*Artificial Intelligence for Engineering Design, Analysis and Manufacturing* (2003), *17*, 235–252. Printed in the USA.

DOI: 10.1017/S0890060403173064

# A study in applying case-based reasoning to engineering design: Mechanical bearing design

XIAOLI QIN AND WILLIAM C. REGLI

Department of Computer Science, College of Engineering, Drexel University, Philadelphia, Pennsylvania 19104, USA

(RECEIVED April 10, 2002; ACCEPTED January 12, 2003)

**Abstract**

Case-based reasoning (CBR) is a promising methodology for solving many complex engineering design problems. CBR employs past problem-solving experiences when solving new problems. This paper presents a case study of how to apply CBR to a specific engineering problem: mechanical bearing design. A system is developed that retrieves previous design cases from a case repository and uses adaptation techniques to modify them to satisfy the current problem requirements. The approach combines both parametric and constraint satisfaction adaptations. Parametric adaptation considers not only parameter substitution but also the interrelationships between the problem definition and its solution. Constraint satisfaction provides a method to globally check the design requirements to assess case adaptability. Currently, our system has been implemented and tested in the domain of rolling bearings. This work serves as a template for application of CBR techniques to realistic engineering problems.

**Keywords:** Artificial Intelligence; Case-Based Reasoning; Computer-Aided Design; Design; Variant Design

## 1. INTRODUCTION

Case-based reasoning (CBR) techniques are a promising methodology for solving many problems in engineering design. CBR is a subfield of artificial intelligence (AI), based on the idea that past problem-solving experiences can be reused and learned from when solving new problems. This paper shows how to use CBR techniques to build a CBR system to solve a domain-specific engineering design problem: the design of mechanical bearings. This paper presents a three-phase approach to building a practical CBR system for this domain:

1. *Knowledge representation for bearing design problems:* Determine the key parameters in the design problem and use them to build a knowledge base.
2. *Case-based reasoning engine:* Design and implement a case-based reasoner that can retrieve and adapt past design knowledge.
3. *Implementation and examples*: Develop a prototype based on this approach and show how the CBR system can be used during the design phase of product development.

In presenting technical solutions to each of these problems and the system prototype, this work serves as an example for others to use in applying case-based techniques to more complex engineering design problems.

## 2. BACKGROUND

Case-based and knowledge-based systems have been an active research area for the past 15 years (Hammond, 1989; Riesbeck & Schank, 1989; Bardasz & Zeid, 1991, 1992; Hinrichs & Kolodner, 1991; Slade, 1991; Kolodner, 1993; Pu, 1993; Aamodt & Plazas, 1994; Maher et al., 1995; Leake, 1996). This work represents a foundation of structures, algorithms, and techniques for reasoning about and adapting archived knowledge. An area of considerable interest has been engineering, design, and manufacturing, which provides a vast array of challenging, real-world problems that test theoretical developments and create new technologies.

Reprint requests to: William C. Regli, Department of Computer Science, College of Engineering, Drexel University, 3141 Chestnut Street, Philadelphia, PA 19104. E-mail: regli@drexel.edu

This section will first review a subset of CBR literature driven by engineering, design, and manufacturing most relevant to this paper and then provide a primer on bearing design.

### 2.1. Previous work on CBR in engineering

CADET (Navin-Chandra et al., 1991; Navin-Chandra, 1992*a*; Sycara et al., 1992; Sycara & Navin-Chandra, 1992; Miyashita & Sycara, 1993) and its descendent projects focus on conceptual design solving problems using relationships that capture function, structure, and behavior. CADET builds solutions to new design problems from pieces taken from previous design cases. CADET's representations are behavioral and functional, with input to the system consisting of symbolic descriptions of the desired device along with some physical constraints. The design knowledge base of CADET is a store of function, behavior, and the device's structural relationships. Indexing and retrieval are performed using linguistic descriptions of these properties as well as queries on the symbolic information and parameters. The retrieval and indexing methods are based on variations of graph matching and support retrieval at different degrees of abstraction.

Goel et al.'s KRITIK and its descendent systems (Chandrasekaran et al., 1993; Bhatta and Goel, 1994; Goel, Bhatta, et al., 1996; Goel, Gomez de Silva Garza, et al., 1996; Goel & Stroulia, 1996; Goel, 1997) operate on design problems using a case base of designs represented by symbolic component descriptions and their relationships and behaviors. A central contribution of KRITIK was the formalization of a structure–behavior–function model for designs, in which design cases can be indexed according to the functions they deliver. The functional representation is hierarchical, consisting of a component-substance model to capture the structure and performance of a given device.

KRITIK's design domain is not linked to specific CAD geometry and topology specifications (such as are captured in current engineering databases and product data management systems) and is limited to devices whose functions can be characterized as a flow of substances between components. The more recent work has extended many of the earlier KRITIK concepts; however, their powerful reasoning techniques are still primarily symbolic and have not been coupled with detailed engineering data.

Other systems include those for assembly (REV-ENG; Kim, 1997), architecture (Archie, Arichie II, Domeshek & Kolodner, 1997; CADRE, Hua & Faltings, 1993; Fabel, Voss, 1997), and civil engineering (Cadsyn, Casecad, Gencad; Maher & Zhang, 1993; Maher et al., 1995; Gomez de Silva Garza & Maher, 1996; Maher & Gomez de Silva Garza, 1996), among other engineering disciplines (Hennessy & Hinkle, 1992; Shi et al., 1997). Smithers (1989) describes the need to unite geometry with richer AI representations; Silverman and Mezher (1992) overview the work on design critics.

Work by Bose, Gini, and Riley (1997) applies CBR to the design of planar linkage assemblies. In this work, planar linkages are stored as parametrized 2-dimensional geometric information, along with functional information about the elements. The case storage structure is also multilevel, allowing for problem specification and retrieval at varying levels of abstraction. Case retrieval is executed using an algorithm that is a traversal of a variation of a KD tree, which hierarchically stores the cases.

Other case-based systems for problems in design and manufacturing include the case-based assembly planner of Pu and Reschberger (1991a, 1991b), Falting's Design-CADRE System (Hua & Faltings, 1993), and the Tsatsoulis application of the TOLTEC Planner (Tsatsoulis & Kashyap, 1993) to manufacturing problems. Lambright and Ume (1996) applied CBR/KBR to the design of flat panel displays. In addition, issues of *IEEE Expert and Intelligent Systems* have emphasized past accomplishments and current challenges in the extension of AI and CBR to complex engineering problems (Brown & Birmingham, 1997; Goldman & Boddy, 1997; Lee, 1997; Maher & Gomez de Silva Garza, 1997; Sauer & Bruns, 1997; Umeda & Tomiyama, 1997; Wielinga & Schreiber, 1997). Some of the observations in this series of articles include the following:

1. although there has been much research in knowledge-based engineering systems, the integration of this research into existing computer-aided design (CAD) tools has yet to really begin (Brown & Birmingham, 1997);
2. existing research systems still have great difficulty in scaling to complex design cases such as those posed by large CAD systems (Maher & Gomez de Silva Garza, 1997);
3. current CAD systems and their underlying representations are predominantly geometric and integrating knowledge about form and function is a major open research challenge (Umeda & Tomiyama, 1997); and
4. solving even the simplest design problems, such as the creation of a part configuration layout, requires advanced AI technology and novel extensions to the state of the art.

In a survey of work on variant and case-based design, Fowler (1996) makes several similar observations: better abstract models are needed for mechanical artifacts so that function information can be stored in the CAD knowledge base (in much the same way that functional indices are computed in KRITIK). Complex issues need to be considered to develop systems for automatically retrieving and applying existing designs to solve new design problems. Augmenting CAD systems with CBR/case-based design techniques can lead to great benefits to designers.

### 2.2. The CBR method

The CBR Cycle (Aamodt & Plaza, 1994) is a methodology to build a CBR system for a given domain. A CBR system

https://doi.org/10.1017/S0890060403173064 Published online by Cambridge University Press

can be viewed as a combination of *case-base* and *knowledge reasoning* process modules. These modules form a *CBR (or reasoner*), and they form the functions used to manipulate the knowledge in the case base. They act to *process* user inputs, *recall* similar cases, *retrieve* the most similar case, and *evaluate and adapt* the retrieved case and update the case memory.

Normally, the following problems must be addressed in the development of a CBR system: *knowledge acquisition*, *knowledge representation*, *case retrieval*, *case adaptation*, and *learning mechanisms*. We review the basic aspects of each step below:

1. *Knowledge acquisition:* How does one acquire useful knowledge from the application problem domain? This activity often consists of manual indexing of past design knowledge; sometimes automated or semiautomated indexing of design knowledge is possible.
2. *Knowledge representation:* How does one use a formal language, such as first order logic, to represent domain knowledge? The knowledge representation methodologies used in CBR systems are primarily concerned with how to structure knowledge stored in the case base to facilitate effective searching, matching, retrieving, adapting, and learning. One influential knowledge representation model is the *dynamic memory model* (Riesbeck & Schank, 1989), based on Memory Organization Packet (MOP) theory.
3. *Case retrieval:* Once we have determined how to represent knowledge and have populated a knowledge base with cases, how do we efficiently retrieve the case most similar to the current problem? There are two subprocesses involved in case retrieval: how to retrieve a set of similar cases from the case base and how to find the most similar case in this set. The first subprocess is accomplished by designing an appropriate index scheme for the domain problem. The second task is often done using techniques such as the *Nearest Neighbor Matching Algorithm* (NNM; Kolodner, 1993).
4. *Case adaptation strategies:* After a CBR system retrieves the most similar case from the case base, it normally needs to perform some modification on this retrieved case to adapt it to the new problem. There are several adaptation strategies that can be used in a CBR system. They include Simple Substitution, Parameter Adjustment, and Constraint Satisfaction (Kolodner, 1993).
5. *Learning mechanisms:* Learning is the last step in the CBR system. In a CBR system, after a new problem is solved, the case base is changed by adding the new case into it. In this way, the system can retain more knowledge along with problem-solving augmentation and achieve learning.

A CBR engine forms the control system that allows designers to use archived cases to solve new bearing design problems. Once domain knowledge has been used to build the case base, organize memory, build indices, and so forth, the reasoning engine can execute searches based on the index scheme. The engine also performs the other reasoning processes, including case retrieval, adaptation, and system learning.

### 2.3. Bearing design

Bearings are standard mechanical elements that play a very important role in product design and are used extensively in a wide array of mechanical artifacts. They usually support rotating shafts and make relative rotation possible among shafts and other parts (i.e., gears). Whenever a newly designed machine requires rotational function, it also requires bearings. A bearing designed for a certain machine must satisfy the requirements of the overall assembly structure and working environment. The basic way to solve this problem is to perform intensive calculations based on the working conditions and develop a bearing configuration which can satisfy these working requirements. Some computer programs have been developed to help deal with these intensive calculations (HEXAGON, 1999). Although these approaches release human engineers from manual mathematical calculation, they cannot perform higher level design actions.

Because of the complexity of the bearing design problem, the knowledge space in this domain is incomplete and dynamic. Therefore, knowledge acquisition has to be achieved by specifying only the important features relevant to solving the specific problem. Knowledge not directly related to solving the problem is discarded. In this work, we predefine a set of important features for the bearing design problem, and knowledge acquisition is done manually by a knowledge engineer.

## 3. CASE REPRESENTATION FOR THE BEARING DESIGN PROBLEM

### 3.1. Problem formulation

There are two basic types of bearings commonly used in industry: *rolling* bearings and *sliding* bearings. This paper consider only the former. Rolling bearings are further divided into subcategories according to the geometric shape of their rolling components. Some have rolling components that are cylinders and some are spheres, called ball bearings.

The basic components of a bearing are an *inner ring*, an *outer ring*, the *rolling components* and a *supporting cage*, which keeps the rolling components distributed uniformly. Figure 1 depicts the bearings, but the cages are not shown.

Normally, the bearings are installed on a rotating shaft. The inner ring of a bearing is fastened on a shaft, and the outer ring is installed in a housing. The fundamental pur-

https://doi.org/10.1017/S0890060403173064 Published online by Cambridge University Press

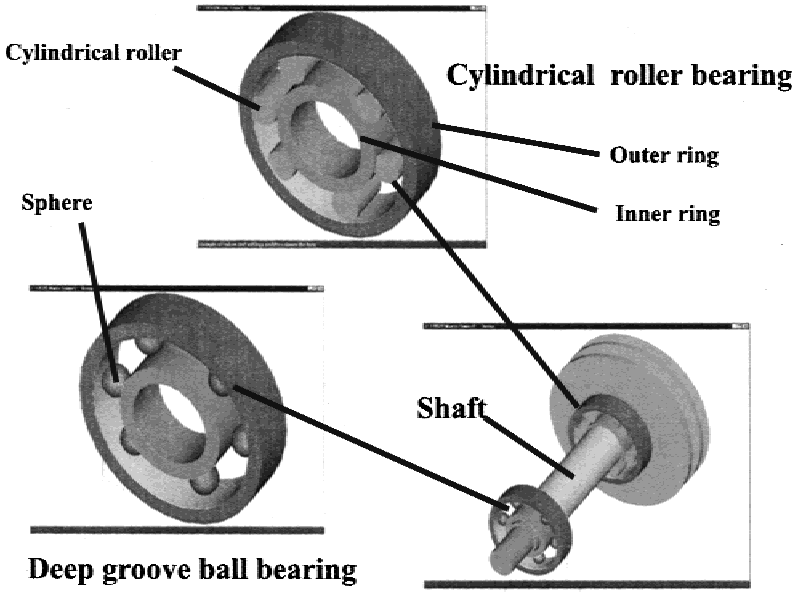


**Fig. 1.** The bearings and where they are installed.

pose of a bearing is to transmit the load between a stationary part of a machine (commonly a housing) and the rotating part of the machine (commonly a shaft) with the minimum resistance.

*3.1.1. The bearing design problem*

Bearing design is interpreted from the perspective of an application engineer, that is, he or she designs bearings for machines or any applications where bearings are needed. When performing design, he or she must consider:

1. the working environment for the design problem, including ambient conditions, load conditions, and so forth;
2. based on this information and information given by a manufacturer's catalog (which gives different bearings' maximum load capacity, speed limit, etc.), how to design and calculate the size of a bearing that is suitable for the specified shaft diameter, maximum dynamic life under the working load, maximum speed, and so forth.

The goal is to make correct decisions in regard to bearing type, size, and material, through analysis of the working environment and extended calculation based on the given working conditions. Appropriate bearing design is vital to the trouble-free operation of the machinery.

*3.1.2. Important design factors*

The inputs include the working conditions, load to be applied on the bearings, shaft speed, lubrication (i.e., oil or grease), assembly space, ambient temperature, corrosive atmospheres and vibrations, and so forth. There are also other important factors that must be considered, such as misalignment, quiet running, and so forth. The primary design factors that are considered in this research are the following:

1. *Load*: The magnitude of the load is the factor that usually determines the size of the bearing to be used. The direction of loads applied on the bearings is also very important.
2. *Speed*: The speed at which rolling bearings can operate is limited by the permissible operating temperature.
3. *Available space*: When radial space is limited, it is necessary to choose bearings with a small cross section, particularly those with low cross section height (i.e., needle roller bearings; see Fig. 2).

These design parameters, although not exhaustive, cover the major aspects of most bearing design problems.

*3.1.3. Design calculations*

The primary calculations are to predict the probability of bearing failure: "How long can a bearing be used in a certain working environment?" The first step in predicting bearing life expectancy is to calculate the equivalent load applied on the bearing. Figure 3 illustrates this calculation. Any load applied on a bearing can be decomposed into a *radial load* and an *axial load*. The radial load and axial load are the component forces of an equivalent compound force whose directions are *radial* and *axial*. Normally, a radial load and an axial load can be obtained from a special testing instrument and the equivalent compound load can be calculated from these measurements.

The variants of the formula given in Figure 3 can be expressed with two formulae. The first is the theoretical

https://doi.org/10.1017/S0890060403173064 Published online by Cambridge University Press

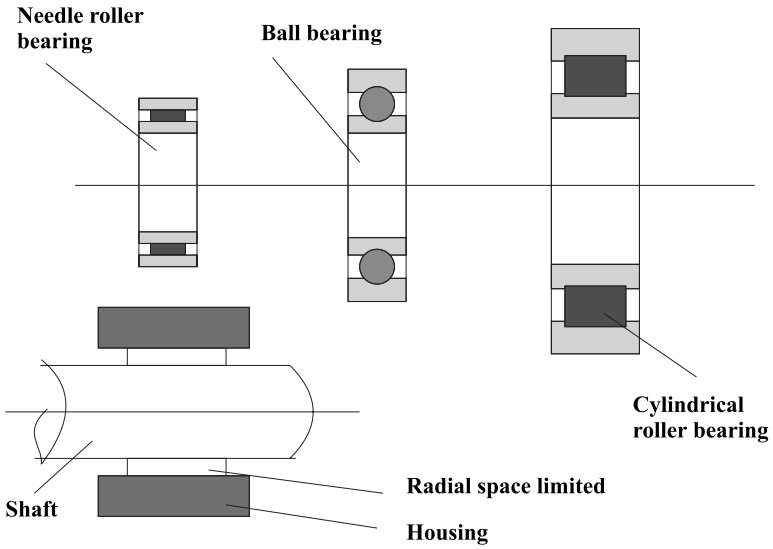


**Fig. 2.** The available space in the design configuration.

formula for computing equivalent load applied on a bearing (Wilcock & Booser, 1957):

$$W = \frac{(1 - \sin\alpha)F_r + (\cos\beta)F_t}{(2.5 - \sin\alpha)}$$

or

$$W = F_r, \quad \text{if } F_r > W.$$

The second is the heuristic formula for computing an equivalent load applied on a bearing (Wilcock & Booser, 1957):

$$W = 0.37F_r + 2.0F_t$$

or

$$W = F_r, \quad \text{if } F_r > W,$$

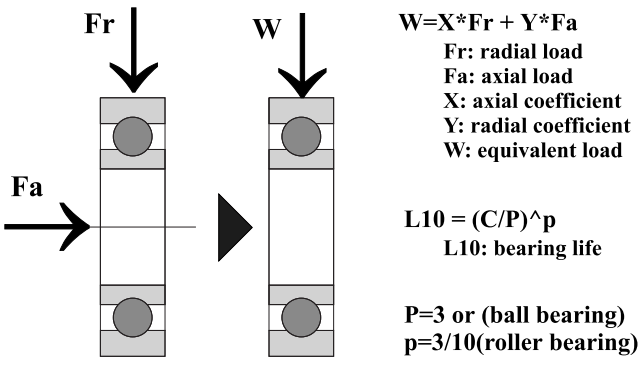


**Fig. 3.** The calculations for the equivalent load applied on the bearing.

where $W$ is the bearing load (N), $F_r$ is the radial load applied to bearing (N), $F_t$ is the axial load applied to bearing (N), $\alpha$ is the operating contact angle (rad), and $\beta$ is the initial contact angle (rad).

The formula for computing bearing life can be expressed as (Wilcock & Booser, 1957)

$$L_n = \left(\frac{C}{W}\right)^3 \quad \text{(millions of revolutions)}$$

or

$$L_{10} = \frac{C^3 \times 10^6}{60 \times N \times W^3} \quad \text{(h)}$$

where $L_n$ is the bearing life in millions of revolutions, $L_{10}$ is the bearing life (h), $C$ is the load rating constant (N), $N$ is the speed of the shaft (rpm), and $W$ is the equivalent load imposed on the bearing (N).

Although other calculations do exist that are involved in bearing design problems, these calculations are omitted in order to simplify discussion of how we will represent domain knowledge for use in a case-based design system.

### 3.2. Case representation schema

The knowledge pertaining to bearing design problems can be represented in any kind of formal knowledge representation language. We have chosen to use CBR Language (CASL; Center for Intelligent Systems, 1999), a language specially designed for CBR. CASL can be used to define the contents of the case base (in a *case file*), and the reasoner uses this case file to create a case base to be accessed and adapted in order to solve design problems.

https://doi.org/10.1017/S0890060403173064 Published online by Cambridge University Press

*3.2.1. General syntax and semantics of CASL*

Like any other representation language, CASL has strict syntax, semantics, keywords, and operators. The syntax of CASL specifies the grammar rules of organizing knowledge, and the semantics of CASL give the concise interpretation of a sentence written in CASL with correct grammar. CASL defines some basic types in the language: identifiers, strings, numbers and operators, and so forth.

CASL normally divides a case base into several modules, each of which has its own syntax features and semantic explanations. These modules are the following:

**Introduction**
**Case Definition**
**Index Definition**
[**Modification Definition**]
[**Preprocessing Rule Definition**]
[**Repair Rule Definition**]
**Case Instance**

CASL semantics define the meaning of a sentence by specifying the interpretation of the keywords and basic types and specifying the meanings of operators. In the syntax blocks of CASL, all keywords and literals are given in bold type. Brief explanations of the primary modules are given below:

**Introduction** defines introductory text that documents to the user understand the contents of the case base or anything else of note.

**Case Definition** defines the problem features contained in a case.

**Index Definition** defines which fields are to be used as indices.

**Modification Definition** The purpose of this block is to define rules used to modify a retrieved case from the case base to make it fit the current problem specifications. The *global repair rule definition* defined in this module allows adaptation rules to be applied on any modified case. The rules defined here are derived from domain knowledge, formulae and constraints.

**Case Instance Definition** defines the structure of a case instance. A case must contain two parts: the problem part and the solution part. The *local repair rule definition* defined in this module allows adaptation rules to be associated with a case. These rules are invoked after the *global* adaptations have run their course.

*3.2.2. Examples for bearing design domain representation*

*The feature definitions for user input.* When a bearing for a machine is designed, working conditions are specified and given to the *CBR reasoner*. The **case definition is** block in CASL is used to structure the input specifications. It structures the knowledge about case instances and input problems by defining the primary features of a problem. Following the **case definition is** keyword is the definition of problem features, which can have different *weights* according to their importance in the problem definition. The **weight is** keyword is used to specify the weight of a feature.

In the bearing design problem, the most important features are axial load and radial load. These features' weight values are set to be 5 (the reference weight). Load direction, shaft housing diameter, allowed radial limited space, and so forth are not that important, comparatively speaking. Therefore, their weight values are set to be 0 (the reference weight). A sample case definition using CASL can be given as:

**case definition is**
**field** shaft_housing_diameter **type is**
(d_12_24,d_12_28) weight is 5;
**field** load_direction **type is**
(radial,axial,combined) weight is 0;
**field** radial_limited_space_requirement **type is**
(Yes,No) weight is 0;
**field** radial_load **type is number**
weight is 5;
**end**;

Explanations of the case definition include the following:

1. The feature shaft_housing_diameter defines shaft and housing diameters. The purpose of this field is to define a series of possible shaft and housing diameters that may appear in the problems.
2. The load_direction field defines the load direction that is applied to the bearing. The purpose of defining this field is that some bearings can only carry axial direction loads, some can only carry radial direction loads, and some can carry loads in both directions.
3. The feature radial_limited_space_requirement defines the available radial space in the machine in which the designed bearing can be assembled. In some cases, the design has certain assembly space requirements for special purposes. That is, the available space for bearing design may be restricted in a certain dimension. These space requirements can help a designer predetermine his choice of bearing.
4. The field radial_load defines the magnitude of the load that is applied to the bearing in the radial direction. This is the most important factor in deciding the bearing design for a machine, and in this work the reference weight is specified as to be 5.

*The index feature definition.* This part defines the fields that are used as indices when searching for a matching case.

https://doi.org/10.1017/S0890060403173064 Published online by Cambridge University Press

The index scheme defines the methods by which the reasoner should access the case memory. Indices are intended to streamline the matching process. The index features are parts of the new problem specification. For example, we use the features shaft_housing_diameter and load_direction as main indices to search the knowledge base. The sample representation is given below:

**index definition is**

**index on** shaft_housing_diameter;

**index on** load_direction;

*The definitions of adaptation rules.* When the old bearing design whose "description of problem definition" part is the most "similar" to the current problem definition is retrieved from the case base, its solution part must be modified to fit the current problem definition. The reasoner performs adaptations to an old solution according to certain rules defined by domain experts. The **repair rule definition is** block of CASL can be used to define those rules. In the bearing design problem, the following strategies are defined:

1. Perform simple parameter substitution: substitute parameters of old problem definition into new user input.
2. Perform old solution adjustment to make it fit substituted user input (the current problem) according to domain formulae.
3. Check global constraints defined in the case base to guarantee that no conflicts result.

In the sample given in Algorithm 1, *change*_value_1 is an adaptation rule. It tests a certain condition (represented by a formula) first; when the condition is satisfied, the action is fired. The action here is the recalculation of bearing life (represented by a formula) according to the current user input.

**Algorithm 1.** Representation of adaptation rules

(1) **repair rule definition is**
(2) **repair rule** |change_value_1| **is**
(3) **when**
(4) $(0.37{*}radial\_load + 2{*}axial\_load) \geq radial\_load$
(5) **then**
(6) **evaluate** |bearing_life| **to**
(7) $$\frac{10^6{*}support\_value\_dynamic\_C^3}{(0.37{*}radial\_load + 2{*}axial\_load)^3 60{*}average\_speed}$$
(8) **repair;**
(9) **end;**
(10) **end;**

*The definition of a stored case.* An experience (case) includes a problem statement part and a solution part. The **case instance is** block of CASL provides a kind of structure and function. This block defines the same structure of problem statement as the **case definition is** block defines.

In a bearing design for an application, some relationships between the problem statement and the solution are unique only for this design (case). For this reason, some features of a case are defined as "local," meaning the attributes for these features are valid only for this design. For example, the features *average*_speed and *expected*_bearing_life are defined as local because every bearing designer specifies his or her own shaft speed and requires his or her own expected bearing life. In addition, every bearing has its own permissible speed limitation defined by the manufacturer and its own life expectancy according to the working environment.

If it is necessary to define some rules to adapt local features, then these rules must be specified as local. That is, the local rules are defined in a **case instance is** block. In the given sample below, the rule *rule*_1 is local because this rule checks the constraints for local features (i.e., *expected_bearing_life*). A sample representation of a case is Algorithm 2.

**Algorithm 2.** Case instance representation

(1) **case instance** |needle_roller_hk1512| **is**
(2) |shaft_housing_diameter = d_15_21;|
(3) |load_direction = combined;|
(4) |radial_limited_space_requirement = Yes;|
(5) |axial_limited_space_requirement = No;|
(6) |radial_load = 550;|
(7) |axial_load = 100;|
(8) **local field definition is**
(9) **field** |average_speed| **type is number;**
(10) **field** |expected_bearing_life| **type is number;**
(11) **solution is**
(12) |bearing_type = needle_roller_hk1512;|
(13) |calculation_speed = 10000;|
(14) |drill_hole_diameter = 15;|
(15) |outer_diameter = 21;|
(16) |width = 12;|
(17) |support_value_dynamic_C = 7650;|
(18) |permissible_speed = 11000;|
(19) |bearing_life = 11350;|
(20) **local repair rule definition is**
(21) **repair rule** |rule_1| **is**
(22) **when**
(23) *expected_bearing_life* ≥ *bearing_life*
(24) **then**
(25) **print** |'Abandon your selection!';|
(26) **print** |'Bearing life can not meet your requirement!';|
(27) **reselect;**
(28) **repair;**
(29) **end;**
(30) **end;**

## 4. PROTOTYPE BEARING DESIGN SYSTEM

### 4.1. System overview

The CBR *engine* allows the designer to navigate and manipulate the case base through a graphical user interface.

https://doi.org/10.1017/S0890060403173064 Published online by Cambridge University Press

In this work, our CBR engine is implemented with C and the Microsoft Visual C++ programming environment. Our system uses CASL (Center for Intelligent Systems, 1999) to represent our design knowledge and the case base and MOP theory (Riesbeck & Schank, 1989) to develop a structure of the case base. The kernel of the CBR engine is based on the CASL environment from the University of Wales (Center for Intelligent Systems, 1999).

Once the user enters the problem specifications and provides a case base, the reasoner analyzes the problem and returns an answer to the user automatically. Our reasoning engine consists of four process modules, each performing certain functions to implement the complete CBR cycle. The first module, *Retrieved Case*, takes the current problem specifications as input and outputs a retrieved case. The second module, *Solved Case*, decides whether a retrieved case needs to be adapted. This module either returns the user a solution without further modification or passes a solution to the next module, which will perform adaptation on the case. The third module, *Repaired Case*, performs the adaptation and returns an adapted case to the next module. The fourth module, *Learned Case*, decides whether this new resolved case needs to be stored in the case base. The following sections will detail how these modules are implemented.

### 4.2. Reasoning engine

The flowchart in Figure 4 shows the main algorithm behind the implementation of a reasoning engine. The two hollow arrows in the figure illustrate that the reasoning engine must interact with the case base.

The flowchart shows that the requirements of a module can be broken into pieces or procedures called by the main function. It also shows that a CBR engine forms a reasoning loop. This reasoning loop begins with the procedure *User Specification* and ends with the procedure *Add Case*.

### 4.3. Building the case index

The performance of a CBR system is determined by the CBR reasoning engine whose efficiency in turn is determined by the design of the *index scheme* and *case-base memory organization*. The index scheme design includes how to specify index features and how to build them in computer memory. The index features are set by domain experts and are represented by the block **index definition is** of CASL. The procedure *Build Indices* takes the representations of index features as input and uses these to build the index scheme. A linked list data structure holds the index feature input. The procedure *Build Indices* places all

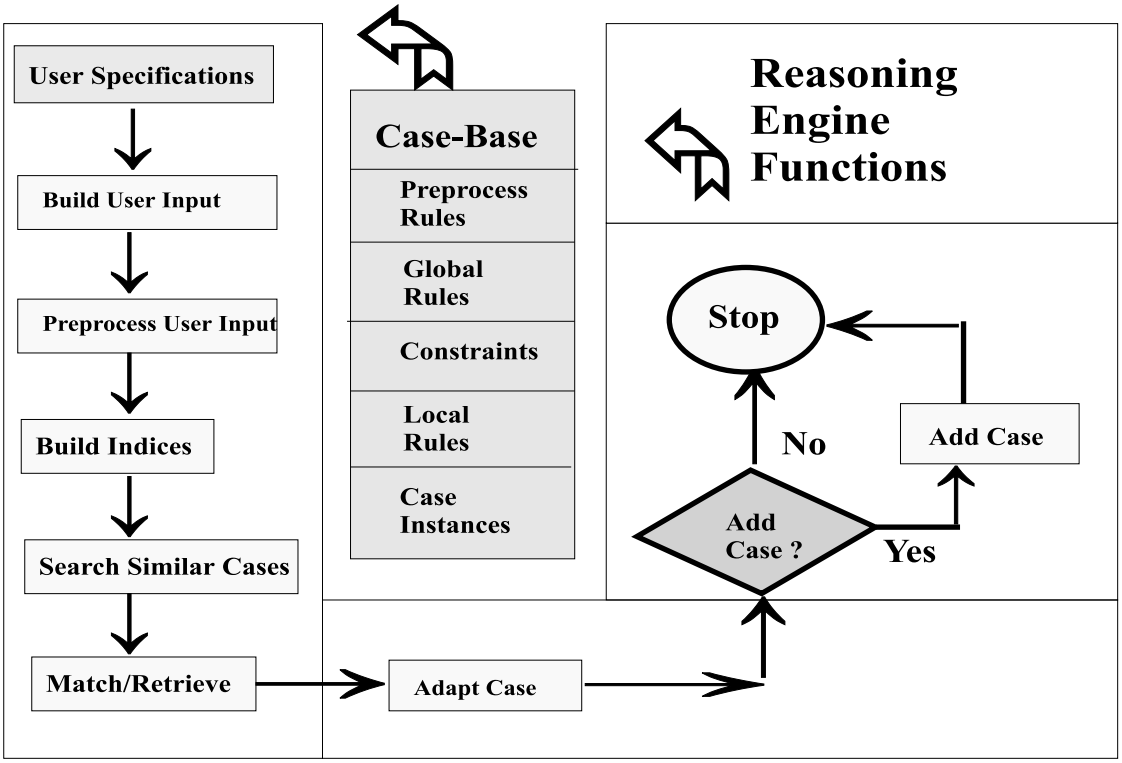


**Fig. 4.** The primary functions of the CBR engine.

https://doi.org/10.1017/S0890060403173064 Published online by Cambridge University Press

the index features into the list and, at the same time, builds the case-base memory organization (shown in Fig. 5).

In this CBR system, two features have been specified as index features: *shaft diameter* and *load direction*. Each index feature is a node in the list and the feature's attributes are associated with the nodes. Figure 5 shows this data structure for the index features and case-base memory. The procedure *Build Index* first links the index features *shaft diameter* and *load direction*. It then checks every attribute of the index features. For each attribute, *Build Index* searches for all the cases with the same attribute value in the case-base file and links all of these together.

### 4.4. Case matching, ranking, and retrieving

The purpose of building an index scheme is to speed up *searching*. Here, *searching* means to find a set of cases from the case base that are similar to the current input case. However, the goal here is to find the case which has the maximum similarity to the input case. Thus, a mechanism to rank the similarity of cases is needed. In section presents how to achieve these two goals: finding a similar case set and finding the most similar case in this set.

A mathematical model is first presented to show how to find a set of similar cases in the case base. What are similar cases? *Given an input case with certain index features and their attributes, similar cases are those cases whose index features and attributes are exactly the same as the corresponding input case.* Figure 6 shows these ideas.

The upper part of Figure 6 presents the mathematical model for finding similar cases. The left and right circles represent attributes $F(A)$ and $F(B)$ of index features $A$ and $B$ of an input case, respectively; and $C(n)$ represents a case $n$. If the left circle includes $C(b), C(d), C(h)$, and $C(a)$, which are the cases with attribute $F(A)$ of feature $A$, and the right circle includes $C(i), C(j), C(a)$, and $C(h)$, which are the cases with attribute $F(B)$ of feature $B$, then their intersection contains cases $C(a)$ and $C(h)$, which have both attributes $F(A)$ and $F(B)$:

$$\{C(a), C(h)\} \subset F(A) \cap F(B).$$

The lower part of Figure 6 gives a corresponding example that illustrates how this process occurs in the case base. After all similar cases are found, a mechanism to find the most similar case in this set is needed. We used the NNM algorithm (Kolodner, 1993). Figure 7 shows how this algorithm works in our CBR system for bearing design. To simplify discussion, we assume that all the component loads (axial load and radial load) applied on the bearing are in the same direction.

The basic idea of the NNM algorithm is to compare the attribute value of each feature of each case in the set of similar cases to every corresponding feature's attribute of the input case, calculate the comparison values, and then sum them for each case to get a total comparison value.

In the upper part of Figure 7 the circles represent cases, the dots represent attribute values of features, index $i$ represents the input case, index $j$ represents cases in the set of similar cases, and index $k$ represents the features in a case. Case $A$ and case $B$ in the figure are the cases from the set of similar cases. The function $d(k)(ij)$ represents the attribute's comparison value of one of the features (feature $k$) between the input case and case $A$, which is equal to the following formula (Kolodner, 1993):

$$W(ij) \times \mathrm{sim}(F(k)(R)i, F(k)(I)j),$$

where $k$ is a feature of a case; $W(ij)$ is the weight of a feature, defined in the case-base file; and $\mathrm{sim}(F(k)(R)i, F(k)(I)j)$ is the degree of similarity between one of the features in the input case and the corresponding feature in a case from the similar case set.

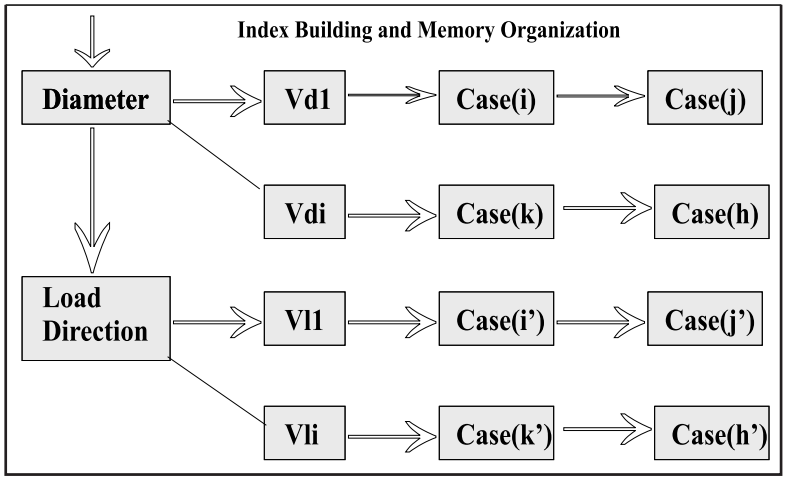


**Fig. 5.** The index building and case-based memory organization.

https://doi.org/10.1017/S0890060403173064 Published online by Cambridge University Press

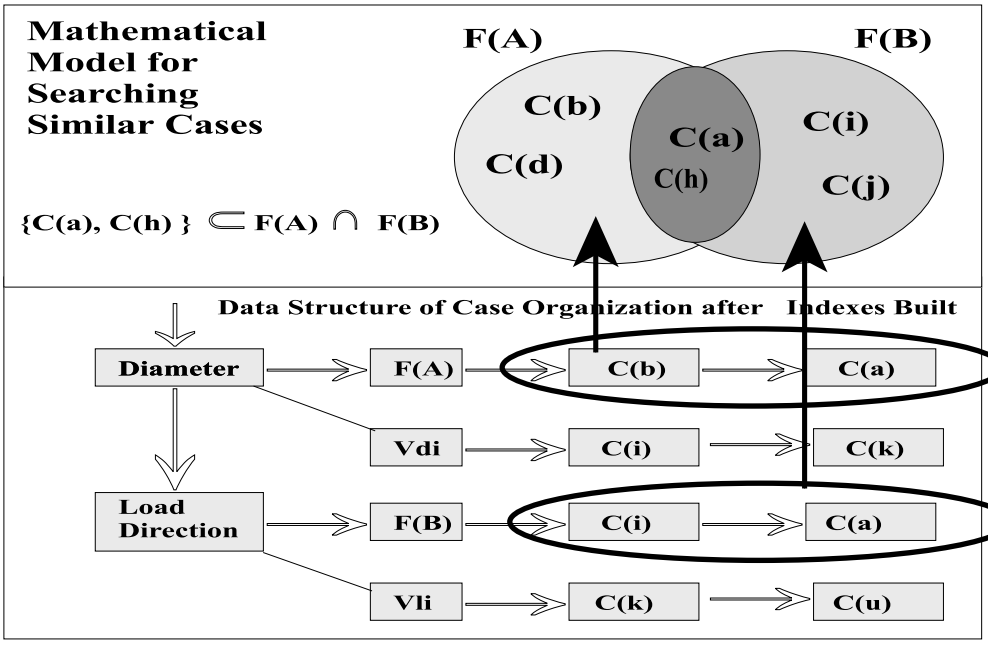


**Fig. 6.** The mathematical model and an example of searching similar cases.

The total attributes' comparison value for a case is $D(k)(IA)$, which is equal to the numeric function

$$\sum_{k=1}^{n} W(ij) \times \mathrm{sim}(F(k)(R)i, F(k)(I)j).$$

The NNM algorithm ranks the case that has the highest value of $D(k)(ij)$ as the most similar to the query case.

The key computation in the NNM algorithm determines the distance between the feature attributes for the input case and the cases in the case-base. A *relevance matrix*, shown in the lower part of Figure 7, is used to explain how to calculate every feature's attribute comparison value. In the matrix, $F(k)(R)i$ represents "the feature $k$ of a case from the similar case set that has possible attribute $i$, where the range of $i$ can be from 1 to some finite number." Except in reference to the input case, $F(k)(I)j$ has a similar meaning. Thus, the first row of the matrix represents all the possible attributes of feature $k$ of a similar case, and the first column represents all the possible attributes of feature $k$ of the input case. The intersection of row and column is the comparison value of the feature $k$. Expression $W(ij)$ is the weight of a feature in a similar case. The degree of similarity, $\mathrm{sim}(F(k)(R)i, F(k)(I)j)$, has three possible values. First, if two features match exactly, the degree of similarity equals 1. Second, if two *abstract symbols* are similar, its value is $\frac{3}{4}$.

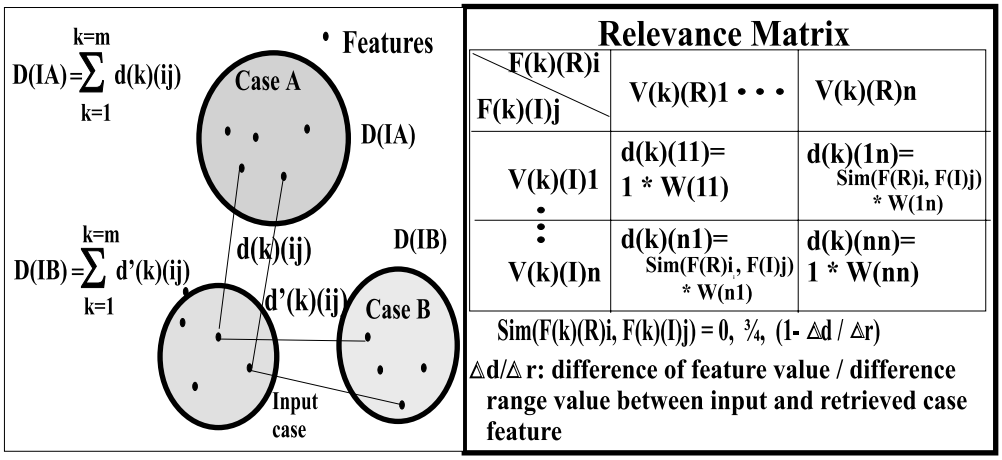


**Fig. 7.** The nearest neighbor matching algorithm.

https://doi.org/10.1017/S0890060403173064 Published online by Cambridge University Press

Third, if two *numbers* are similar (i.e., both fall within the range defined in the *modification* block), then a value is calculated that reflects how close they are in proportion to the range. Then, the $\mathrm{sim}(F(k)(R)i, F(k)(I)j)$ can be calculated by

$$1 - \frac{\Delta d}{\Delta r},$$

where $\Delta d$ is the difference of the feature values between the input case and the retrieved case and $\Delta r$ is the difference range value. For example, if the attribute value of feature *radial load* for the input case is 100 N and the corresponding value for a similar case is 120 N, then $\Delta d = 120 - 100 = 20$. If the definition for the range of similarity is from 90 to 140, then $\Delta r = 140 - 90 = 50$, where the similarity ratio is computed as $1 - (20/50) = 0.6$.

Algorithm 3 defines the functions that implement the finding of similar cases and the most similar case, as mentioned above. The procedure *Index*_List_Searching() performs searching on the linked list of index features. Procedure *Case*_List_Searching() searches out cases whose attribute value for certain features is the same as the input case. Procedure *Computing*_Weight_Cases() performs calculation of the weight of a retrieved case. Procedure *Evaluating*_Similar_Cases() performs ranking for a case with a weight. Procedure *Retrieving*_Heaviest_Case() retrieves the case with the highest rank.

**Algorithm 3.** Case matching, ranking and retrieving
**Input.** User's input problem specification.
**Output.** The retrieved case with highest weight.
MATCHING_RANKING_RETRIEVING(*UserInput*)

```
(1)   begin
(2)   while true
(3)     do
(4)     |Index_List_Searching();|
(5)     |Case_List_Searching();|
(6)     |Computing_Weight_Cases();|
(7)     if Case_Matching_Exact = True;
(8)       return Retrieving_Case();
(9)     else
(10)      |Evaluating_Similar_Cases();|
(11)      |Retrieving_Heaviest_Case();|
(12)      end
```

### 4.5. Adaptation of cases

Very rarely, a retrieved case is exactly the same as the newly defined problem. Most of the time, however, the retrieved case is only a similar situation and so problem definitions and corresponding solutions need to be modified so that the modified case fully fits the current situation and its solution fully satisfies the current problem requirements. This procedure as a whole is called the case adaptation (or repair) process. A series of rules are defined for adapting cases. These rules are provided by domain experts or domain axioms and are applied to each case whenever it is necessary.

Adaptation rules are divided into *global rules* and *local rules* The reasoner uses *global rules* to examine the problem fields and solution fields of the retrieved case. These rules are also used to adapt the parameters of the retrieved case and check constraints satisfaction conditions that are specified by the knowledge base. If there are any constraint conflicts, the repair rules provide a new problem-solving proposal. Otherwise, they adapt the solution of the retrieved case to the new problem. The sample adaptation rules for global repair are described in Algorithm 1.

After the reasoner finishes checking the global rules, it immediately checks the local rules defined in the retrieved case. It applies these local rules to the retrieved case to perform local adaptation (i.e., unique to this case). Some sample local adaptation rules are given in Algorithm 2.

Figure 8 shows that a linked list data structure is used to store these adaptation rules. In the figure, every node has two fields: one stores the condition of a rule, and the other stores the action. The procedure given in Algorithm 4 scans the rule list repeatedly as it performs adaptation on a retrieved case; if the condition part is true, it executes the corresponding actions on the case.

**Algorithm 4.** Algorithm for case adaptation
**Input.** Retrieved case.
**Output.** The modified case.
CASE_ADAPTATION(*RetrievedCase*)

```
(1)   begin
(2)   while true
(3)     do
(4)     if Global_Rules = True;
(5)       |Finding_Global_Rule_Headpointer();|
(6)       |Searching_Global_Rules();|
(7)       |Apply_Modifying_Retrieved_Case();|
(8)       |Parametric_Adaptation();|
(9)       |Constraints_Adaptation();|
(10)      |Evaluating_Solutions();|
(11)    else
(12)      |Finding_Local_Rule_Headpointer();|
(13)      |Searching_Local_Rules();|
(14)      |Apply_Modifying_Retrieved_Case();|
(15)      |Parametric_Adaptation();|
(16)      |Constraints_Adaptation();|
(17)      |Evaluating_Solutions();|
(18)      return |Modified_Satisfied_Case|;
(19)      end
```

## 5. AN EXAMPLE RUN

This section describes the implementation of the CBR system and gives some examples. A sequence of screen shots is used to show how the system operates.

https://doi.org/10.1017/S0890060403173064 Published online by Cambridge University Press

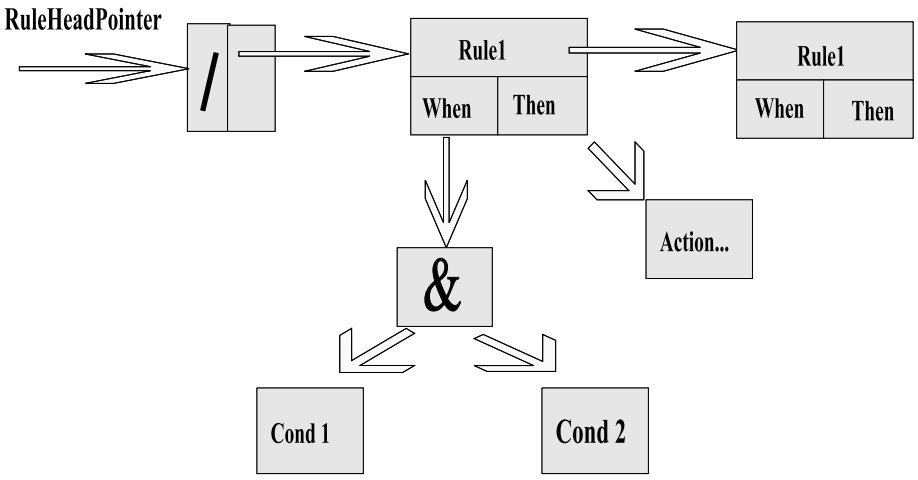


**Fig. 8.** The data structure of global and local rules.

### 5.1. Building the case repository

Initially, the system allows designer to choose the search method. This function provides the designer with the flexibility to search the case base according to his or her own needs. If the designer chooses "Search for matching case," the system will ask the designer to input problem definitions. If the designer selects "Search specifying indexes separately," the system will ask the designer to specify the indexes and their desired values. Figure 9 illustrates how to select searching methods. In this window example, we select "Search for matching case."

### 5.2. Problem specifications

#### *5.2.1. Global problem specifications*

Knowledge is acquired through user interaction, as shown in Figure 10. The designer is prompted to input the problem specifications, like shaft (bearing bore) diameters, load direction, allowed bearing axial/radial space, and the amount of loads. Here, different inputs will bring up various other windows and message boxes to indicate different reasoning results.

*Shaft (bearing bore) and shaft diameter*: The diameter is *d*_20_52, which means that the shaft diameter (bearing bore diameter) is 20 mm and housing diameter is 52 mm.

*Load direction*: Combined, which means that the loads applied on the bearing are combined (can be decomposed into an axial load and a radial load).

*Required radial space*: No, which means that bearing is designed without a radial space requirement, that is, it is rigidly mounted on the shaft.

*Required axial space*: No; see above explanation.

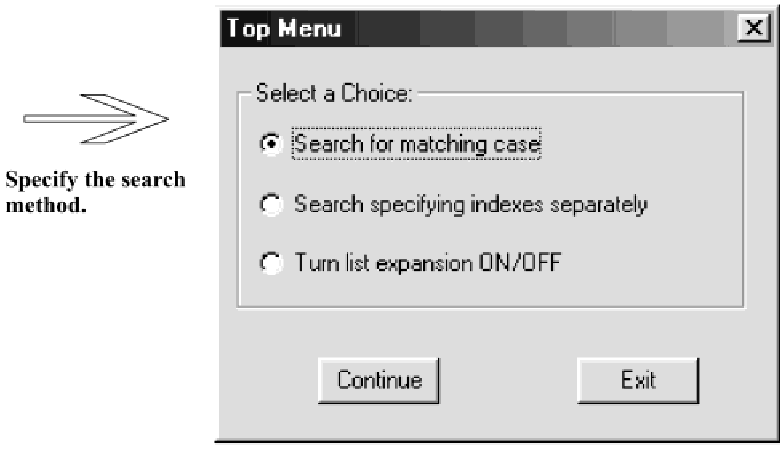


**Fig. 9.** A system overview: the window for selecting searching methods.

https://doi.org/10.1017/S0890060403173064 Published online by Cambridge University Press

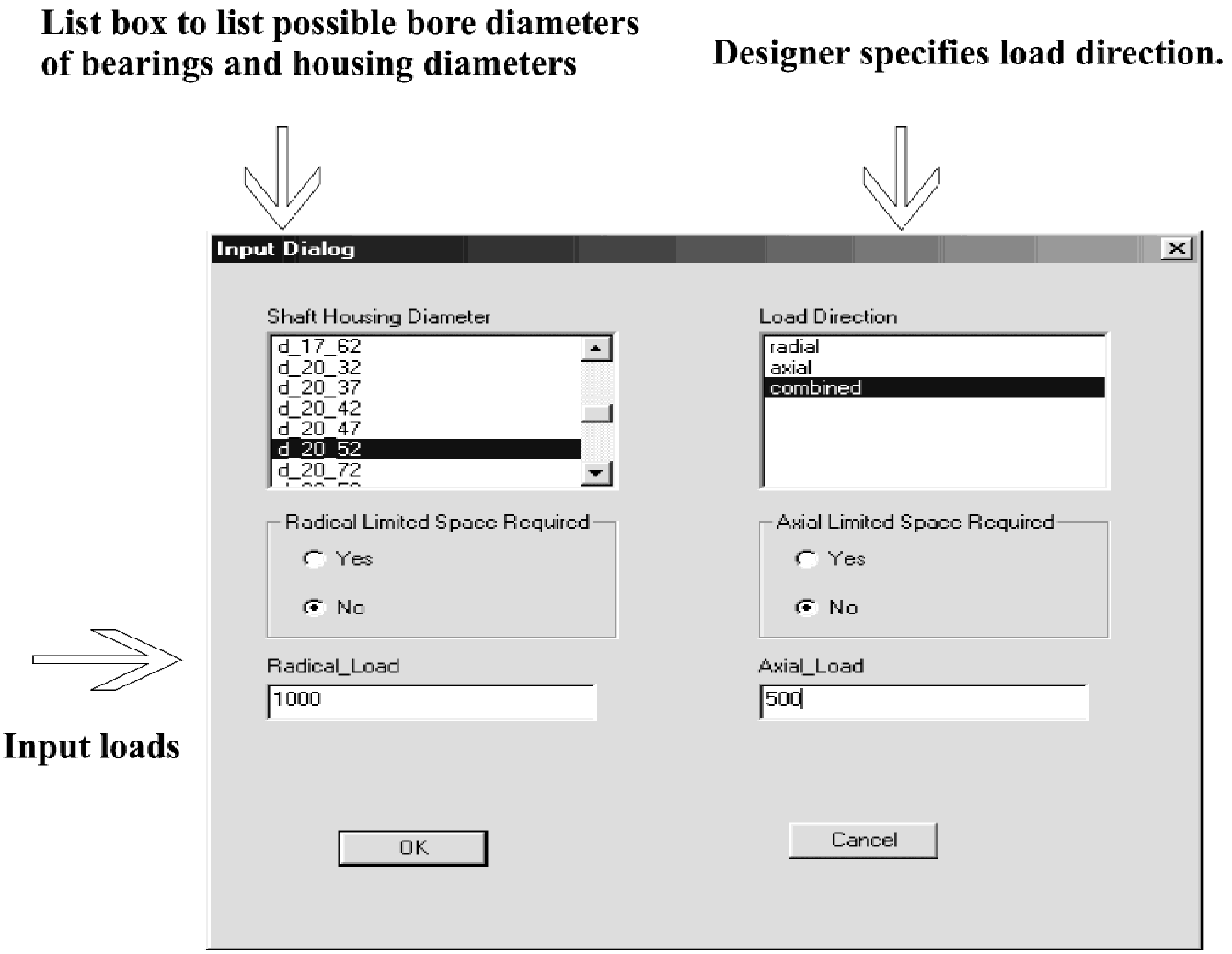


**Fig. 10.** A system overview: the window for problem specifications.

*Radial load*: 1000, which means the external radial load applied on the bearing is 1000 N.

*Axial load*: 500, which means the external axial load applied on the bearing is 500 N.

After the designer inputs all the above parameters, the system performs the following actions:

1. search the index list using the index features about the shaft (bearing bore) diameter and housing diameter (in this example, their values are 20mm and 52mm);
2. search the index list with the index feature load direction (in this example, its value is "combined");
3. link all the cases satisfy above indices requirements;
4. compare other features to each case selected according to index features;
5. calculate the weight of each case; and
6. list the ranking of each case.

### 5.3. Local problem specification

After the designer finishes the global problem specifications, another message box will be brought up. It asks whether the designer wants to use the "Weight Algorithm." If the designer's answer is "Yes," the system will perform actions based on the NNM algorithm (Kolodner, 1993). If the designer's answer is "No," the system will simple assign all the features' weight as 0 and find similar cases based on the numbers of matched features. Figure 11 shows a message box that asks the user whether to use the weight algorithm. In this window example, we select "Yes."

In our prototype, there are two local fields that define the specific features for each case. In this window, we input the

https://doi.org/10.1017/S0890060403173064 Published online by Cambridge University Press

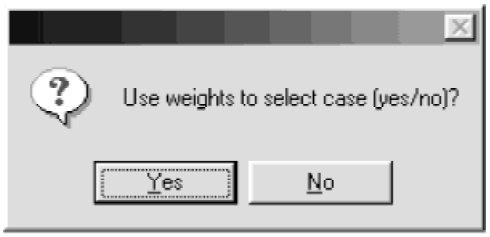


**Fig. 11.** A system overview: the window for the weight algorithm.

rotation speed of the shaft and the bearing life that the designer expects. See Figure 12 to input local problem specifications.

### 5.4. Case adaptation

Although the details of the adaptation procedures are hidden from the designer, the system presents a series of message boxes that indicate which case it is using to performing adaptation. In addition, the system keeps track of which cases that have failed during adaptation. This loop continues until the system finds a case that satisfies the problem specification or announces it failed to find any case that could fit the current problem (shown in Fig. 13).

After the system has found a set of retrieved cases and performed successful adaptation on one of them, it automatically returns the adapted case. The system can also return a successful or failed case to the designer, allowing the designer to understand why the case is successful or why it failed. Hence, the designer can use these cases as a starting point for creating new designs. Figure 14 shows how the adaptation of successful cases is tracked. Figure 15 shows a case that failed adaptation.

## 6. CONCLUSIONS, CONTRIBUTIONS, AND FUTURE WORK

This paper presented a system that uses CBR as both a cognitive model and problem-solving methodology to deal

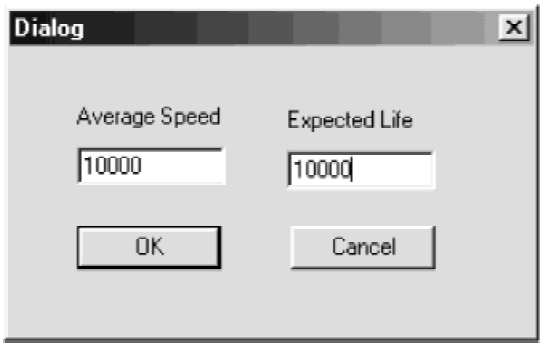


**Fig. 12.** A system overview: the window for the input of local problem specifications.



**Fig. 13.** A system overview: the message box shows the system is performing an adaptation on a retrieved case.

with the bearing design problem found in mechanical design. We believe that this work has produced several insights into how AI and CBR techniques can be better applied to more realistic engineering problems:

1. *Knowledge capture:* Because the knowledge space for the bearing design domain is extremely incomplete and dynamic, it is difficult to formalize general, *a priori*, rules to help the designer solve problems or automate the design process. In contrast, by using CBR techniques, a set of bearing design experiences can be stored in a case library to guide the designer. Through building a knowledge acquisition system, an autonomous CBR intelligent system can evolve and grow more easily than a traditional, knowledge-based system.
2. *Adaptability:* CBR techniques can integrate knowledge acquisition, reasoning mechanisms, knowledge storage, and learning in one platform. Therefore, a system using CBR techniques can possibly grow and be expanded to encompass a wider variety of assemblies without changing the fundamental system structure.
3. *Augmenting intelligence:* Our system, rather than being completely autonomous, interacts with the user to obtain knowledge. It provides the flexibility to draw design conclusions either from the reasoning system itself automatically or by allowing the designer to directly choose a past case as his problem solution.
4. *Human-guided search:* Our system also provides the flexibility to allow the designer to loosen index constraints to continue reasoning when an exact search fails. In this manner, the designer has the most opportunities to obtain a design solution that is useful for the current problem. This solution also can be used as a reference for her current design.

https://doi.org/10.1017/S0890060403173064 Published online by Cambridge University Press

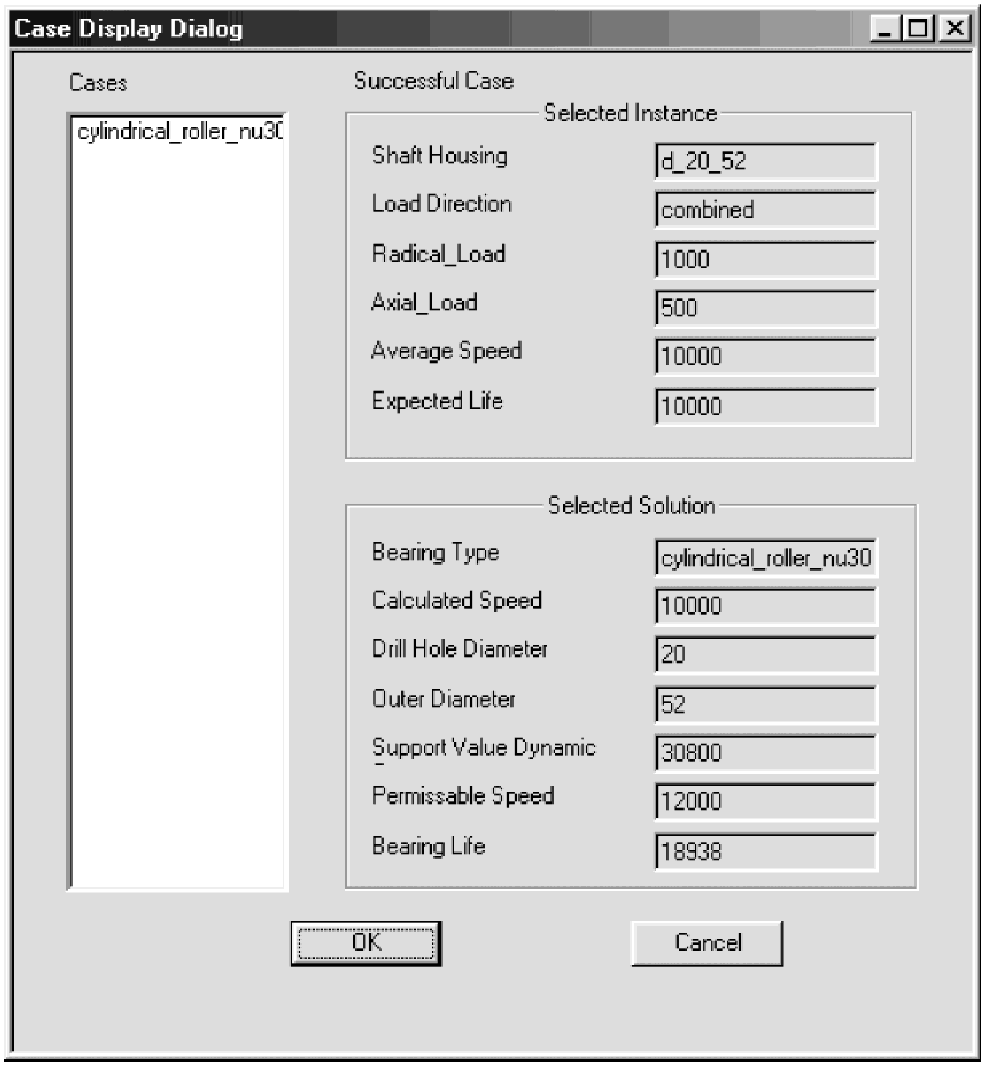


**Fig. 14.** A system overview: the window shows the successful case.

The contributions of this research touch on both AI/CBR and engineering design. We view the system for CBR design as a template for other CBR environments to create design aides focused on different design problems. We see the following areas as opportunities for future research:

1. *Knowledge engineering:* Because of the limitations of the CASL used to build our system, there are still many limitations in expressing design intent. The case collection process is quite complicated and inefficient and case-base maintenance is very unstructured, which makes debugging the case base very difficult. Better methodologies for case collection and good protocols to maintain the case base are needed.
2. *Knowledge acquisition:* We built attribute (features) pairs at design time to allow the user to interactively input this knowledge. For larger problems, autonomous knowledge acquisition system will become important.
3. *Indexing:* We built a fixed feature-based index scheme at design time to speed up searching. Scaling the system would require a more dynamic index scheme and more flexibility in feature specification.
4. *Intelligent CAD:* Because almost every designer uses CAD or other graphical software to conduct the design, a future goal is to better integrate CBR tools with CAD tools.
5. *Cross-domain reasoning:* The system presented in this article operates in a very specific domain; expansion of this system to other similar design domains is an important area to explore as well. Because we will correspondingly need to develop cross-domain knowledge representations and adaptations, a cross-domain

https://doi.org/10.1017/S0890060403173064 Published online by Cambridge University Press

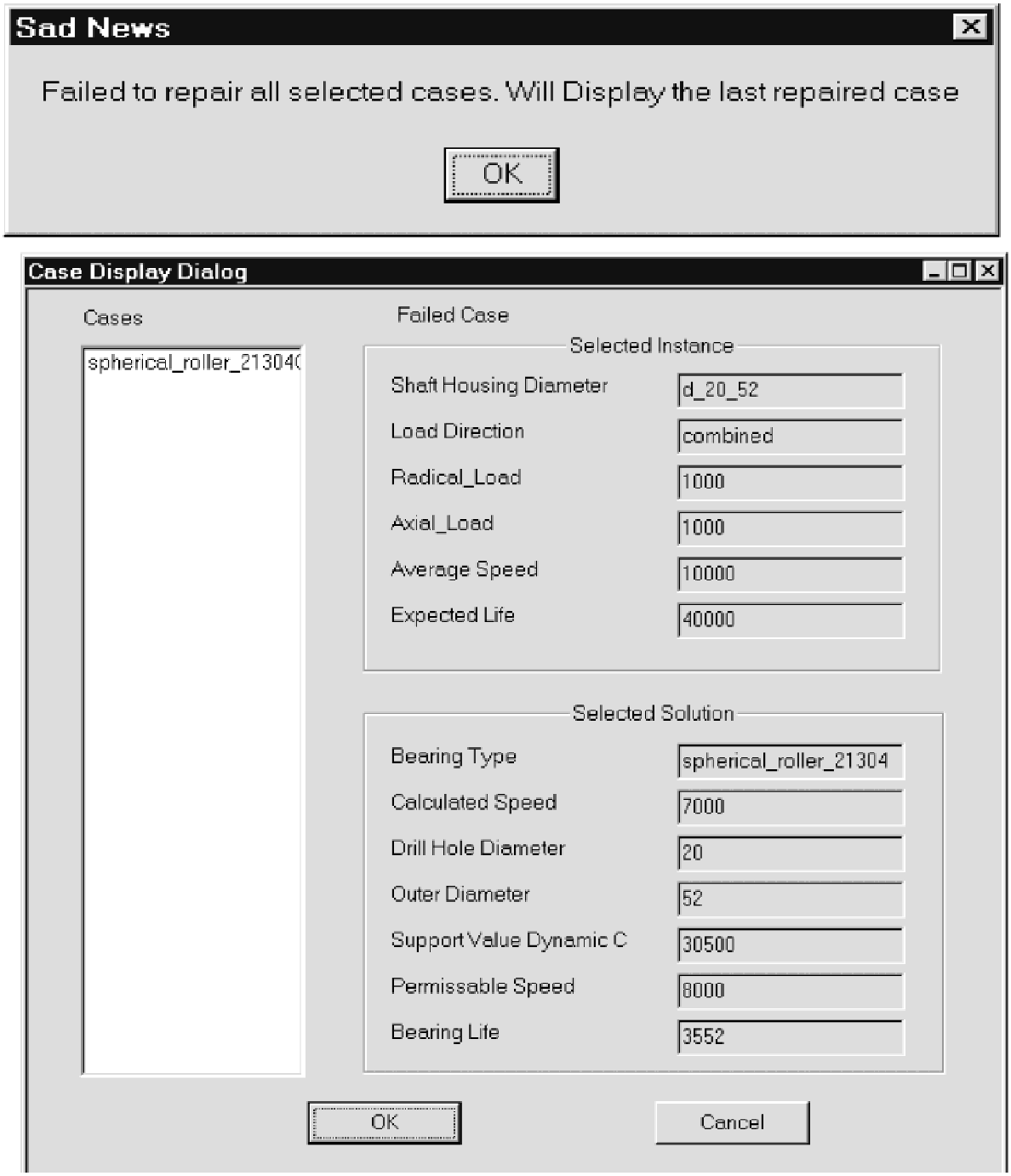


**Fig. 15.** A system overview: the window shows the failed case.

reasoning system becomes very complicated but also very useful.

## ACKNOWLEDGMENTS

This work was supported in part by National Science Foundation Knowledge and Distributed Intelligence in the Information Age Initiative Grant (CISE/IIS-9873005); a CAREER Award (CISE/IIS-9733545), an engineering Grant (ENG/DMI-9713718), and an Office of Naval Research Grant (N00014-01-1-0618).

Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation, Office of Naval Research, or the other supporting government and corporate organizations.

## REFERENCES


Aamodt, A., & Plaza, E. (1994). Case-based reasoning: Foundational issues, methodological variations, and system approaches. *Artificial Intelligence Communications 7(1)*, 39–59.

Bardasz, T., & Zeid, I. (1991). Applying analogical problem solving to mechanical design. *Computer Aided Design 23(3)*, 202–212.

Bardasz, T., & Zeid, I. (1992). Cognitive models of memory for mechanical design problems. *Computer Aided Design 24(6)*, 327–342.

Bhatta, S., & Goel, A. (1994). Discovery of physical principles from design experiences. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing 8(2)* [Special issue on Machine Learning in Design] Available on-line at ftp://ftp.cc.gatech.edu/pub/ai/students/bhatta/dp-aiedam94.ps

Bose, A., Gini, M., & Riley, D. (1997). A case-based approach to planar linkage design. *Artificial Intelligence in Engineering 11(2)*, 107–119.

Brown, D.C., & Birmingham, W.P. (1997). Understanding the nature of design. *IEEE Expert and Intelligent Systems 12(2)*, 14–16.

Center for Intelligent Systems, University of Wales (1999). Available online at http://www.aber.ac.uk/dcswww/Research/arg/cbrprojects/getting_caspian.shtml

Chandrasekaran, B., Goel, A.K., & Iwasaki, Y. (1993). Functional representation as design rationale. *IEEE Computer 26(1)*, 48–56.

Domeshek, E., & Kolodner, J. (1997). The designer's muse. In *Issues and Applications of Case-Based Reasoning in Design* (Maher, M.L., & Pu, P., Eds.), pp. 11–38. Hillsdale, NJ: Erlbaum.

Fowler, J.E. (1996). Variant design for mechanical artifacts: A state-of-the-art survey. *Engineering with Computers 12*, 1–15.

Goel, A. (1997). Design, analogy, and creativity. *IEEE Expert and Intelligent Systems 12(3)*, 62–70.

Goel, A., Bhatta, S., & Stroulia, E. (1996). KRITIK: An early case-based design system. In *Issues and Applications of Case-Based Reasoning to design* (Maher, M.L., & Pu, P., Eds.). Hillsdale, NJ: Erlbaum. Available on-line at ftp://ftp.cc.gatech.edu/pub/ai/goel/murdock/kritik.ps

Goel, A., Gomez de Silva Garza, A., Grue, N., Murdock, J.W., Recker, M., & Govindaraj, T. (1996). Explanatory interface in interactive design environments. In *Fourth Int. Conf. Artificial Intelligence in Design, AID '96* (Gero, J.S., & Sudweeks, F., Eds.). Boston: Kluwer Academic. Available on-line at ftp://ftp.cc.gatech.edu/pub/ai/goel/murdock/aid96.ps

Goel, A., & Stroulia, E. (1996). Functional device models and model-based diagnosis in adaptive design. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing 10*, 217.

Goldman, R.P., & Boddy, M.S. (1997). A constraint-based scheduler for batch manufacturing. *IEEE Expert and Intelligence Systems 12(1)*, 49–56.

Gomez de Silva Garza, A., & Maher, M. (1996). Design by interactive exploration using memory-based techniques. *Knowledge-Based Systems 9(3)*.

Hammond, K.J. (1989). *Case-based planning: Viewing planning as a memory task*. Boston: Harcourt Brace Jovanovich.

Hennessy, D., & Hinkle, D. (1992). Applying case-based reasoning to autoclave loading. *IEEE Expert and Intelligence Systems 7*, 21–26.

HEXAGON. (1999). *Bearing Calculation*. Available on-line at http://www.hexagon.de

Hinrichs, T., & Kolodner, J. (1991). The roles of adaptation in case-based design. *AAAI-91, Proc. Ninth National Conf. Artificial Intelligence*.

Hua, K., & Faltings, B. (1993). Exploring case-based building design-cadre. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing 7(2)*, 35–43.

Kim, G.J. (1997). Case-based design for assembly. *Computer Aided Design 29(7)*, 497–506.

Kolodner, J.L. (1993). *Case-Based Reasoning*. San Mateo, CA: Morgan Kaufmann.

Lambright, J.P., & Ume, C. (1996). A flat composite panel design advisory systems using knowledge based and case based reasoning. *Transactions of the ASME, Journal of Mechanical Design 118*, 461–469.

Leake, D.B. (Ed.). (1996). *Case-Based Reasoning: Experiences, Lessons, and Future Directions*. New York: AAAI Press/ MIT Press.

Lee, J. (1997). Design rationale systems: Understanding the issues. *IEEE Expert and Intelligence Systems 12(3)*, 78–85.

Maher, M., & Gomez de Silva Garza, A. (1996). Developing case-based reasoning for structural design. *IEEE Expert and Intelligent Systems 11(3)*.

Maher, M., & Zhang, D. (1993). Cadsyn: A case-based design process model. *Artificial Intelligence for Engineering, Design, and Manufacturing 7(2)*, 97–110.

Maher, M.L., Balachandran, M.B., & Zhang, D.M. (1995). *Case-Based Reasoning in Design*. Mahwah, NJ: Erlbaum.

Maher, M.L., & Gomez de Silva Garza, A. (1997). Case-based reasoning in design. *IEEE Expert and Intelligent Systems 12(2)*, 34–41.

Miyashita, K., & Sycara, K. (1993). Case-based incremental schedule revision. In *Knowledge-Based Scheduling* (Fox, M., & Zweben, M., Eds.). San Mateo, CA: Morgan Kaufmann.

Navin-Chandra, D. (1992*a*). Innovative design systems, where are we and where do we go from here? Part I: Design by association. *Knowledge Engineering Review 7(3)*, 183–213.

Navin-Chandra, D. (1992*b*). Innovative design systems, where are we and where do we go from here? Part II: Design by exploration. *Knowledge Engineering Review 7(4)*.

Navin-Chandra, D., Sycara, K.P., & Narasimhan, S. (1991). Behavioral synthesis in CADET, a case-based design tool. *Proc. Seventh Conf. Artificial Intelligence Applications*, pp. 217–221, Miami, FL, April 1991. New York: IEEE.

Pu, P. (1993). Introduction: Issues in case-based design systems. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing 7(2)*, 79–85.

Pu, P., & Reschberger, M. (1991*a*). Assembly sequence planning using case-based reasoning techniques. *First Int. Conf. Artificial Intelligence in Design*, Edinburgh, United Kingdom.

Pu, P., & Reschberger, M. (1991*b*). Case-based assembly planning. *1991 DARPA Workshop on Case-Based Reasoning*, Washington, DC.

Riesbeck, C.K., & Schank, R.C. (1989). *Inside Case-Based Reasoning*. Hillsdale, NJ: Erlbaum.

Sauer, J., & Bruns, R. (1997). Knowledge-based scheduling systems in industry and medicine. *IEEE Expert and Intelligent Systems 12(1)*, 24–31.

Shi, Z., Zhou, H., & Wang, J. (1997). Applying case-based reasoning to engine oil design. *Artificial Intelligence in Engineering 11(2)*, 167–172.

Silverman, B.G., & Mezher, T.M. (1992). Expert critics in engineering design: Lessons learned and research needs. *AI Magazine 13(1)*, 45–62.

Slade, S. (1991). Case-based reasoning: A research paradigm. *AI Magazine*, 42–55.

Smithers, T. (1989). AI-based design versus geometry-based design or why design cannot be supported by geometry alone. *Computer-Aided Design 21(8)*, 141–149.

Sycara, K., & Navin-Chandra, D. (1992). Retrieval strategies in a case-based design system. In *Artificial Intelligence in Engineering Design* (Tong, C., & Sriram, D., Eds.), Vol. II. New York: Academic.

Sycara, K., NavinChandra, D., Guttal, R., Koning, J., & Narasimhan, S. (1992). CADET: A case-based synthesis tool for engineering design. *International Journal of Expert Systems 4(2)*, 157–188.

Tsatsoulis, C., & Kashyap, R. (1993). Case-based reasoning and learning in manufacturing with the TOLTEC planner. *IEEE Transactions on Systems, Man and Cybernetics 23(4)*, 1010–1023.

Umeda, Y., & Tomiyama, T. (1997). Functional reasoning in design. *IEEE Expert and Intelligent Systems 12(2)*, 42–48.

Voss, A. (1997). Case design specialists in fabel. In *Issues and Applications of Case-Based Reasoning in Design*. (Maher, M.L., & Pu, P., Eds.), pp. 11–38. Hillsdale, NJ: Erlbaum.

Wielinga, B., & Schreiber, G. (1997). Configuration-design problem solving. *IEEE Expert and Intelligent Systems 12(2)*, 49–56.

Wilcock, D.F., & Booser, E. (1957). *Bearing Design and Applications*. New York: McGraw–Hill.


---

**Xiaoli Qin** received her MS in computer science from Drexel University in Philadelphia, PA. Her main research interests are in knowledge-based systems, knowledge acquisition, knowledge representation, machine learning, and intelligent database applications, particularly as applied to engineering design problems.

**William C. Regli** is an Associate Professor in the Department of Computer Science at Drexel University. He

https://doi.org/10.1017/S0890060403173064 Published online by Cambridge University Press

holds a courtesy appointment in the Department of Mechanical Engineering and Mechanics and is Director of Drexel's Geometric and Intelligent Computing Laboratory. Dr. Regli received his PhD in computer science in 1995 from the University of Maryland at College Park and his BS (cum laude) in mathematics and computer science in 1989 from Saint Joseph's University in Philadelphia. He is the recipient of a 1998 National Science Foundation CAREER Award, the University of Maryland Institute for Systems Research Outstanding Graduate Student Award (1994–1995), a NIST Special Service Award (1995), and a General Electric Corporation Teaching Incentive Grant (1994–1995), among other awards. He is a member of ACM, IEEE Computer Society, AAAI, and Sigma Xi. Dr. Regli has authored or coauthored more than 100 technical publications.

https://doi.org/10.1017/S0890060403173064 Published online by Cambridge University Press