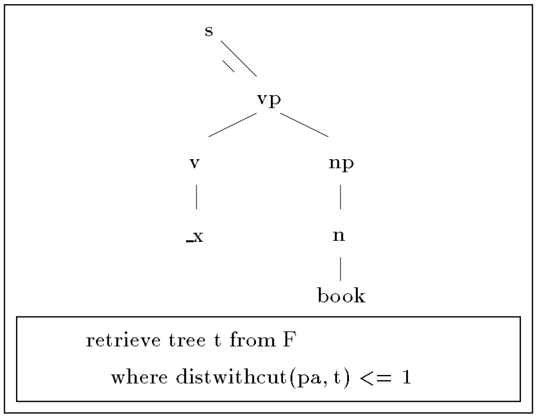


```
retrieve tree t from F
  where distwithcut(pa, t) = 0
```

Fig. 10. ATBE query for finding nouns that are the direct object of "reads".

The above query can be generalized by replacing distwithcut(pa, t) = 0 with distwithcut(pa, t) <= k. We took our inspiration for these operations from the APT system [10], which handles only exact tree matching (i.e., distwithcut(pa, t) must be zero). The extension to approximate tree matching can help in many applications. For example, users may type verbs of the past tense in their pattern tree, even though the matching data tree uses the present tense. Using distwithcut(pa, t) <= k (for k > 0) will find the matching data tree, whereas distwithcut(pa, t) = 0 (as in APT) would not.

At times, computational linguists might want to find the semantic properties of a noun, particularly as determined by the predicates for which it may serve as an argument [26]. Consider, for example, the query "what can be done to a book?" Here the user wishes to get the set of verbs for which "book" is the object from the database. In ATBE, this query can be expressed as shown in Figure 11.

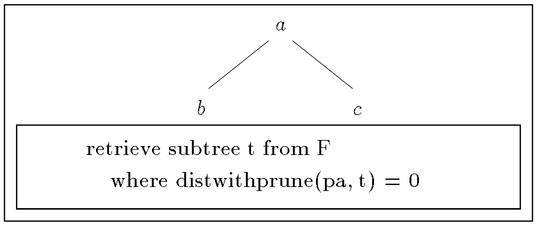


```
retrieve tree t from F
  where distwithcut(pa, t) <= 1
```

Fig. 11. ATBE query for finding verbs for which "book" is the object.

This query illustrates the use of bars. The bar specifies that a path may contain at certain points zero or more unspecified, intermediate nodes. This mark is useful in locating verbs in more complicated sentences such as "The boy wanted to read the book", or "The woman knew that the boy wanted to read the book" [10]. The approximate matching helps locate verbs in sentences where "book" is misspelled or appears in plural form [42].

nonprocedural programming languages such as LISP or LUCID. One important operation in this application is to locate occurrences of a pattern in a subject tree. Figure 9 shows an ATBE query that finds subtrees t in file F that exactly match a given pattern, allowing zero or more prunings at nodes from t. Thus, the query distwithprune(pa, t) = 0 is able to locate portions of a tree that match the pattern in the sense of Hoffmann and O'Donnell. (This can be generalized to *approximate* the Hoffmann and O'Donnell style matching by replacing distwithprune(pa, t) = 0 with distwithprune(pa, t) <= k.)



```
retrieve subtree t from F
   where distwithprune(pa, t) = 0
```

Fig. 9. ATBE query for retrieving subtrees.

Solution subtrees obtained from the query are displayed one at a time; they may also be displayed on a tree basis, namely, the entire tree is displayed with corresponding subtrees highlighted [42].

Like most other query languages [3], [33], ATBE also allows users to store solution (sub)trees in a file, rather than only display them on the screen. For example,

```
retrieve tree t from F into G
     where dist(pa, t) = max ( dist(pa, v) where v is tree of F)
```

stores trees of file *F* that are most dissimilar to (worst matching) the pattern in file G.

## 3.4 Example Queries – Information Extraction

The previous section presents several examples for information retrieval. Another major function of ATBE is to support information extraction from trees. Let us consider some examples drawn from natural language processing.

Consider the tree shown in Figure 1. Suppose the user wishes to find all the nouns that can be the direct object of the verb "reads" from a database of parsed text [10]. He would type the query as shown in Figure 10. This query retrieves data trees that exactly match the pattern, allowing zero or more cuttings at nodes from t. The cut subtrees (nodes) represent *don't-care* parts, i.e., they specify that the given pattern should match a data tree even if that data tree has some additional subtrees, which are thus considered irrelevant with respect to this pattern. The query illustrates the use of variables. Nodes used to instantiate the variable _x represent objects of the verb "reads"; they are highlighted when displaying the mapping between trees [42].

mapping that yields the distance. When a solution tree is large (e.g., contains hundreds of nodes), its edges and nodes are shrunk proportionally, so that the entire tree can fit in the window; users may then lasso the part of interest and zoom in to see more detail [42].[9]

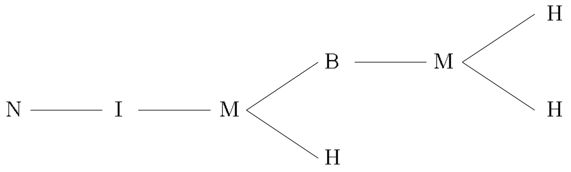


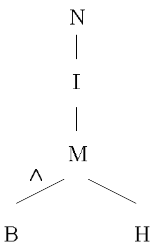
Fig. 7. Horizontal normal form for the tree in Figure 5.

In situations where trees represent noisy information, users might wish to find data trees that are within certain distance of the pattern (this type of retrieval is known as the good-match retrieval [39]). For example, the query

```
retrieve tree t from F
    where dist(pa, t) < 7 and height(t) > 5
```

finds data trees that are within distance 7 of the pattern and whose height is greater than 5.

It is possible that some portion of a data tree is unimportant. In such a situation, the user may provide a pattern containing umbrellas, as shown in Figure 8.



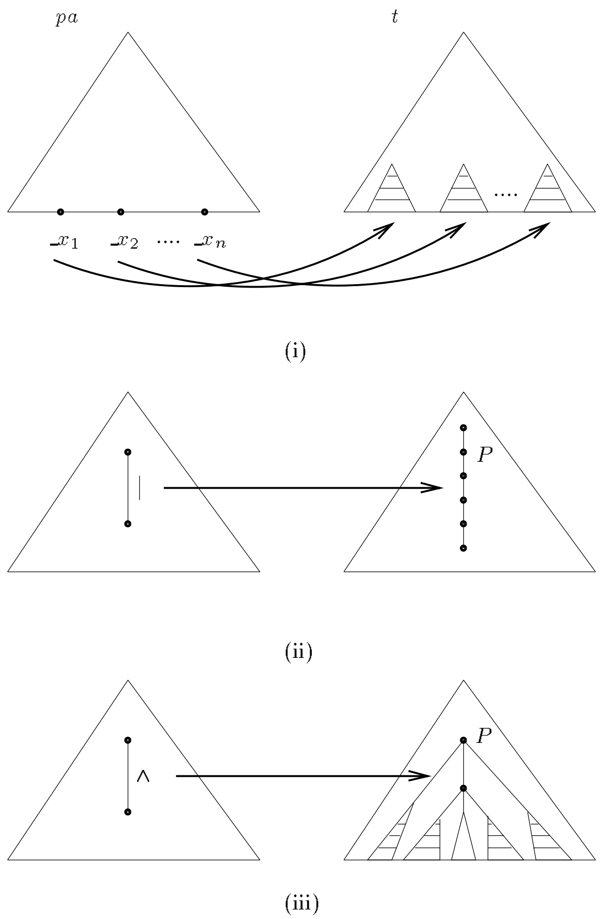
```
retrieve tree t from F
    where dist(pa, t) = 0
```

Fig. 8. ATBE query for retrieving data trees portion of which are unimportant.

This query retrieves data trees consisting of nodes N, I, M and H. The data trees have a subtree rooted at M. The shape of the subtree is unimportant, provided it contains B as one of its leaves.

In some applications, users may wish to retrieve portions of trees, rather than entire trees. Hoffmann and O'Donnell [14], for instance, discussed how to apply the tree replacement techniques to produce interpreters for

[9] When the tree is too large to be in the window, the user can scroll the window up, down, left, and right, to examine one portion of the tree at a time.



(i)

(ii)

(iii)

Fig. 6. (i) Variable instantiation: the variables in *pa* are matched with the shaded subtrees in *t*. (ii) Bar instantiation: the bar is matched with the nodes (black dots) on a path $P$. (iii) Umbrella instantiation: the umbrella is matched with the nodes (black dots) on a path $P$ and the shaded subtrees. Notice that some (consecutive) children along with their descendents of the lowest node of $P$ (represented by the unshaded subtree in $t$) are excluded from the instantiation; they will be mapped to the nodes underneath the umbrella in *pa*.

## 3.3 Example Queries – Information Retrieval

One of the major functions of ATBE is to support (tree) information retrieval. A most commonly used retrieval operation in applications is perhaps to find trees closest to a given pattern.[7] Assuming the pattern is as shown in Figure 5, the query might be expressed as follows:

```
retrieve tree t from F
    where dist(pa, t) = min ( dist (pa, u) where u is tree of F )
```

Trees obtained from the query are displayed one at a time on the screen.[8] The user is able to see the best

[7] For example, in analyzing features of a newly sequenced RNA, there may not exist RNAs in the database that exactly match the new RNA. Under this circumstance, researchers often attempt to get those that are most similar to the new one. This type of query is also known as the best-match retrieval [32].

[8] They are displayed either in *vertical normal form* (as shown in Figure 5) or in *horizontal normal form* (see Figure 7).

- Each variable is matched with (replaced by) a subtree in $t$. (Repeated variables (i.e., different occurrences of the same variable) are matched with identical subtrees.)

- Each bar is viewed as a pseudo node in $pa$, which is matched with part of a path from the root to a leaf of $t$.

- Each umbrella is also viewed as a pseudo node, which is matched with part of such a path and all the subtrees emanating from the nodes of that path, except possibly at the lowest node of that path. At the lowest node, the umbrella is matched with a set of leftmost subtrees and a set of rightmost subtrees. (The mark is so named because of this substitution pattern.) Formally, let the lowest node be $n$ and let the children of $n$ be $c_1, \ldots, c_k$ in left-to-right order. Let $i, j$ be such that $0 \leq i < j \leq k+1$. An umbrella is matched with the subtrees rooted at $c_1, \ldots, c_i$ and $c_j, \ldots, c_k$ in addition to the node $n$, ancestors along a path starting at $n$, and the subtrees of those proper ancestors of $n$.

Let $s(pa)$ be the resulting (mark-free) pattern tree. We require that any mapping from $s(pa)$ to $t$ map the substituting nodes to themselves. (Thus, no cost is induced by mark substitutions.) The distance (distwithcut, distwithprune, respectively) between $pa$ and $t$ with respect to the substitution $s$, denoted $\text{dist}(pa, t, s)$ ($\text{distwithcut}(pa, t, s)$, $\text{distwithprune}(pa, t, s)$, respectively), is defined to be $\text{dist}(s(pa), t)$ ($\text{distwithcut}(s(pa), t)$, $\text{distwithprune}(s(pa), t)$, respectively).[6] The distance (distwithcut, distwithprune, respectively) between $pa$ and $t$ is obtained from one of the best mark-substitutions, i.e.,

$$\text{dist-op}(pa, t) = \min_{s \in \mathcal{S}} \{\text{dist-op}(pa, t, s)\}$$

where $\mathcal{S}$ is the set of all possible mark-substitutions, and dist-op is one of the above distance operators. Intuitively, a query may minimize over substitutions, minimize over cuttings or prunings, and minimize the resulting number of edits.

Notice that the APT [10] and LQL systems [5] also provide similar operations, though their capabilities are much more limited. They only support the following exact match query (the pattern $pa$ can have variables [5] or bars [10], but no umbrellas):

retrieve tree $t$ from <file-name> where $\text{distwithcut}(pa, t) = 0$

In the next subsections, we illustrate the use of ATBE queries using examples drawn from various applications.

[6] Alternatively, $\text{dist}(pa, t, s)$ could be defined as the minimum cost of all sequences of edit operations that transform the nodes (excluding the marks) in $pa$ to the nodes in $t$ which are not involved in the substitution $s$.

- **size**: Computes the total number of nodes in the (sub)tree $t$.
- **height**: Computes the number of edges in a longest path from the root to a leaf of $t$.

There are three kinds of *distance* operators:

- **dist**: Computes the distance between $pa$ and $t$.
- **distwithcut**: Computes the *minimum* distance between $pa$ and $t$, allowing zero or more cuttings at nodes from $t$ (cf. Section 2.3). (There is no cost associated with these cuttings.) Formally, let $someroots(t)$ represent a set of nodes in $t$ where for any two nodes $m, n \in someroots(t)$, neither is an ancestor of the other. Let $cut(t, someroots(t))$ represent the resulting tree after we remove the subtrees rooted at nodes in $someroots(t)$. Then

$$\text{distwithcut}(pa, t) = \min_{someroots(t)} \{ \text{ dist}(pa, cut(t, someroots(t)))\}.$$

- **distwithprune**: Computes the *minimum* distance between $pa$ and $t$, allowing zero or more prunings at nodes from $t$ (cf. Section 2.3). As for distwithcut, there is no cost associated with the prunings.

The expression can be a constant, or an aggregate expression; the latter has the form

```
<aggreg-op> ( <dist-op> ( <pattern>, <iter-var> )
    where <iter-var> is <tree-type> of <file-name> )
```

The *aggregate* operator can be either **min** or **max** (see examples in Section 3.3). The expression is evaluated by binding each (sub)tree in the specified file to the iteration-variable, and then computing the distance between the pattern, which by convention is identified by $pa$, and the (sub)tree (with or without cuttings or prunings). The minimum (or maximum) of these distance values is then returned as the result.

### 3.2.1 Semantics of the Distance Operators

In addition to having constant nodes, namely, ones whose labels and contents are specified, a pattern may contain the following marks:

- variables ($\_x$, $\_y$, etc.);
- bars ( | );
- umbrellas ( $\wedge$ ).

Both the bars and umbrellas appear on edges of the pattern tree. (They are collectively referred to as *variable length don't care* marks [47].) The variables appear on leaves and are preceded with an underscore. These marks may appear in several places in a pattern.

A mark-substitution (instantiation) $s$ on the pattern $pa$ replaces the marks by nodes in the data tree $t$ as follows (Figure 6):

## 3.2 Query Description and Interpretation

Table 1 gives a complete BNF-like syntax of ATBE statements.[5]

| | | |
|---|---|---|
| <ATBE stmt> | := | **retrieve** <tree-type> <tree-var> **from** <file-name> |
| | | [ **into** <file-name> ] [**where** <bool-expr> ] |
| | | \| **insert** <tree-name> **into** <file-name> |
| | | \| **delete** <tree-name> **from** <file-name> |
| <tree-type> | := | tree \| subtree |
| <tree-var> | := | <id> |
| <file-name> | := | <id> |
| <tree-name> | := | <id> |
| <bool-expr> | := | <bool-expr> **and** <bool-expr> |
| | | \| <bool-expr> **or** <bool-expr> |
| | | \| ( <bool-expr> ) |
| | | \| <dist-op> ( <pattern>, <tree-var> ) <arithcmp> <expr> |
| | | \| <tree-op> ( <tree-var> ) <arithcmp> <const> |
| <dist-op> | := | **dist** \| **distwithcut** \| **distwithprune** |
| <tree-op> | := | **size** \| **height** |
| <pattern> | := | *pa* |
| <arithcmp> | := | > \| >= \| = \| ! = \| < \| <= |
| <expr> | := | <const> \| <aggreg> |
| <aggreg> | := | <aggreg-op> ( <dist-op> ( <pattern>, <iter-var>) |
| | | **where** <iter-var> **is** <tree-type> **of** <file-name> ) |
| <aggreg-op> | := | **min** \| **max** |
| <iter-var> | := | <id> |

Table 1. Syntax for the ATBE statements.

In general, ATBE's retrieve statement has the following construct

**retrieve** <tree-type> <tree-var> **from** <file-name> **where** <bool-expr>

The *tree-type* can be either tree or subtree. The *from* clause specifies which file users want to search. The *where* clause imposes constraints on trees, specifying conditions a solution (sub)tree must satisfy. The query is implemented by a search through the specified file in which each data (sub)tree belonging to the file is selected and stored into the tree-variable. Each time that a new (sub)tree is stored in the variable, the boolean expression is evaluated; if the expression is true, the (sub)tree becomes an answer. Each file is treated as a set, and therefore the search through the file is unordered with no (sub)tree selected twice.

A boolean expression consists of terms connected with the logical connectives *and* (for intersection), *or* (for union). Let *pa* refer to the pattern and $t$ a (sub)tree in the file. A term has the form

$$\text{<tree-op>} \ ( \ t \ ) \ \theta \ \text{<const>}$$

or

$$\text{<dist-op>} \ ( \ pa, t \ ) \ \theta \ \text{<expr>}$$

where $\theta$ is a comparison operator (e.g., >, >=, =, ! =, <, <=), and *expression* evaluates to a constant.

There are two kinds of *tree* operators:

[5] There are templates available to save the user some typing when inputing these statements.

LQL [5].

Fig. 5. A query and the screen layout for ATBE; the pattern represents an RNA secondary structure [30]; node contents (e.g., size properties) are displayed via pop-up windows; the string shown in the key-in window is the linear form of the pattern.

of the two operations and their practical applications.

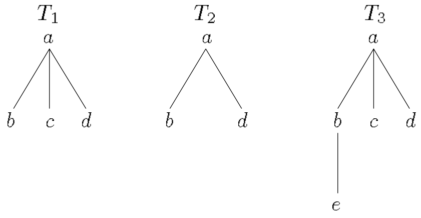


Fig. 4. Example trees.

# 3 ATBE Queries

## 3.1 Query Specification

In ATBE, the user formulates a query by building a pattern tree on the screen, and providing an appropriate statement. In building the tree, the user may draw it from scratch, may edit an existing tree in the database (e.g., the existing tree may be a template), may edit a solution tree of another query,[4] or may key in the tree in its *linear form* directly. The linear form of the tree is a fully parenthesized expression which is a preorder enumeration of the tree (i.e., first the root then the subtrees, from left to right). Node contents (if any) are enclosed in braces, and follow immediately their corresponding node labels.

A statement can be of type **retrieve**, **insert** or **delete**. The first one is for information retrieval and extraction. The second and third are used for modifying the underlying database.

Figure 5 illustrates an ATBE query. The pattern represents an RNA secondary structure drawn from [30]; it is displayed in the *Approximate Tree By Example* window. Node contents are normally not shown on the screen (for saving space purposes), and can be seen through pop-up windows (e.g., the pop-up window associated with the node labeled N indicates that the node's size is 2). The statement is entered in the *Statement* window. Also shown in the figure is the linear form of the pattern, which was keyed in using the text editor.

The query-by-example paradigm employed by ATBE allows rapid and incremental development of queries, which can be easily refined to highlight certain structural properties of trees under investigation. Many systems have used similar concepts in constructing queries [15], [25], [36], [49], [50]. The difference is that, whereas most of these systems express operations in tabular skeletons, ATBE expresses operations in tree structures, which represent the entries in the underlying database. In this sense, ATBE's queries are similar to those of

[4] Using the result of a query as an operand of some other queries is often desired in developing query languages for advanced information systems [2]. The motivation for having this property in ATBE is that, at times users may find that one solution tree is promising and closely matching other trees in some file. In such a situation, he may edit the solution tree and then use it as a new pattern to search for data trees in that file.

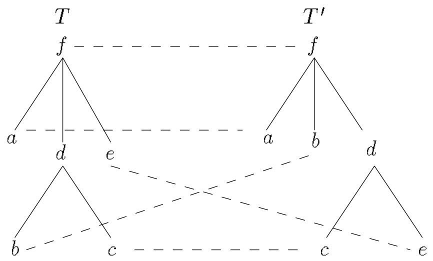


Fig. 3. A mapping from $T$ to $T'$. A dotted line from a node $u$ in $T$ to a node $v$ in $T'$ indicates that $u$ should be changed to $v$ if $u \neq v$, or that $u$ remains unchanged if $u = v$. The nodes of $T$ not touched by a dotted line are to be deleted and the nodes of $T'$ not touched are to be inserted.

Thus, the mapping in Figure 3 is $\{(1, 1), (2, 2), (4, 3), (5, 5), (6, 6)\}$.

Let $M$ be a mapping from $T$ to $T'$. Let $I$ and $J$ be the sets of nodes in $T$ and $T'$, respectively, not touched by any dotted line in $M$. Then we can define the cost of $M$:

$$\gamma(M) = \sum_{(i,j)\in M} \gamma(T[i] \to T'[j]) + \sum_{i\in I} \gamma(T[i] \to \Lambda) + \sum_{j\in J} \gamma(\Lambda \to T'[j]).$$

Given $S$, a sequence of edit operations from $T$ to $T'$, it can be shown that there exists a mapping $M$ from $T$ to $T'$ such that $\gamma(M) \leq \gamma(S)$; conversely, for any mapping $M$, there exists a sequence of edit operations $S$ such that $\gamma(S) = \gamma(M)$ [Lemma 2, 46].

Hence, we have

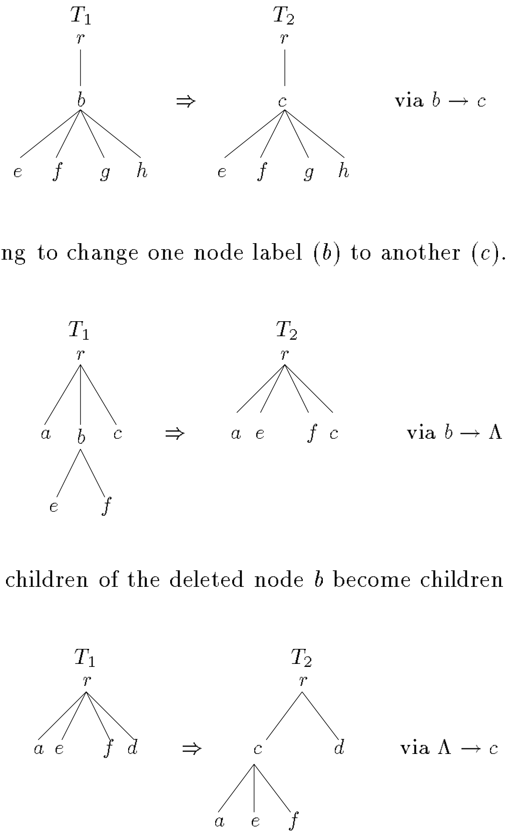
$$dist(T, T') = \min \, \{\gamma(M) \mid M \text{ is a mapping from } T \text{ to } T'\}.$$

## 2.3 Approximate Tree Matching Operations

In [37], Ukkonen discussed approximate string matching operations that allow prefixes of strings to be removed when comparing strings. Myers and Miller [21] discussed similar operations for regular expressions. We extend these operations to trees by considering prefixes as a collection of subtrees. The following two operations are introduced:

- *Cutting at node $n$ from tree $T$* means removing $n$ and all its descendants (i.e., removing the subtree rooted at $n$).
- *Pruning at node $n$ from tree $T$* means removing only the descendants of $n$; $n$ itself remains in $T$. (Thus, a pruning never eliminates the entire tree.)

The operations are useful in locating portions of a tree that closely match a given pattern. Consider, for example, the trees in Figure 4. $T_1$ exactly matches the subtree rooted at $a$ in $T_3$ if we prune at node $b$ from $T_3$ (or cut at node $e$). (This type of subtree matching corresponds to the one defined in Hoffmann and O'Donnell [14].) Note that there may not exist applicable pruning operations for certain matchings yielded by cuttings. For example, by cutting at node $c$ and node $e$ from $T_3$, the resulting tree matches $T_2$. However, no pruning operation can be applied in this case to yield such a matching. In Section 3, we shall further discuss the use



(i) Relabeling to change one node label ($b$) to another ($c$).

(ii) Deletion of a node. (All children of the deleted node $b$ become children of the parent $r$.)

(iii) Insertion of a node. (A consecutive sequence of siblings among the children of $r$ (here, $a$, $e$ and $f$) become the children of $c$.)

Fig. 2. Examples illustrating the edit operations.

## 2.2 Mappings

The edit operations correspond to a *mapping* that is a graphical specification of which edit operations apply to each node in the two trees. The mapping in Figure 3 shows a way to transform $T$ to $T'$. It corresponds to the sequence (delete (node with label $d$), insert (node with label $d$)).

Let $T[i]$ represent the $i$th node of tree $T$ according to some ordering (e.g., preorder). Formally, a mapping from $T$ to $T'$ is a triple $(M, T, T')$ (or simply $M$ if there is no confusion), where $M$ is any set of pairs of integers $(i, j)$ satisfying the following conditions:

1. $1 \leq i \leq |T|$, $1 \leq j \leq |T'|$, where $|.|$ represents the number of nodes in the indicated tree;

2. For any pair of $(i_1, j_1)$ and $(i_2, j_2)$ in $M$,

   (a) $i_1 = i_2$ if and only if $j_1 = j_2$ (one-to-one);

   (b) $T[i_1]$ is to the left of $T[i_2]$ if and only if $T'[j_1]$ is to the left of $T'[j_2]$ (sibling order preserved);

   (c) $T[i_1]$ is an ancestor of $T[i_2]$ if and only if $T'[j_1]$ is an ancestor of $T'[j_2]$ (ancestor order preserved).

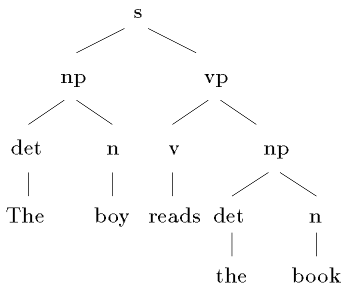


Fig. 1. Parse tree representing the sentence "The boy reads the book" [10].

Many algorithms have been developed for exact tree comparison [14], [17]. Our system is based on the inexact algorithms presented in [45], [46], [47]. The algorithms are a generalization of those used for determining the editing distance between sequences [4]. There are three types of edit operations: *relabel*, *delete* and *insert* a node. We represent these operations as $u \rightarrow v$, where each of $u$ and $v$ is either a node or the null node ($\Lambda$). We call $u \rightarrow v$ a relabeling operation if $u \neq \Lambda$ and $v \neq \Lambda$; a delete operation if $u \neq \Lambda$ and $v = \Lambda$; an insert operation if $u = \Lambda$ and $v \neq \Lambda$. Let $T_2$ be the tree that results from the application of an edit operation $u \rightarrow v$ to tree $T_1$; this is written $T_1 \Rightarrow T_2$ via $u \rightarrow v$. Figure 2 illustrates the edit operations.

Let $S$ be a sequence $s_1, s_2, \ldots, s_k$ of edit operations. $S$ transforms tree $T$ to tree $T'$ if there is a sequence of trees $T_0, T_1, \ldots, T_k$ such that $T = T_0, T' = T_k$ and $T_{i-1} \Rightarrow T_i$ via $s_i$ for $1 \leq i \leq k$.

Our definition of edit operations is really a shorthand for the specification. Here is the specification in full detail. Consider a single edit operation, e.g., one that transforms $T_{i-1}$ to $T_i$. If it is a relabeling operation, we specify the node to be relabeled in $T_{i-1}$. The same holds for a delete operation. An insert operation is a little more complicated: we must specify the parent $p$ of the node $n$ to be inserted and which consecutive sequence of siblings among the children of $p$ will be the children of $n$. If that consecutive sequence is empty, then we need to specify the position of $n$ among the children of $p$. However, we will continue to use our shorthand because these other specifications will be clear from the mapping structure defined below.

Let $\gamma$ be a cost function that assigns to each edit operation $u \rightarrow v$ a nonnegative real number $\gamma(u \rightarrow v)$. We constrain $\gamma$ to be a distance metric. That is, it satisfies the following three properties: $\gamma(u \rightarrow v) \geq 0$ and $\gamma(u \rightarrow u) = 0$ (non-negative definiteness); $\gamma(u \rightarrow v) = \gamma(v \rightarrow u)$ (symmetry); $\gamma(u \rightarrow w) \leq \gamma(u \rightarrow v) + \gamma(v \rightarrow w)$ (triangle inequality).

We extend $\gamma$ to a sequence of edit operations $S = s_1, s_2, \ldots, s_k$ by letting $\gamma(S) = \sum_{i=1}^{k} \gamma(s_i)$. The editing distance, or simply the *distance*, from tree $T$ to tree $T'$, denoted $dist(T, T')$, is defined to be the minimum cost of all sequences of edit operations which transform $T$ to $T'$, i.e.,

$$dist(T, T') = \min \{\gamma(S) \mid S \text{ is a sequence of edit operations transforming } T \text{ to } T'\}.$$

The definition of $\gamma$ makes *dist* a distance metric as well.

misspelling words (terminals) in the trees.[2]

This paper presents an inexact tree matching system, called Approximate-Tree-By-Example (ATBE), developed at New York University. ATBE is designed to support constructing, comparing and querying sets of trees. Given a pattern (tree), the system allows users to retrieve (approximately) matched trees to the pattern from a database, or extract information from trees pertinent to the pattern.[3]

ATBE has many salient features.

1. It can support a wide variety of applications: ATBE provides a query language for tree comparison based on a relational database language [33]. The system is customizable. Users can tailor the system to meet the needs of their applications by inserting application-specific trees and application-specific distance metrics and operations.

2. It manipulates trees in a uniform manner: There are numerous ways to represent or describe a tree [27], [43]. However, once a tree is input, ATBE manipulates, stores, and displays them in standardized forms.

3. It is user-friendly: ATBE provides substantial graphical devices to facilitate users to input queries. It gives users flexibility to edit trees at any time instead of using templates. The system has a multiple window display that makes effective use of the screen, and utilizes on-screen and pop-up menus as alternatives for typing most commands.

4. It is machine-independent: ATBE is implemented in C and X-windows [44]. The X-based implementation permits the system to be used on a variety of workstations.

The paper gives an overview of the ATBE system. In Section 2, we introduce terminology and background for tree comparison. Section 3 presents the query language. Section 4 describes the system's architecture. Section 5 discusses some underlying algorithms. In two companion papers, [41] and [42], we describe in detail the graphical interface and the use of the system.

# 2 Background

## 2.1 Edit Operations

The trees we are concerned with are ordered, labeled ones. Each node in a tree has a label and possibly some additional information. (This information is referred to as node contents.) Node contents could be size properties, like those in RNA secondary structures [30], or lexical features for grammar parses [10]. Figure 1 illustrates a grammar parse representing the sentence "The boy reads the book."

[2] The importance of dealing with this type of inexact matching in actual applications has been widely addressed in the literature. See, e.g., [11], [16], [31].

[3] We include the term *approximate* in naming the system for two reasons. First, it states that the system can perform inexact tree matching, i.e., it allows certain inaccuracy or dissimilarity to exist when comparing trees. Second, the approximate string matching operation, which allows a prefix of strings to be removed when comparing strings, is important in many applications [18], [37]. ATBE has analogous operations, allowing certain subtrees to be removed when comparing trees. The term *by-example* refers to the way users query the database, which will be described in Section 3.

# 1 Introduction

Ordered, labeled trees are trees in which each node has a label and the left-to-right order of its children (if it has any) is fixed.[1] Such trees have many applications in vision, molecular biology, programming compilation and natural language processing, including the representation of images [29], patterns [7], [13], [20], intermediate code [1], [14], grammar parses [10], [35], dictionary definitions [5], [23] and secondary structures of RNA [30]. They are frequently used in other disciplines as well.

Many of the above applications involve comparing trees or retrieving/extracting information from repositories of trees. For example,

- In molecular biology, researchers collect vast amounts of RNA structures (trees) whose features have been analyzed. To gain information about a newly sequenced RNA, they compare the RNA's structure against those in the database, searching for ones with very similar topologies. From such topological similarities, it is often possible to infer similarities in the functions of the related RNAs [30], [32], [34].
- In natural language processing, computational linguists store dictionary definitions in a lexical database. The definitions are represented syntactically as trees. Because the syntactic head of a definition is often the genus term (superordinate) of the word being defined [9], linguists extract semantic information from syntactic analysis of these definitions, thereby constructing semantic taxonomies [10], [22].
- In pattern recognition, a commonly used technique to classify an unknown pattern (tree) is to compare it against those in the data sets, and to assign it to the category to which the majority of its closest patterns belong [12].
- In programming languages, one effective way used to select an error recovery or correction has been to compare the parse trees associated with corrected strings and their replacements, as well as the corresponding strings [35].

Whereas ordered, labeled trees have been widely used in different applications, very few systems have been built to support their comparison, and the information retrieval/extraction from repositories of such trees. As far as we know, there are only two systems, both developed at IBM, that support such operations: APT [10] and LQL [5]. APT is designed to extract linguistic information from a corpus of parse trees. It allows the user to mark a target node in a parse tree, and then automatically constructs a partially instantiated PROLOG term which serves as a pattern for finding nodes that occupy a comparable structural location in other parse trees. LQL, on the other hand, provides users with a template tree structure, which represents a superset of all possible structures that individual entries in a lexical database have. Through the template, users can query, maintain, and extract information from the database. However, since both of the systems employ unification techniques for tree matching, their capabilities are limited to exact matches. They cannot be used, for example, to extract information from trees that are noisily represented, possibly caused by mistyping or

[1] Throughout the paper, we shall refer to ordered trees simply as trees when no ambiguity occurs.

# A System for Approximate Tree Matching[*]

Jason Tsong-Li Wang    Kaizhong Zhang    Karpjoo Jeong    and    Dennis Shasha[†]

April 28, 1992

**Abstract**

Ordered, labeled trees are trees in which each node has a label and the left-to-right order of its children (if it has any) is fixed. Such trees have many applications in vision, pattern recognition, molecular biology, programming compilation and natural language processing. Many of the applications involve comparing trees or retrieving/extracting information from a repository of trees. Examples include classification of unknown patterns, analysis of newly sequenced RNA structures, semantic taxonomy for dictionary definitions, generation of interpreters for nonprocedural programming languages, and automatic error recovery and correction for programming languages.

Previous systems use exact matching (or generalized regular expression matching) for tree comparison. This paper presents a system, called Approximate-Tree-By-Example (ATBE), which allows inexact matching of trees. The ATBE system interacts with the user through a simple, but powerful query language; graphical devices are provided to facilitate inputing the queries. The paper describes the architecture of ATBE, illustrates its use and describes some aspects of ATBE implementation. We also discuss the underlying algorithms, and provide some sample applications.

**Index Terms** – Editing distance, graphics, query language, query processing, pattern matching, tool, trees, tree comparison.

[*]This work was supported in part by the National Science Foundation under Grants IRI-8901699 and CCR-9103953, by the Office of Naval Research under Grants N00014-90-J-1110 and N00014-91-J-1472, by the New Jersey Institute of Technology under Grant No. 421280, and by by the Natural Sciences and Engineering Research Council of Canada under Grant OGP0046373.

[†]Affiliation of authors: J. T. L. Wang was with the Courant Institute of Mathematical Sciences, New York University, 251 Mercer Street, New York, NY 10012. He is now with the Department of Computer and Information Science, New Jersey Institute of Technology, University Heights, Newark, NJ 07102. K. Zhang is with the Department of Computer Science, Middlesex College, The University of Western Ontario, London, Ontario, Canada N6A 5B7. K. Jeong and D. Shasha are with the Courant Institute of Mathematical Sciences, New York University, 251 Mercer Street, New York, NY 10012.