Data cleaning is an inevitable problem when integrating data from distributed operational databases, because no unified set of standards spans all the distributed sources. One of the most challenging phases of data cleaning is removing fuzzy duplicate records. Earlier approaches, which required hard coding rules based on a schema, were time consuming and tedious, and you couldn’t later adapt the rules. We propose a novel duplicate-elimination framework that lets users clean data flexibly and effortlessly, without any coding.

Finding fuzzy duplicates

Four types of data problems generally cause differences between tuples: incomplete data (such as data with missing fields), incorrect data (which might have spelling errors or invalid or wrong codes), incomprehensible data (possibly caused by word transposition or unknown codes), or inconsistent data (comprising inconsistent formats or abbreviations).

Table 1 demonstrates some of these discrepancies using sample tuples.

We attempt to address these types of problems when cleaning and preprocessing data. One of the last stages of the data-cleaning process is to eliminate fuzzy duplicates. However, because any combination of discrepancies can exist between two tuples, the discrepancies can be ambiguous and difficult to detect.

If we want to use a human to detect fuzzy duplicates, we must assign a domain expert who’s familiar with the table schema and with the semantic interpretation of the tuple’s attributes. This person must compare the tuples based on his or her expertise and conclude whether the two tuples refer to the same entity. However, considering the uncertainties involved, this is sometimes difficult—even for a human expert.

It’s more practical to automate this cumbersome task by replacing the human with an expert system. Generally, we use an equational theory to compare two tuples and infer the probability of them being the same, following the sorted neighborhood method. In the SNM, we create a key for each tuple and sort the tuples using that key. The sort operation clusters the
duplicates, and then a window of size $w$ slides over the sorted data and the tuple entering the window is compared to all the tuples in the window.

**Our framework**

Figure 1 demonstrates our framework’s workflow. We first preprocess and clean the data. Then we start the six-step duplicate-elimination process, where the user selects

1. a clustering algorithm,
2. attributes for comparing a pair of tuples,
3. corresponding similarity functions for measuring attribute similarity,
4. fuzzy rules used in the fuzzy inference engine,
5. membership functions, and
6. a merging strategy.

In each step, the user can select an existing “method” (that is, an existing algorithm, similarity function, and so forth) or can implement his or her own method and add it to the library.

In step 1, the user selects an algorithm to cluster the cleaned tuples, grouping those that are most likely to be duplicates. We view the SNM’s key creation and sorting and sliding window phases as a clustering algorithm. Users can also employ other existing clustering algorithms.

In step 2, the user selects the attributes that are important when comparing two records. The system selects all possible pairs from each cluster and compares records within each cluster, using these attributes.

In step 3, the user chooses a specific similarity function for each selected attribute from a library. The user should choose the function according to the attribute’s data type and domain—for example, numerical, string, or domain-dependent functions for address, surname, and so forth. Each function measures the similarity of an attribute in a pair of tuples. In this way, the user can easily integrate any novel similarity function into the system. (We discuss steps 4 and 5, which involve machine learning and adaptation, in later sections.)

In step 6, the detected duplicates are merged. Different merging strategies exist, so the user must decide what tuple to use as the prime representative of the duplicates—the tuple with the fewest empty attributes, the most recent one, and so on.

The system records the merged tuples and their prime representatives in a log. It also saves the fuzzy inference engine’s input and output for the detected duplicates. This information helps the user review and verify the changes in the duplicate-elimination process. Using the framework’s rule viewer, the user can use the input and output to fine-tune the system’s rules and membership functions (MFs) automatically or by hand (explained later). Note that the framework is extensible and users can implement and add various modules to the system for steps 1, 3, and 6.

**Acquiring an expert’s knowledge**

An example of a fuzzy rule that describes a simple fact is

$$\text{If pressure is high, then volume is small,}$$

where `pressure` and `volume` are linguistic variables and `high` and `small` are linguistic terms that are characterized by MFs. Owing to the concise form of fuzzy if-then rules, such rules are often employed to capture the imprecise modes of reasoning that play an essential role in the human ability to make decisions in an uncertain and imprecise environment. We partition the variables in terms of natural language linguistic terms. This linguistic partitioning, an inherent feature of what Lotfi A. Zadeh calls “computing with words,” greatly simplifies model building. Linguistic terms—the words we speak in everyday linguistic reasoning—represent fuzzy subsets over the corresponding variable’s domain.

In our framework, rules specify the criteria for determining if two tuples are duplicates. The rules effectively capture the user’s knowledge, which is required in the decision-making process (step 4). (The user is always an expert who has domain knowledge about tuple attributes and their interpretation.)

In contrast to our framework’s fuzzy reasoning approach, the coding of conditions and declarative rules for comparing tuples proposed in previous literature is particularly difficult and time consuming, and the coding must be repeated for each table schema.\(^2\)\(^5\)

Even then, we must use trial and error to set the thresholds for rules in the code to achieve acceptable performance.

The fuzzy reasoning approach provides a fast, intuitive way of letting the user define the rules using natural language, with the aid of a simple GUI. It also eliminates the repetitive hard-coding process. An example of a rule in this system is

$$\text{If (Id-Similarity is high) \land (Name-Similarity is high) \land (Address-Similarity is low), then } z = 0.9.$$  

The antecedent part of the rule can include a subset (or all) of the attributes, which the user selects in step 2. The rule’s consequence or output ($z$) is a crisp value and represents the probability of two tuples being duplicates.

When the user is entering the rules, setting a value for $z$ might seem difficult, but it’s still easier than with previous methods, where defining the rules, setting thresholds, and setting certainty factors for rules and other parameters required programming.\(^5\) Additionally, our approach alleviates parameter-tuning problems by using an automatic learning pro-

---

Table 1. Examples of discrepancies in database tuples.

<table>
<thead>
<tr>
<th>Discrepancy</th>
<th>Name</th>
<th>Address</th>
<th>Phone number</th>
<th>ID number</th>
<th>Date of birth</th>
<th>Gender</th>
</tr>
</thead>
<tbody>
<tr>
<td>Spelling error</td>
<td>John Doe</td>
<td>Lucent Labs.</td>
<td>615 5544</td>
<td>553066</td>
<td>07 07 1970</td>
<td>Male</td>
</tr>
<tr>
<td>Abbreviation</td>
<td>J. Dow</td>
<td>Lucent Lab.</td>
<td>615 5544</td>
<td>553066</td>
<td>07 07 1970</td>
<td>Male</td>
</tr>
<tr>
<td>Missing fields</td>
<td>John Dow</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>07 07 1970</td>
<td>Male</td>
</tr>
<tr>
<td>Inconsistent format</td>
<td>John Dow</td>
<td>Lucent Labs.</td>
<td>(021)6155544</td>
<td>553066</td>
<td>07 July 1970</td>
<td>1</td>
</tr>
<tr>
<td>Word transposition</td>
<td>Dow John</td>
<td>Lucent Labs.</td>
<td>615 5544</td>
<td>553066</td>
<td>07 07 1970</td>
<td>Male</td>
</tr>
</tbody>
</table>
procedure that can mold the MFs’ shapes and the rules’ output (z) according to the training data. As you’ll see, the user decisions in steps 4 and 5 aren’t critical when adaptation is an option.

The Sugeno method of inference

A fuzzy-system model is basically a knowledge-based representation of the functional relationship between antecedent variables and a consequent variable. In the Mamdani method of inference,6 the output variable is also a fuzzy subset—for example, a rule ends with something like \( z = \text{very high} \). However, a crisp output value, as shown by the example in the previous section, lets us use the Sugeno method of inference,7 a simplified version of the Mamdani method. When using the Sugeno fuzzy if-then rules, each rule’s output is a linear combination of the input variables or a constant term, and the final output is the weighted average of each rule’s output.

If the fuzzy inference system’s final output is above a certain threshold, we classify the tuples as duplicates. The threshold’s actual and appropriate value should be determined according to the rules’ z-values, as set previously. However, the learning will shape the MFs and the rules’ z-values, such that the duplicates fall above the threshold for a given set of rules and training data.

We can explain the inference mechanism with a simplified example. Assume that we have only three rules:

1. If (Id-Similarity is high) \( \land \) (Name-Similarity is high), then \( z = 0.95 \)
2. If (Id-Similarity is high) \( \land \) (Name-Similarity is low), then \( z = 0.90 \)
3. If (Id-Similarity is low) \( \land \) (Name-Similarity is low), then \( z = 0.85 \)

The user can easily specify these rules in natural language. Figure 2 shows the linguistic terms (low, high) and their corresponding MFs for the input variables. Figure 3 illustrates how the Sugeno method of inference works for determining the duplicates. The system feeds the input vector \( I = (0.62, 0.65) \), which represents Id-Similarity and Name-Similarity, respectively, into the inference engine. It then calculates the weighted average for the three rules’ consequences, and the probability of the tuples being duplicates (output variable) is \( z = 0.94 \). If the threshold is set at 0.9, the tuples are classified as duplicates. As figure 3 shows, the system’s rule viewer lets the user see each rule’s exact effect for every input and provides an explanation facility.

To formalize this, consider that the ith rule is

\[
\lambda_i = \min_{j=1,\ldots,p} \left\{ A_{ij}(x_j^*) \right\}
\]

Here, \( A_{ij}(x_j^*) \) is the membership grade of \( x_j^* \) in the fuzzy subset \( A_{ij} \).
2. Calculate the model’s unique output as a weighted average of the firing levels and the consequents,

\[ y^* = \frac{\sum_{i=1}^{n} \lambda_i b_i}{\sum_{i=1}^{n} \lambda_i} \]

**Adaptation and learning capabilities**

The benefits of the Sugeno method of inference are computational efficiency and suitability for adaptive techniques. The inference produces a unique value for the consequent using a simple computation. In our framework, the only tricky part for the user is determining the inference engine’s fuzzy rules and MF shapes. By exploiting neuro-fuzzy techniques\(^8\) on top of the Sugeno method of inference, we can train the framework using the available numerical data, virtually abolishing the need for human intervention. This also provides adaptation and learning capabilities for the framework and enhances the results.

We use fuzzy modeling to construct a fuzzy inference system (FIS). This process lets us

- integrate human knowledge and expertise about the decision-making process into the system’s structure determination. Most modeling methods don’t use domain knowledge. (Structure determination includes determining the relevant inputs, number of MFs for each input, number of rules, and fuzzy model type—such as Mamdani or Sugeno).
- employ other conventional system-identification methods when input and output data for the system to be modeled are available. Neuro-fuzzy modeling is a way of applying learning technique, developed in the neural networks literature, to parameter identification of a FIS. Parameter identification deals with recognizing MF shapes and rules output to optimize performance.

Employing the Adaptive Network-based Fuzzy Inference System (ANFIS)\(^9\) molds the MFs into shape and fine-tunes the consequence of the rules to model the training data set more closely.\(^8\) This method proposes a five-layered adaptive network (which is functionally equivalent to a first-order Sugeno fuzzy model) and trains the network (that is, the fuzzy model).

In essence, the spirit of an FIS is “divide and conquer”—that is, the antecedents of fuzzy rules partition the input space into a number of local fuzzy regions, while the consequents describe the behavior in a given region. In our experiments, we used grid partitioning\(^10\) and subtractive clustering\(^11\) to determine the initial structure of the FIS and partitioning of the problem space before applying ANFIS for learning and fine-tuning the parameters. Grid partitioning uses similar, symmetric MFs for all the input variables to generate equal partitions, without clustering. The subtractive clustering method partitions the data into groups called clusters and generates an FIS with the fewest rules required to distinguish the fuzzy qualities associated with each cluster. When we use adaptation for parameter identification and one of the two methods mentioned—that is, grid partitioning or subtractive clustering—for initial FIS structure determination, the user doesn’t need to specify anything in steps 4 and 5. Optionally, we can

---

*Figure 3. The Sugeno method of inference for determining the duplicates. The input vector is \( I = (0.62, 0.65) \), which represents Id-Similarity and Name-Similarity, respectively, producing the output variable, \( z = 0.94 \), using the three rules mentioned in the main text.*
specify the rules and MFs by hand and apply learning on top of that, or not apply learning at all.

We employ two methods to update the MF parameters in ANFIS learning: back-propagation (BP) and a hybrid method. We use BP for all parameters (a steepest-descent method). The hybrid method consists of BP for the parameters associated with the input membership functions and least-squares estimation for the parameters associated with the output MFs. As a result, the training error decreases in each fuzzy region, at least locally, throughout the learning process. Therefore, the more the initial MFs resemble the optimal ones, the easier it will be for the parameter training to converge.

In previous methods, the static rules and conditions were hard-coded using C or other programming languages and didn’t have any learning capabilities (see the “Related Work in Eliminating Duplicates” sidebar). Determining the thresholds for the rules and other parameters (such as for a certainty factor) was done purely through trial and error. However, our approach exploits fuzzy logic to remove coding from the equational theory. The other crucial advantage is the inclusion of machine learning capabilities. In fact, with training, the MF shapes and the rules (z-values) are tuned so that the fuzzy system best models the data. Therefore, the system adapts to the specific notion of similarity based on the problem domain, using the training examples provided.

Let us elaborate on why we use fuzzy logic. Besides the inherent handling of uncertainty in duplicate elimination and getting rid of programming, it’s extremely valuable that, even in cases when numerical data for training is unavailable, we can still employ the framework. Without any training, the user can achieve good performance by defining a few basic MF shapes and simple common-sense rules, which represent the user’s domain knowledge. Sample shapes were illustrated in figure 2. The upper bound for the number of possible fuzzy rules is limited to $a^{a \cdot MF}$, where $a$ is the number of attributes used in comparison and MF is the number of MFs per attribute.

The rules are very intuitive in the Mamdani method of inference—they don’t require a z-value. Although you might be able to use other learning mechanisms, none are likely to be so accommodating and user friendly, letting the user operate with or without training data.

Also, our framework learns at a metalevel to capture the specific notion of record similarity, which is the quantity we must measure to detect fuzzy duplicate records. This involves more than just developing trainable similarity functions for specific types of fields or for domain-independent similarity functions for strings. In fact, this framework lets the user employ any complex existing learnable string similarity measure.

Performance and adaptation

To implement the fuzzy duplicate-elimination framework, we used the Borland C++ Builder Enterprise Suite and Microsoft SQL Server 2000. The data resided in relational database tables and was fetched through ActiveX Data Object components. We employed the Data Transformation Service of MS SQL Server to load the data into the Object Linking and Embedding DB Provider. Our hardware was a Pentium 4 (1.5 GHz) PC with 256 Mbytes of RAM and the Windows XP operating system.

The data set we used comprised segmented census records with five attributes, which W.E. Winkler originally gathered. The data resulted from integrating two different sources, each with duplicates as well as other inconsistencies. Here, we evaluate the framework’s general performance and the effect of adaptation on MF shapes and rules.

We used the basic SNM to cluster records. A user selected and employed four out of five available tuple attributes—last name, first name, code, and address—in the inference process. We used a very simple attribute similarity function in our implementation, which matched only the characters in the two fields and correspondingly returned a value between zero and one. However, adding smarter, more sophisticated domain-dependant similarity functions (for example, functions that handle abbreviations or check addresses) would only improve the final results.

Without learning

Using a GUI for steps 4 and 5, we had a user define two simple bell-shaped hand-drawn MFs, representing the “low” and “high” terms for the four input variables (as figure 2 showed). This lets the user define a maximum of 4² rules. Humans find it easier to state rules that have fuzzy output variables (rather than crisp ones), as in the Mamdani method. Hence, using this method, the user defines the output variable with three linguistic terms—low, medium, and high (see figure 4). The user added 11 simple rules in natural language, similar to the following, with the aid of a GUI:

- If (LastNameSimilarity is low) ∧ (FirstNameSimilarity is high) ∧ (CodeSimilarity is high) ∧ (AddressSimilarity is high), then (z is medium)
- If (LastNameSimilarity is low) ∧ (FirstNameSimilarity is low), then (z is low)

To evaluate the approach’s performance and adaptation effectiveness, we measured recall and precision. Recall is the ratio of the number of retrieved duplicates to the total number of duplicates. False-positive error ($FP_e$) is the ratio of the number of wrongly identified duplicates to the total number of identified duplicates. Precision is equal to 1 – $FP_e$. Obviously, the system performs better with higher precision at a given recall rate.

Employing learning

Figure 5 shows the precision-recall curves for five FISs. In all cases, we used two linguistic terms (MFs) for each input variable. The best hand-drawn curve represented the user-defined FIS as just described, which achieved the best results in experimenting with several bell-shaped MFs with various crossing points for the two terms. We produced the other FISs using grid partitioning with different MF shapes (such as bell or Gaussian), on top of which we applied ANFIS learning (such as BP or a hybrid method). When using learning, the initial FIS structure was formed using grid partitioning, so the user doesn’t have to specify any rules or MFs. All the trained FISs performed better than
Table A summarizes and compares various data cleaning approaches. We've integrated Mauricio Hernandez's sorted neighborhood method\(^1\) into our framework. Alvaro Monge proposes a priority queue to reduce the number of pair-wise comparisons of tuples.\(^2\) Helena Galhardas presents an execution model, declarative language and algorithms, similar to SQL commands, to express data cleaning specifications and perform the cleaning efficiently.\(^3\) In contrast to our system, these rules are static and hard to code and manipulate.

Vijayshankar Raman describes an interactive data cleaning system that lets users see the changes in the data with the aid of a spreadsheet-like interface.\(^4\) It uses gradual construction of transformations through examples using a GUI but is somewhat rigid. The detection of anomalies is by human visual inspection, which is problematic. The knowledge-based approach introduced by Wai Lup Low is similar to our work, in the sense of representing user's knowledge, but it uses programmatic rules.\(^5\) Low doesn't employ fuzzy inference and uses a certainty factor for the rules. Mikhail Bilenko employs learning to improve the string similarity function for a specific domain, using training examples.\(^6\) The table demonstrates how our framework is easy to use, extensible, adaptive, and accommodative.

Nowadays, researchers are also encountering similar problems in the context of the Semantic Web when constructing and merging ontologies.\(^7\) One proposed solution is a stepwise process for establishing referential identity in ontology merging.\(^8\)

### Related Work in Eliminating Duplicates

Table A. Comparison of data cleaning approaches.

<table>
<thead>
<tr>
<th>Method for acquiring domain/expert knowledge</th>
<th>Hernandez(^1)</th>
<th>Monge(^2)</th>
<th>Galhardas(^3)</th>
<th>Raman(^4)</th>
<th>Low(^5)</th>
<th>Bilenko(^6)</th>
<th>Our framework</th>
</tr>
</thead>
<tbody>
<tr>
<td>Coded rules</td>
<td>None</td>
<td>SQL-like syntax</td>
<td>Using a GUI</td>
<td>Coded rules</td>
<td>Learning</td>
<td>Natural language fuzzy rules, learning</td>
<td></td>
</tr>
<tr>
<td>Flexibility of process/rule manipulation (requires coding)</td>
<td>Low</td>
<td>None</td>
<td>Medium</td>
<td>High</td>
<td>Low</td>
<td>None</td>
<td>High</td>
</tr>
<tr>
<td>Computational efficiency</td>
<td>SNM</td>
<td>Priority queue</td>
<td>SQL query optimization</td>
<td>Optimizing transformation code</td>
<td>SNM, Rete algorithm</td>
<td>Clustering</td>
<td>SNM, clustering Sugeno inference</td>
</tr>
<tr>
<td>Duplicate detection method</td>
<td>Rules</td>
<td>None</td>
<td>None</td>
<td>Human visual inspection</td>
<td>Rules</td>
<td>None</td>
<td>Fuzzy rules</td>
</tr>
<tr>
<td>Uncertainty handling</td>
<td>Low</td>
<td>None</td>
<td>None</td>
<td>None</td>
<td>Medium (certainty factor)</td>
<td>Medium (with learning)</td>
<td>High (inherent in fuzzy logic and with learning)</td>
</tr>
<tr>
<td>Extensibility for employing new similarity functions/clustering algorithms</td>
<td>Low</td>
<td>None</td>
<td>Low</td>
<td>Low</td>
<td>Low</td>
<td>None</td>
<td>High</td>
</tr>
<tr>
<td>Adaptation and learning capabilities</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Can operate with or without training data (so it's accommodative)</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

### References


the FIS using hand-drawn MFs and user-defined rules (labeled “best hand-drawn” in figure 5). Hybrid learning on Gaussian MFs showed the best performance, achieving 10 to 20 percent better precision at a given recall rate compared to the best hand-drawn curve.

To compare our framework to previous approaches (see the “Related Work” sidebar), we coded a set of nine rules (according to the table schema) and tried to use rational thresholds in the rules. Predictably, although coding the rules took many hours, the first version of the rules didn’t perform well and detected only 61 percent of the duplicates with 72 percent precision (see figure 5). After running many experiments and trying different thresholds in the rules, the best recall rate our data set achieved was 75 percent with 79 percent precision (see figure 5). Fine-tuning the rules took approximately one day of trial and error. We can fine-tune the rules when we know the correct answers. Otherwise, if we’re developing this duplicate detection code for an actual database application, we would have to put up with the original performance or verify some of the detected duplicates to evaluate rule performance.

In existing approaches, manipulating the hard-coded rules is difficult, because any change in the similarity function would require altering the program and thresholds. Also, the user must repeat the same coding process for each table schema, and the coded rules are static, so the user can’t perform any training. By using a very simple hand-drawn shape and primitive rules defined by the user, our framework detected 70 percent of duplicates with 90 percent precision, without any hard coding. The resultant records from this approximate match were quite accurate, considering the NP-hardness of optimally solving the duplicate detection problem, the inherent uncertainty of the problem, and the results achieved using rigid and static hard-coded rules.

The framework even achieved better results when we employed the learning capabilities, successfully detecting 85 percent of the duplicates with 90 percent precision.

The training data set consisted of the fuzzy inference engine’s input and output. The inputs were the similarity of the chosen attributes in a tuple, as measured by a user-selected function. The output was the probability of the tuples being duplicates, which were set above the threshold if they were duplicates and below if they weren’t.

The training data set consisted of the comparisons performed for a window size of 10. The system recorded 5,130 comparisons, and the duplicates were identifiable from the unique ID. We randomly broke this data set into three equal parts for training, testing, and validation.

In the ANFIS training process, we trained the FIS using the training data set, monitored the error rate for the validation data set, and chose parameters, which perform best on the validation data set (not the training data set), for the inference system. Then we tested the FIS on the (unseen) testing data set. This way, we avoided model overfitting on the training data set, which degrades overall performance. Threefold cross-validation, as used here, ensured that the system learned to perform well in the general case—that is, for the unseen data.

Figure 6 illustrates the effect of ANFIS training. Figure 2 showed the best hand-
The drawn shape of bell MFs used for the four input variables; figure 6a shows the initial shape of the Gaussian MFs with grid partitioning. After training, these MFs were molded into shape, according to the given input/output data set and the system’s rules to reduce the error rate as much as possible (see figures 6b–e). The change in shape is clear for the first-name attribute (figure 6c). Which MFs changed and how much merely relates to the training data set characteristics and the problem domain.

In figure 7, we compare the precision-recall curve for the trained grid-partitioned FIS that performed best, with the FISs generated using subtractive clustering on top of which we applied hybrid and BP learning. Performance was similar for the three cases, indicating that subtractive clustering is quite effective for training. Again, the user didn’t have to specify any rules or MFs. Here, the subtractive clustering used five MFs per input variable.

Hard coding for duplicate elimination, based on a schema, is tedious and cumbersome. When we employ adaptation, this task is virtually effortless, and even the requirement of specifying common-sense natural language rules and regular membership functions is alleviated. Together, these features make the framework suitable and promising in developing an application-oriented commercial tool for fuzzy duplicate elimination, which is our future plan.

References


For more information on this or any other computing topic, please visit our Digital Library at www.computer.org/publications/dlib.

**The Authors**

**Hamid Haidarian Shahri** is a PhD student in Department of Computer Science, University of Maryland, and a graduate research assistant in the MINDSWA research group. His research interests include information integration, knowledge representation, and adaptive and intelligent systems. He received his MS in computer science from Amirkabir University of Technology (Tehran Polytechnic), Iran. Contact him at A.V. Williams Bldg., Dept. of Computer Science, Univ. of Maryland, College Park, MD 20742; hamid@cs.umd.edu; www.cs.umd.edu/~hamid.

**Saied Haidarian Shahri** is an MS student at the University of Tehran. His research interests include learning in probabilistic graphical models, pattern recognition, reinforcement learning, and decision making under uncertainty. He received his BS in computer science from Ferdowsi University of Mashad, Iran. Contact him at the Artificial Intelligence and Robotics Lab, Dept. of Electrical and Computer Engineering, Univ. of Tehran, Tehran, Iran; s.haidarian@ece.ut.ac.ir.

---

**IEEE Pervasive Computing** delivers the latest peer-reviewed developments in pervasive, mobile, and ubiquitous computing to developers, researchers, and educators who want to keep abreast of rapid technology change. With content that’s accessible and useful today, this publication acts as a catalyst for progress in this emerging field, bringing together the leading experts in such areas as

- Hardware technologies
- Software infrastructure
- Sensing and interaction with the physical world
- Graceful integration of human users
- Systems considerations, including scalability, security, and privacy

**FEATURING IN 2007**

- Healthcare
- Building a Sensor-Rich World
- Urban Computing
- Security & Privacy

**Subscribe Now!**

Visit www.computer.org/pervasive/subscribe.htm