Case-based reasoning disassembly system

Ibrahim Zeid, Surendra M. Gupta, and Li Pan
Laboratory for Responsible Manufacturing
334 SN, Department of MIME
Northeastern University
360 Huntington Avenue
Boston, MA 02115

ABSTRACT

This paper presents a new approach to address the problem of Planning for Disassembly (PFD). The approach is based on the Case-based reasoning technique. To assist planners to solve PFD problems, a system must have some heuristics and domain specific knowledge, which is related to the representation of the disassembly knowledge. In previous work, the authors suggested to use EMOPs (Eposodic Memory Organization Packet) for the knowledge representation of the PFD plan. This paper demonstrates the implementation of the EMOP memory model. The model has been implemented in C++, and tested. An example is presented to demonstrate the capabilities of the memory model.

Keywords: Disassembly, Planning for disassembly, Case-based reasoning, EMOP memory model, Knowledge representation.

1. INTRODUCTION

Traditional manufacturing systems consist only of a one way process, i.e., using different parts to assemble products. When the products reach their life expectancy, or are replaced by new products, they are discarded and sent to landfills. Research on disassembly has evolved as people begin to be concerned about the capacity of the environment to sustain such activities. Government is taking steps to limit such activities by increasing disposal taxes, setting regulations to force manufacturers to take back their products at the end of their lives, and reuse their materials. These changes lead to two-way operations, viz., assembly and disassembly, leading to a new manufacturing paradigm.

Many currently recycled products were designed more than a decade ago. There were no considerations during the design phase of the products about an easy, low cost disassembly process. How to efficiently disassemble these products needs the knowledge of "Planning For Disassembly" (PFD).

PFD identifies efficient sequences to disassemble products. A PFD problem can be defined as - Given a product structure, a disassembly goal, and a set of disassembly constraints, find a plan to successfully disassemble the product[1]. The disassembly plan has to satisfy these goals and constraints. The solution (or disassembly plan) depends on how well the problem is defined according to the goals, constraints and product structure. The solution may require optimization and improvements. Even if the problem is well defined, there may still be multiple solutions to satisfy the goals and constraints. This characteristic is defined as open-ended and hence may require a few iterations to obtain the final optimized disassembly plan. Open endedness and iterations are the two main characteristics of the PFD problem. This is partially related to the inherent nature of the problem and partially due to the flexibility of disassembly plans[1].

The identification of alternative solutions and the selection of the optimal one (based on quantitative analysis) are perhaps the most important steps in finding a solution to a PFD problem. This issue is related to knowledge reuse. In this area a lot of research has been done by using case-based reasoning (CBR) to solve new problems of the same class. The episodic memory organization packet (EMOP) is the most pervasive memory structure in CBR for knowledge expression. For example, Bardasz and Zeid[2] have applied it to mechanical design. Zeid et al.[1] proposed to use EMOP in disassembly planning. This paper demonstrates its implementation, which involves building a memory model to save and retrieve the disassembly plans.

The research activities on PFD are relatively limited and focus on disassembly sequences generation, strategies, simulation, and analysis. Penev and deRon[3] use the theory of graphs and the method of dynamic programming to generate and evaluate

the feasibility of disassembly plans. Johnson and Wang[4] establish four criteria to optimize the generation of the disassembly sequence for material recovery: material compatibility, clustering for disposal, concurrent disassembly operations, and maximizing yield. Yokota and Brough[5] use a precedence graph to express partial orders of disassembly. A hierarchical object representation is introduced as the basis of assembly/disassembly sequence planning. Arai, Uchiyama and Igoshi[6] introduce a part assemblability verification system by testing the product's disassemblability. Arai and Iwata[7] develop a CAD system that can assist designers in making a product assembly/disassembly plan. Kinematics simulation is used to construct the CAD system and select the best method for the disassembly sequence. Beasley and Martin[8] focus their discussion of disassembly sequences on the objects with only a finite number of unit cubes. Each disassembly step consists of one or two linear motions of single parts.

Gupta and his research team[9, 10] focus on operation planning issues and scheduling disassembly. Contrary to other researchers, they think that the disassembly sequence is not the exact opposite of the assembly sequence. In reference [9], they address some disassembly issues such as item segregation, reverse material flow, and item explosion. In reference [10], an algorithm that reverses the MPR (material Requirements Planning) is developed for scheduling the disassembly of those discrete-parts products characterized by a well-defined product structure. But the algorithm is not the reverse of the MRP algorithm, although the objective of the disassembly case is the reverse of the MRP. It is considerably more complicated.

All of the above studies place emphasis on the cost analysis and disassembly sequences generation for geometric or product structure constraints. None of these efforts addressed the issue of developing a computational model to either automate disassembly planning or to capture past disassembly knowledge for PFD. Zeid et al.[1] and Gupta and Veerakamolmal[11] are the only authors who consider this issue. They use case-based reasoning to assist planners to solve PFD problems.

For a more detailed review of the literature on disassembly, recycling and product recovery, see Gungor and Gupta[12]. Moyer and Gupta[13] present a state-of-the-art review on operational issues of disassembly and recycling in the electronics industry.

Considering the nature of the PFD problem (open ended and iterative) and the previous research efforts, it's obvious that the PFD knowledge is product specific. The knowledge of disassembling a product can be used for similar products. In order to use previous knowledge, we propose a memory model that uses disassembly features as indices to retrieve or store disassembly plans from a knowledge base. Using hash tables, the memory model can save a specific disassembly sequence for one product into files, then store them in the knowledge base. The disassembly sequences can be retrieved for later use.

2. DISASSEMBLY KNOWLEDGE REPRESENTATION

The desired memory model of the proposed knowledge base should be able to store and retrieve disassembly plans properly. The features used to index the memory model are critical to its successful implementation. It is especially important that the features are generic enough so the memory model is domain independent and can be used in any kind of product domain. Disassembling a product is easy when the product structure is given, and is difficult when the product structure is unknown. Suppose that the structure of a lamp is as shown in figure 1.

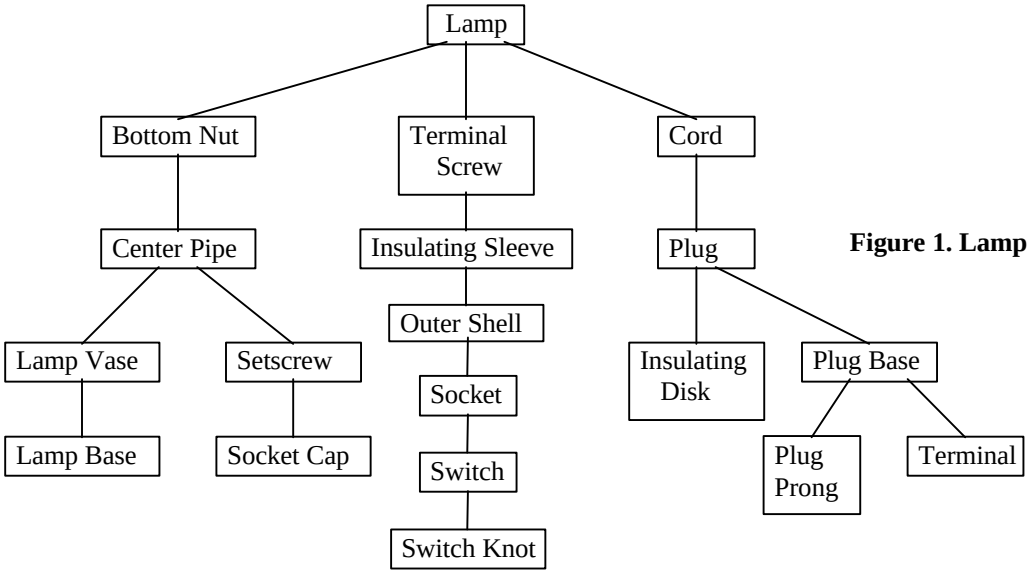

Figure 1. Lamp Structure Tree

If the disassembly goal is to get the Lamp Vase, the disassembly sequence should be: 1st, take off the Bottom Nut; 2nd, take out the Center Pipe, 3rd, take off the Lamp Vase from the Lamp Base. If the disassembling goal is to get the Socket with Switch on it, the disassembly procedure may look like: 1st, take off the Socket assembly by unscrewing the Terminal Screw; 2nd, separate the Outer Shell and Insulating Sleeve from the Socket assembly. Obviously, different disassembly goals will generate different disassembly plans. A disassembly plan describes the disassembly sequence a disassembler needs to know.

By observing several products such as table lamps, cars, and window fans, one can conclude that most products have assemblies and parts. An assembly may have subassemblies and parts; a subassembly may have further subassemblies and parts. Different products usually have different structures and thus a different disassembly sequence may be needed. Sometimes, even a subassembly can be treated as a smaller assembly. Assemblies and subassemblies are really overlapped in categorization. For the memory model implementation, four levels have been defined. They are product, assembly, component and parts. A product level contains all artifacts that can be used by the customer directly. An assembly level contains artifacts that make up a product or an assembly. Since there are many more assemblies than products, it is better to divide the assembly level into two levels for a better memory model implementation. A component level is defined as a level that contains artifacts made up of all parts. A part level contains artifacts that cannot be further disassembled. In this paper, product's name, model, and type are used to distinguish products. By using indices of product name, type, model, assembly, component, part and constraints, the disassembly plan can be stored in a database, and the same indices are used to retrieve the disassembly plans.

The common features for a product (based on the above discussion) can be generated as shown below:
Constraint: reuse or recycle
Level: product
Name: jeep
Type: two-stroke engine
Model: Grand Cherokee

Similar features have been developed for an assembly, a component, or a part. All these features are listed in Table 1.

Table 1. Index Types

Product Indices	Assembly Indices	Component Indices	Part Indices
Constraint	Constraint	Constraint	Constraint
Level	Level	Level	Level
Name	Name	Name	Name
Type	Type	Type	Type
Model	Model	Model	Model
	Parent model	Parent model	Parent model

In this paper, we utilize an EMOP memory model[14, 15, 16] to represent the identified disassembly knowledge. The knowledge is represented in the model as disassembly plans, one plan per product. Each plan is a hierarchical tree that represents the product structure that can be traversed to generate disassembly plans. The memory model design is based on the concept that the current problem-solving plan can benefit from the experience of past similar problem solving plans. EMOPs are indexed in the memory by the features they describe. EMOPs that store general information can be treated as nodes, while EMOPs that store particular PFD plans can be treated as leaves.

The proposed memory model has the following characteristics: 1) The model is dynamic and will grow bigger as disassembly plans are accumulated. 2) The size of the memory model is hard to predetermine. 3) Both the fast retrieval and the fast storage are needed. 4) Occupy as less computer memory space as possible. Considering all of these, the memory model is organized in four parallel levels. From top to bottom, they are: product level, assembly level, component level and part level. In each level, the memory model is designed in a tree-like structure.

It is obvious that a non-destructive disassembly plan will differ from a destructive disassembly plan. Taking the Constraint Feature into account, the designed memory model actually consists of eight categories, which are combinations of levels and constraints. Figure 2 shows the structure of the EMOP memory model that uses these eight categories.

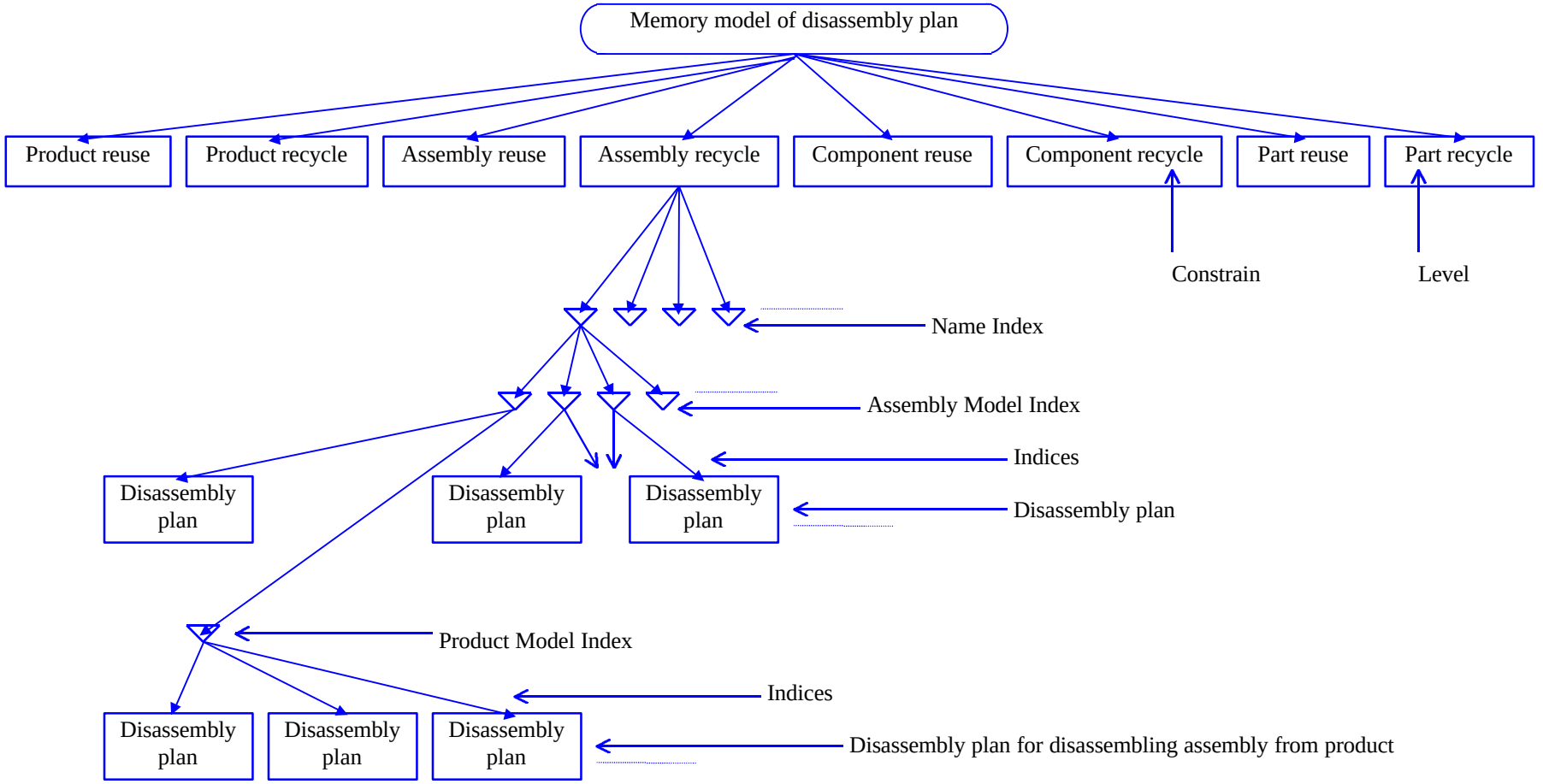

Figure 2. The EMOP Memory Model for PFD

This memory model supports five general operations. Under each general operation except for the display operation, several specified operations are supported. The following lists these operations:

Search for and retrieve a disassembly plan
- Search disassembly plan by features
- Search for a disassembly plan
- Search a disassembly plan to disassemble an assembly from a product (a component from an assembly, or a part from a component)
- Search a disassembly plan by artifact structure

Update a disassembly plan
- Modify a disassembly plan
- Modify a disassembly plan to disassemble an assembly from a product (a component from an assembly, or a part from a component)

Add a new disassembly plan
- Add a new disassembly plan
- Add a new disassembly plan to disassemble an assembly from a product (a component from an assembly, or a part from a component)

Remove a disassembly plan
- Remove a disassembly plan
- Remove a disassembly plan to disassemble an assembly from a product (a component from an assembly, or a part from a component)

Display all disassembly plans in a category

3. IMPLEMENTATION

The memory model is implemented using hash tables[17]. For each hash table, a link list is formed. The entire program consists of five classes. They include:

Class of hashtable; a class defining the common characteristics of all hash tables. The hash table contains an array of hashbuckets, into which entries are stored. Each hash table is an EMOP that is an organizer. Each hash table organizes one of the eight categories. New disassembly plan can be added according to the combination of a mechanical artifact's level and disassembling operation constraints. Existing disassembly plan can be deleted by indices of a mechanical artifact's level and constraint. Search can be performed using the same indices as adding and deleting.

Class of hashbucket; a class defining the bucket. Each bucket contains a linked list of hashentry pointers. A hashbucket is a SUBEMOP, which organizes a series of disassembly plans. Every mechanical artifact's name is hashed by the hash function. According to the hashing result, mechanical artifacts are distributed evenly into hashbuckets. In each hashbucket, the mechanical artifacts are arranged in alphabetical order. New disassembly plans can be added by mechanical artifacts' name. Existing disassembly plans can be deleted by the index of name, and search can be performed for an index of a given name.

Class of hashentry (Name-Node) a class defining the features of entries in a bucket's linked list. Each entry (node) contains a mechanical artifact's name, which distinguishes it from others. Every hashentry (node) contains a few model indices.

Class of *myfile* (Model-Node) a class defining the features of Model-Node. Each node contains a mechanical artifact model, which can be used to identify this mechanical artifact from other similar ones. It also contains the disassembly plan of this model. Under each model node, a few of model indices are contained.

Class of *fromfile* (DPfile-Node) a class defining the features of DPfile-Node. Each node contains a parent model, which can be distinguished from other siblings. Each node contains a disassembly plan of disassembling a mechanical artifact from its parent mechanical artifact. The parent type of a mechanical artifact is stored. The feature of parent type can be used to help the search algorithm to retrieve a better disassembly plan whenever a product structure is given.

4. EXAMPLE

To illustrate the use of the proposed memory model, the disassembly plan of a two-stroke engine is considered. Section 4.1 shows the memory model after the two-stroke engine is added to the system. A retrieval request is discussed in Section 4.2.

4.1. Two-Stroke Engine

Two-stroke engine is an assembly that is a part of lawn mover. The two-stroke engine is composed of carburetor, crankcase, crankshaft, muffler, cylinder head, spark plug, air shroud, reed valve and piston. The "*myfile*" is the one that has the disassembly plan of the entire engine assembly. The "*fromfile*" is the one that has the disassembly plan of the muffler from the engine, the piston from the engine, or the air shroud from the engine, etc. After inputting the two-stroke engine data into the memory model, the memory model's graphical representation is shown in figure 3. Table 2 is the content frame for a two-stroke engine.

Table 2. Features for two-stroke engine

Feature Type	Feature Value
Constraint	reuse
Level	assembly
Name	engine
Type	two-stroke
Model	engine-201
Parent Model	lawn-mower-200

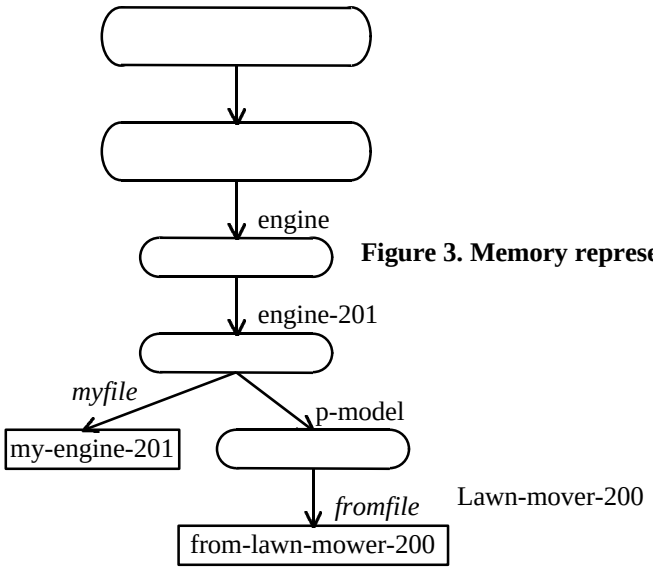

Figure 3. Memory representation after storing two-stroke engine

4.2 Retrieve the Disassembly Plans

In this section we discuss a retrieval request. The following query requests to retrieve the disassembly plans under the REUSE constraint. So these plans are for reusing operations, not recycling operations. The query is: **Retrieve the disassembly plan of a four-stroke engine.** A four-stroke engine differs from a two-stroke engine. The major distinction is the position of the carburetor and air filter. In the two-stroke engine, the air-fuel mixture enters the engine through the crankcase; the intake port (and thus the carburetor and air filter) is at the base of the cylinder and crankcase. For the four-stroke engine, the intake port is at the cylinder head. Otherwise, the parts are identical. A flexible fuel line connects the fuel tank to the carburetor; a flywheel, covered by a metal shroud, is attached to one end of the crankshaft; a muffler covers the exhaust port; and a spark plug is screwed into the cylinder head[18]. The features of four-stroke engine are in table 3.

Table 3. Features for four-stroke engine

Feature Type	Feature Value
Constraint	reuse
Level	assembly
Name	engine
Type	four-stroke
Model	engine-401
Parent Model	garden-tiller-400

The model will search all the disassembly plans under name ENGINE. Since there is no disassembly plan for four-stroke engine which has model ENGINE-401, the memory model will tell the user that the disassembly plan for four-stroke engine is not available. The user has three choices. First, the user can choose a similar engine model if he/she knows how to choose one. The memory model will search again, and return a result of either found or not found. Second, the user can select to display all the disassembly plans under name ENGINE if the user does not know which one should be chosen. Finally, the user can input the structure of the four-stroke engine. The memory model will search and retune a result. If the user chooses to retrieve a disassembly plan of two-stroke engine instead, the model will return my-engine-201. After the file is returned, the model will ask if the user wants to modify the retrieved disassembly plan. If the user changes the disassembly plan, the new disassembly plan is stored into the memory model. In this example, the same disassembly plan is used since the two engines' structures are similar, which means the disassembly sequences will be the same. Hence the same disassembly plan will be referenced.

5. CONCLUSIONS

A memory model has been designed and implemented for efficient saving and retrieval of disassembly plans with the function to add/remove new/old disassembly plans. In designing the memory model, information of the products, such as name, type, model, parent model and level, are used as features to index the product memory model. This alleviates users from having to know the detail of mechanical artifacts' structure.

REFERENCES

1. Zeid, I., Gupta, S. M. and Bardasz, T., "A case-based reasoning approach to planning for disassembly", *Journal of Intelligent Manufacturing*, Vol. 8, No. 2, 97-106, 1997.
2. Bardasz, T. and Zeid, I, "DEJAVU: A case-based reasoning designer's assistant shell", *Proceedings of the International Conference on Artificial Intelligence in Design*, 477-496, 1992.
3. Penev, K. and DeRon, Ad J., "Determination of a disassembly strategy", *International Journal of Production Research*, Vol. 34, No. 2, 495-506, 1996.
4. Johnson, M.R. and Wang, M.H., "Planning product disassembly for material recovery opportunities", *International Journal of Production Research*, Vol. 33, No. 11, 3119-3141, 1995.
5. Yokota, K. and Brough, D.R., "Assembly/Disassembly sequence planning", *Assembly Automation*, Vol. 12, No.3, 31-38, 1992.
6. Arai, E., Uchiyama, N. and Igoshi, M., "Disassembly Path Generation to verify the Assemblability of Mechanical Products", *JSME International Journal, Series C*, Vol. 38, No. 4, 805-810, 1995.

7. Arai, E. and Iwata, K., “CAD system with product assembly/disassembly planning function”, *Robotics & Computer-Integrated Manufacturing*, Vol. 10, No. 1/2, 41-48, 1993.
8. Beasley, D. and Martin, R.R., “Disassembly sequences for objects built from unit cubes”, *Computer-Aided Design*, Vol. 25, No. 12, 751-761, 1993.
9. Brennan, L., Gupta, S. M. and Taleb, K. N., “Operations planning Issues in an Assembly/Disassembly Environment”, *International Journal of Operations and Production Management*, Vol. 14, No.9, 57-67, 1994.
10. Gupta, S. M. and Taleb, K. N., “Scheduling Disassembly”, *International Journal of Production Research*, Vol. 32, No. 8, 1857-1866, 1994.
11. Gupta, S. M. and Veerakamolmal, P., “A Case-Based Reasoning Approach for Optimal Planning of Multi-Product/ Multi-Manufacturer Disassembly Processes”, *International Journal of Environmentally Conscious Design and Manufacturing*, Vol. 9, No. 1, 15-25, 2000.
12. Gungor, A. and Gupta, S. M., “Issues in Environmentally Conscious Manufacturing and Product Recovery: A Survey”, *Computers and Industrial Engineering*, Vol. 36, No. 4, 811-853, 1999.
13. Moyer, L. and Gupta, S. M., "Environmental Concerns and Recycling/ Disassembly Efforts in the Electronics Industry", *Journal of Electronics Manufacturing*, Vol. 7, No. 1, 1-22, 1997.
14. Schank, R.C., *Dynamic Memory*, Cambridge University Press, Cambridge1982.
15. Schank, R.C. and Adelson, R.P., *Scripts, Plans, Goals and Understanding*, Lawrence Erlebaum, Hillsdale, NJ, 1977.
16. Kolodner, J. L. *Retrieval and organizational strategies in conceptual memory: A computer model*, Lawrence Erlbaum, Hillsdale, NJ, 1984.
17. Thomas A. Standish, *Data Structures, Algorithms and Software Principles in C*, Addison Wesley, 1994.
18. The editors of TIME LIFE BOOKS, “Small Engines”, 1982.