

# Using Markov Chain and Nearest Neighbor Criteria in an Experience Based Study Planning System with Linear Time Search and Scalability

Juan Carlos Segura-Ramirez and Willie Chang  
California State University, Sacramento  
{segurarj, changw}@ecs.csus.edu

## Abstract

*Most automated rule-based expert systems developed to aid student study planning and advising have appeared to be ephemeral due to the dynamic property in the ever-changing curricular requirements and rules. We propose a novel case-based study planning system with the search criteria based on the experience-indicated probability in Markov chains and the nearest-neighbor measurement for matches. We provide query results of course sequences to students who need to meet certain constraints such as to graduate within a certain number of academic terms, maintaining a minimal grade-point average, etc., all drawn from past graduate records.*

*We use a collection of well-formed course-study sequences from past graduates as advised curriculum patterns for new students to reference as his or her study plan. When given a set of study constraints, We implement a type of case-based reasoning by searching for the nearest-neighbor vector indices in the past records first, then by selectively listing the similar records according to their Markov-chain probabilities passing a desired threshold. Hence, we identify the similarity in the historic records to the set of constraints queried with a desired probability.*

*The time complexity of computing the nearest-neighbor indices to find the maximum similarity can be very large. Our implementation method achieves a linear-time complexity in both searching and scaling the system. When updating with a new record, each parametric combination represented by a sorted list of the records is linearly looked up, and the new record value is inserted to keep the list sorted. Since each query input is a set of constraints in a pre-determined order, the parametric combinations has an associated sorted list to look up in a one-pass linear process.*

*The first-order Markov chains can also be updated with a linear time complexity whenever a new graduate record is introduced. The probability matrix is first looked up by row and then column, representing a pair of courses taken in two adjacent academic terms, and the look-up time is also linear.*

**Keywords:** *Markov chain, nearest neighbor, linear time, scalability, case-based reasoning, study planning, expert system*

## 1. Issues and Overview

Expert systems are software tools that provide help in decision making for complex problems in a variety of domains ranging from those in medical diagnostic applications to academic advising [1, 2]. There are many technical methods used to help decision makers. Decision Support System (DSS) and Case Base Reasoning System (CBRS) are two main categories. DSS helps the user analyze a given problem from different perspectives and identify the best possible options, while CBRS helps the user in finding solutions to a specific condition using previous experiences [3, 4].

Many have tried to automate student advising but in most cases the applications fall into a special type of DSS called Rule Base System (RBS). RBS has the ability to easily deduce new facts or events given a set of rules as its axioms [3], but this simplicity also creates a problem: at any given time more than one rule may apply, or different rules may apply in certain sequences, and this results in having to generate a set of rules for almost every case. In order to reduce the complexity, RBS must be given some control mechanism to determine the sequence of rules to apply for a given condition. Another challenge for RBS is the knowledge elicitation bottleneck caused by the lack of available expert in the field, and as a result, the knowledge engineer may not be able to interpret the problem, or let it be loosely linked to facilitate the extraction of rules [4]. RBS designed for student advising often faces the challenge of complexity and exceptions in interpreting and maintaining the rules given the ever-changing course numbers, prerequisites, and degree requirements; and as a result, many advising systems of this type have been proven ephemeral [3].

Providing historic successful cases as solutions to a given condition, a system combining DSS and CBRS principles, without seeking adaptation from rule derivation, seems to be a plausible way to tackle the

student-advising problem. One of the advantages is that the advising system does not need to extract rules from conflicting facts due to the changes in the courses, and the knowledge elicitation bottleneck that limits RBS is completely avoided. We develop a student Study Planning System (SPS) as an experiment to prove the viability of what is described above.

The goal of our work is to devise a CBRS to help students with deciding a curricular study pattern. The three different categories of decision-making systems are described below regarding their different approaches to the issue.

### 1.1 Decision Support System

Decision Support System (DSS) allows different users to analyze information and evaluate the assumptions underlying the use of specific models. DSS allows decision makers to analyze the information interactively in a manner that would usually help in leading to a particular decision [2, 9, 10].

One of the characteristics of a DSS is the ability to access data from a variety of sources. With this feature the user will have a nearly complete picture of the problem seen from different perspectives. Another characteristic of a DSS is the ability to transform enormous amount of "data" into "information" and allowing the decision maker to focus in what is important to his or her situation.

A DSS does not seek to find the optimal decision; instead a DSS helps the decision-maker to find a possible action that is good enough, given the limitations of the data. A DSS should provide a range of information without overwhelming the decision-maker; a rule of thumb called the "Seven plus or minus two rule" is used to prevent data overload. The rule states that a decision-maker can assimilate five to nine ideas before feeling overwhelmed.

Rasmussen [10] notes that decision makers used passed experiences as guidelines; that is, if a decision maker has faced a problem and experienced a good outcome, then he or she is likely to use similar approaches and techniques next time. Decision makers, in general, "look at the characteristics of an alternative and compare those to something they know and understand [10]."

### 1.2 Case-Base Reasoning System

Case-Base Reasoning System (CBRS) can be used when little concrete knowledge is available about a

domain but experts exist [1, 4, 5, 12, 13]. Among the advantages of CBRS are the following few:

- a. it can easily be made to learn. In CBR, problem-solving efforts can be saved for future use. CBRS can learn or adapt to small changes in their environment by continuing to collect cases and insert them into the case base library,
- b. by using CBRS, solutions can be justified by the cases they are derived from. CBRS can also illustrate the effects of a particular solution. The cases used to solve a problem justified the derived solutions and their probable outcomes,
- c. it can be designed to anticipate potential problems. An unsuccessful experience can be used to anticipate possible problems that may be the result from solving a problem in certain way,
- d. it provides a way for humans and computers to interact to solve a problem. CBRS is inspired by human behavior; people are good at creative adaptations but poor at remembering. Computers, on the other hand, are good at remembering and CBRS provides a way of using the best qualities of both human and computer,
- e. Knowledge acquisition for CBRS is natural. Experts are good at reporting experiences and CBRS can code each experiences as a case,
- f. it should be used when it is difficult to formulate domain rules due to incomplete, uncertain or inconsistent domain knowledge, or the problem is incompletely specified. Cases alone form the basis of the solution,
- g. it should be considered when generally applicable knowledge is insufficient to find a solution to a problem.

In order for CBRS to be effective, the following conditions must be observed [12, 13]:

- a. it requires cases. Most of the effort in CBRS goes to data collection,
- b. Cases with similar problems should have similar solutions. CBRS is based on the premise that situations occur in predictable ways. Adaptation modifies old solutions to fit new situations. Interpretation assumes old and new situations behave similarly,
- c. the solutions are not guaranteed to be optimal. CBRS may be usable if the heuristics can be adapted to a more thoroughly search.

### 1.3 Study Planning and Advising System

Next to grading, advising involves one of the most expensive tasks for a school. Helping students to determine long term goals and curriculum options can be rewarding but time consuming. Computers can be used to reduce the tedium of student advising [6, 7, 11].

An advising system should resolve many of the advising issues that concern students, like "which courses

should I take? ...What prerequisites are required for the courses I need to take?" [7] Any advising system must be able to adapt dynamically to a particular task, and it must provide the steps needed to accomplish a determined goal.

Previous attempts to create student-advising systems have been successful at achieving what is expected, but the maintenance is usually very cumbersome. The Rule base system has been a methodology used to create this type of system, but the generation of rules is usually very challenging given that catalogs change continuously. Another challenge of Rule Base Systems is all the exceptions and course validations, which make the maintenance very complicated. Most student-advising systems just provide a suggested curriculum pattern, a sequence of courses and it is up to the student to create his or her own course curriculum pattern based on his or her particular situation. This approach is very flexible but demands more attention from advisors.

## 2. Our Study Planning System

Our Study Planning System (SPS) is developed using the case base reasoning technique, which extracts a set of templates or cases based on the historic graduate records in a certain major. The query results for a study plan are based on the templates that match the given criteria of past graduate records for those who had:

- a. a minimum of transfer units or used catalogs (or catalogs within a limited number of years) or
- b. studied within a maximum number of semesters or
- c. studied with a maximum of units per semester or
- d. maintained a good GPA (or a minimum GPA in each academic term).

After SPS is in place a planner is shown to specify courses to be taken at certain semesters as the criteria to search in the templates for matches, and the records of those past graduates that match the criteria are the results of the desired study patterns. The goal at this stage is to provide "good" references from peers' individual experiences that can be invaluable. It doesn't need to deal with the complexity of pre/co-requisites chains, catalog rights, and exceptions. And as a planner freely adds or subtracts criteria (being specific or general), the patterns from the templates can converge or diverge in order to suit the needs.

### 2.1 Linear Time Complexity

To update the SPS nearest-neighbor indices with a new graduate record, each parametric combination represented by a sorted list of the records is linearly looked up, and the new record value is inserted to keep

the list remain sorted. Since each query input is a set of constraints in a preferred order, the combined constraint parameters represents one of the parametric combinations that has been built with an associated sorted list for match look-ups. Hence, the search time can be reduced to a linear process, and renders the time to obtain the result to be within  $|records|$ . And, adding a new record into the sorted indices has a time complexity  $C \times |records|$ , where  $C$  is the number of query constraints, a constant that is irrelevant to the number of records.

The first-order Markov chains are also being updated with linear time complexity whenever a new graduate record is introduced. The probability matrix is looked up first by row and then column, representing a pair of courses taken in two adjacent academic terms. The probability of taking a course  $B$  after taking a course  $A$  is the ratio of the number of such occurrences over the number of all occurrences that any course was taken after taking course  $A$  [15, 16]. The probabilistic results are only given as augmented values when the query specifies a probability threshold that is less than one. The default is to consider all existing pair in a sequence. Since the rows and columns are readily sorted by course ID's, the look-up time is within  $|row| + |columns|$ .

### 2.2 Nearest-Neighbor Measurement

The record templates provide all the information needed by the decision maker. In this implementation the template responds to the following questions:

- What overall GPA do I want to achieve?
- What GPA do I want to achieve per semester?
- In what number of semesters do I want to graduate?
- How many courses do I want to enroll in per semester?
- How many credits or units do I want to take per semester?

Furthermore, every template should provide a roadmap or a curriculum pattern that the user can study and eventually emulate. The template contains a detailed description of the courses passed by graduates per semester and year, as well as the grades and number of units per course. Just as important, the courses enroll in for a specific semester can be displayed and analyzed by the user.

Thus, SPS includes a case database in which all cases will be first validated. The table has these fields: student ID, course number and title, semester, year, grade, units, and sequential semester number.

The Euclidian distance was used to determine the most similar case to the user's constraints [8, 14].

$D(x,y) = \text{SQRT}(\text{SUM}(w_i * (x_i - y_i)^2))$  where  
 $w_i$  = weight assigned to a particular parameter  
 $x_i$  = parameter  $i$  of case  $x$   
 $y_i$  = parameter or constraint  $i$  of case to be matched ( $y$ ).

The algorithm is explicit. The distance between the average for every parameter in the template database and every case is computed. The result is stored in a field that represents all possible parameter combinations (Table 3). When the user enters a set of constraints the system retrieves the information sorted by the parameter combination. It then computes the distance between the parameters entered and the average for every parameter and locates the case closer to the distance previously computed and stored in the corresponding parameter combination field.

One of the biggest limitations of the nearest neighbor measure is the increase in computation time, which behaves linearly. Thus, this approach is more effective when the number of cases in the template database is considered small. To minimize the effect of this limitation, the distance between every case and the average for every parameter is computed only once before the user enters the parameters. In this manner, the nearest neighbor is not computed every time the user enters a set of parameters; only a linear search in the appropriate parameter combination field is needed to locate the nearest neighbor.

Another consideration when using this approach comes when determining the weights for every parameter. For this application every parameter was equally weighted. Given that there is no previous information about what are the best values to use to determine the weight, it was determined that no weights will be used and the user will have the possibility to sort the cases by the parameters more important to him or her.

### 2.3 Template Collection

The template database will be summarized and a set of indicators will be computed to answer the main questions or main query parameters. The query input constraint parameters are list below by their default significance, and first three are primary ones. Some parameters can also be queried by giving a range.

1. Number of credit units per semester,
2. Number of semesters to graduate,
3. Minimum GPA in each semester,
4. Number of courses enrolled per semester, and
5. Overall GPA at graduation.

The template construction starts from the collection of students' transcript. Each transcript is screened for consistency, completeness, and coherency. The transcripts are validated and loaded into the template

database. As the template database is being built, a first-order Markov chains and a set of indicators are also incrementally built to facilitate the query search (Figure 1).

The user query starts from reading the input parameters or curriculum pattern constraints. Afterwards, a set of templates closest to the user parameters by Euclidian distance is displayed, with the probabilities in the course sequence being within the given input. The templates are listed as entries sorted by the similarity to the user's parameters. The user can further examine the details of each entry (Figure 2).

SPS will have four major sections (Figure 3), the welcome page, the user page, the administrator page and the information page. The user section allows the user to query the template database. The administrator section allows the administrator to read and load data as well as run the maintenance routines. The information section allows the user to clarify any information in regard to the system usage as well as the concepts and computations.

### 2.5 Table Structures and User Interaction

There are several tables used in SPS. The first two tables described below are for referencing the course records of graduates: Templates (Table 1) and Template Indicator (Table 2). The Template Indicator contains summary information about the templates and will facilitate the retrieval. Templates and Template Indicator tables are both linked by student ID in a one-to-many relationship (Figure 4).

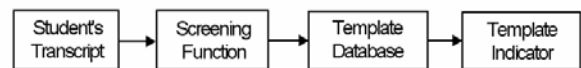


Figure 1. Template construction diagram of SPS.

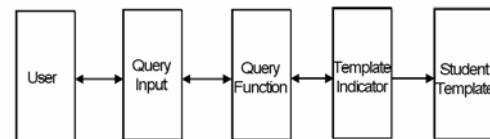


Figure 2. User interaction diagram of SPS.

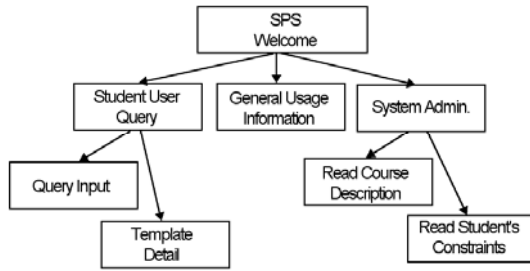


Figure 3. Screen navigation diagram.

Semester	Text	2	Semester the course was taken. S:Spring, F:Fall, V:Summer, O:Other
Year	Number	8	Calendar Year the class was taken
Grade	Text	2	Grade obtained by the student in the particular course
Units	Number	8	Number of units or credits approved by the student
NumSemester	Number	8	Sequential Semester Number

Table 1. SPS Template table.

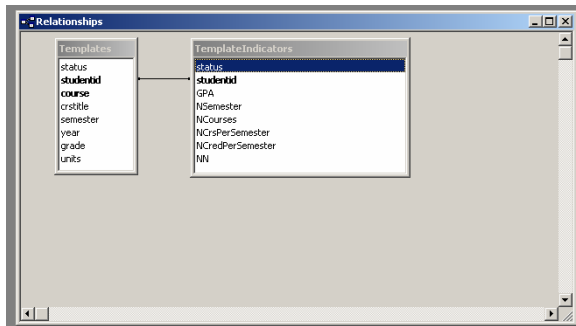


Figure 4. SPS Table relationship

Field Name	Field Type	Field Size	Field Description
Status	Text	1	If active (value=null) template will be considered for analysis
Studentid	Number	8	Student Identification
GPA	Number	8	Student GPA
Nsemester	Number	8	Number of Semesters the student need to complete the degree
GpaPerSemester	Number	8	Average GPA per semester
NCrsPerSemester	Number	8	Number of Courses the Student took per semester as average
NCredPerSemester	Number	8	Number of credits or units the Student approved per semester
NN	Number	8	Nearest Neighbor Indicator

Table 2. SPS Template Indicator table.

Figure 5. SPS read query constraints (courses already taken).

Figure 6. SPS read-course description.

Field Name	Field Type	Field Size	Field Description
Status	Text	1	If active (value=null) template will be considered for analysis
Student ID	Number	8	Student identification number
Course	Text	12	Course identification code
CrsTitle	Text	50	Course Name or Title

Table 3 shows the additional field and the parameter combination represented. Another table used in the systems is the Course List table (Table 4). The Student Course Information screen (Figure 5) requires, for every course: the student ID, course ID, course title, semester, year, grade, and number of units approved by the student. The information will be stored in the Template Database,

which has a unique compound index using two fields: student ID and course.

The course description screen will require the status, course code, course title, course description, recommended semester according to the computer science suggested curriculum pattern, and the number of unit the course requires. The table has a simple primary index based on course code. The screening function determines the values allowed for every one of the parameters. Table 5 defines the minimum and maximum values allowed for each parameter. After the template indicators are computed, the screening function is invoked and it will inactivate any record that does not meet the values allowed per parameter. Each question SPS must answer is transformed into an indicator.

Field Name	Parameter				
	GPA	Nsemester	GpaPerSemester	NCrsPerSemester	NCredPerSemester

GPANSem	X	X			
GPAGPAPSem	X		X		
GPANCrsPSem	X			X	
GPANCredPSem	X				X
GPANSemGpaPSem	X	X	X		
GPANSemNCrsPSem	X	X		X	
GPANSemNCredPSem	X	X			X
GPAGpaPSemNCrsPSem	X		X	X	
GPAGpaPSemNCredPSem	X		X		X
GPANCrsPSemNCredPSem	X			X	X
GPANSemGpaPSemNCrsPSem	X	X	X	X	
GPANSemGpaPSemNCredPSem	X	X	X		X
GPANSemNCrsPSemNCredPSem	X	X		X	X
GPAGpaPSemNCrsPSemNCredPSem	X		X	X	X
GPANSemGpaPSemNCrsPSemNCredPSem	X	X	X	X	X
NSemGpaPSem		X	X		
NSemNCrsPSem		X		X	
NSemNCredPSem		X			X
NSemGpaPSemNCrsPSem		X	X	X	
NSemGpaPSemNCredPSem		X	X		X

NSemNCrsPSemNCredPSem		X		X	X
NSemGpaPSemNCredPSem		X	X	X	X
GpaPSemNCrsPSem			X	X	
GpaPSemNCredPSem			X		X
GpaPSemNCrsPSemNCredPSem			X	X	X
NCrsPSemNCredPSem				X	X

Table 3. Template Indicator field names as parametric combinations.

Field Name	Field Type	Field Size	Field Description
Status	Text	1	If active (value=null) template will be considered for analysis
Course	Text	12	Course identification code
CrsTitle	Text	50	Course Name or Title
CourseDescription	Text	50	Any additional information about the course
RecomendedSemester	Text	2	A value between 1 and 8
Units	Number	8	Number of units or credits approved by the student

Table 4. Course information.

The procedure used to compute the overall GPA is as follows:

1. For a specific student, determine the courses approved in the template database, exclude the courses with no grade if any.
2. For every course, transform the grade letter into a numeric value and multiple the numeric values by the number of units assigned to the course.
3. Accumulate grade value, multiply by number of units and accumulate total number of units approved.
4. Divide accumulated grade value by total number of units approved and round final value to 1 decimal.

The procedure used to compute the NumSemesters, is as follows:

1. For a specific student determines the number of semesters per year.
2. Count the number of different semesters per year.

The sample code in SQL language is as follows:

```
SELECT First(a.semester) AS Sem, First(a.year) AS Year
FROM templates AS a
WHERE (a.studentid = '100') AND ((a.year is not null) or (a.grade is not null))"
GROUP BY a.semester, a.year
```

The procedure used to compute the GpaPerSemester, is as follows:

1. For a specific student, determine the courses per semester and year.
2. For each list of courses compute the GPA.

3. Add the GPA per semester and divide by the total number of semesters to obtain the average GPA per semester

Field Name	Min	Max	Description
GPA	1	4	Student GPA
Nsemester	6	18	A max of 9 years are considered to complete the degree
GpaPerSemester	1	4	Average GPA per semester
NCrsPerSemester	1	10	A max of 10 courses is allowed for a student to take per semester
NCredPerSemester	1	30	A student taking more than 30 units per semester will be excluded

Table 5. Values allowed per parameter.

The procedure used to compute the CredPerSemester, is as follows:

1. For a specific student, determine the number of units per semester and year
2. Add the number of units per semester.
3. Divide the accumulated value by the total number of semester

The sample code in SQL language will be as follows:  
 SELECT First(a.semester) AS Sem, First(a.year) AS Year, Sum(a.units) AS NUnits  
 FROM templates AS a  
 WHERE (a.studentid = '100') AND ((a.year is not null) or (a.grade is not null))  
 GROUP BY a.semester, a.year

The procedure used to compute the CoursesPerSemester, is as follows:

1. For a specific student count the number of courses by semester and year
2. Sum the number of courses per semester.
3. Divide the accumulated value by the total number of semester

The sample code in SQL language will be as follows:  
 SELECT First(a.semester) AS Sem, First(a.year) AS Year, count(a.course) AS NCourse  
 FROM templates AS a  
 WHERE (a.studentid = '100') AND ((a.year is not null) or (a.grade is not null))  
 GROUP BY a.semester, a.year

Another indicator used in the SPS is the sequence of semesters that the student was enrolled in. A sequential number is assigned to every semester to facilitate the retrieval of specific blocks from the template.

### 3. Experiment Results

This section describes the working system in parts: the user's input of query parameters as the study constraints, SPS' interaction with the user through the results of the system-query functions, and the system performance.

#### 3.1 Query Input and Functions

The query input will be captured using the User Input Screen (Figure 7). The user will enter as constraints the goal overall GPA, the expected number of semesters to graduate, the expected semester GPA, the expected number of courses per semester to enroll in and the expected number of credits or units per semester to be received.

The screen will have three basic commands: Find, View Case, and Exit. Find will identify the most similar cases to the constraints inputted. Once the most similar cases are displayed, the user can highlight the most attractive one and, using the View Case command, request that transcript to be displayed. Exit will close the application.

The View Case command will display a detailed transcript or template for the user to analyze and determine the curriculum pattern the particular student followed when completing the degree. The result shows the student identification, course, course title, semester, year, grade, and number of units per course. The information is sorted by year, semester, and course code (Figure 8).

The SemNum (semester number) field allows the user to select a specific semester within the template and display the list of courses the student enrolled. The information displayed for the specific semester is the same as the information displayed in the "Detailed Template Information" report (Figure 9).

When the User Input Screen is opened, the templates are loaded and sorted by student identification number. At this point the averages for overall GPA, number of semesters, average GPA per semester, number of courses per semester, and number of credits per semester are computed using all the templates in the template database. If the user does not enter any parameter and click on the command "Find," no templates will be displayed. If the user enters information for only one parameter, the templates are sorted by that specific parameter. If the user enters one or more parameters, the templates are displayed, sorted by the pre-computed combined parameter.

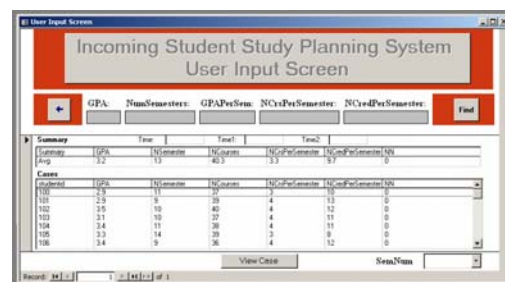


Figure 7. User query input screen.

Incoming Student Study Planning System Detailed Template Information						
studentid	course	credits	semester	year	grade	units
100	cs109	Operating System Prog	S	2006	B	3
100	cs191	Senior Project Part I	S	2006	A	2
100	cs480a1a2		S	2006	B	3
100	cs150	Senior Project Part I	F	2006	A+	2
100	cs480a1		F	2006	C+	3
100	cs480a3		S	2006	B	3
100	cs152	Computing Theory	S	2006	C	3
100	cs480b		S	2006	B	3
100	cs480c		F	2006	C+	3
100	cs480d		F	2006	C	4
100	cs480e		F	2006	C	4
100	cs121	Computer Software Eng	S	2004	B+	3
100	cs153	Business and Computer	S	2004	B+	3
100	cs20	Programming Concepts	F	2003	B	3
100	cs10		F	2003	B	3
100	cs130	Data Structures and Alg	S	2003	C	3
100	cs130	Computer Networks and	S	2003	B	3
100	cs15	Programming Concepts	S	2003	A	3
100	cs20	Discrete Structures for	S	2003	C	3
100	cs18	Assembly Language Prog	S	2003	B+	3
100	cs133	Hand-Oriented Comput	F	2002	C	3
100	cs480f	Calculus I	F	2002	B	4
100	cs480g	General Physics: Mecha	F	2002	B	4
100	cs480h	Introduction to Systems	C	2002	A	3
100	cs134	File Organization for Dal	F	2000	C	3

Figure 8. Detailed template information.

Incoming Student Study Planning System Detailed Template Information						
studentid	course	credits	semester	year	grade	units
101	cs15	Programming Concepts	F	2000	A	3
101	cs1		S	2000	B	3
101	cs2		S	2000	C	3
101	cs3		S	2000	C	3
101	cs4800	Calculus I	S	2000	B	4

Figure 9. Detailed template information for a specific semester number.

### 3.2 System Performance

If each nearest-neighbor index is to be measured in the Euclidian distance, the following steps is needed:

1. For every template, in the template database, and for every parameter, compute the difference between the average template parameter and template value for that parameter.
2. Square the difference.
3. Multiple the results by the weight assigned to each parameter (according to their significances).
4. Sum all the differences multiplied
5. Compute the square root of the sum.
6. Store the square root in the corresponding combined parameter field in the Template Indicator table.

But since the indices have been pre-sorted as each template was added, each parameter in the input is compared in the sorted list by a linear look-up, instead of applying the lengthy computation. Hence, the combined parameter field is identified and the templates are displayed sorted by nearest value first. After this is done, some basic descriptive indicators (average per parameter) are computed to help the user determine the typical case.

SPS performance resulted to be faster than expected (Figure 10). It was expected that the number of parameters entered by the user might increase the response time, but that behavior was not observed. As an average the system responded in about 15 milliseconds to the user request. As the number of parameters or constraints increased from 1 to 5 the response time

fluctuated from 15.2 to 16.8 milliseconds but the increase is not considered significant.

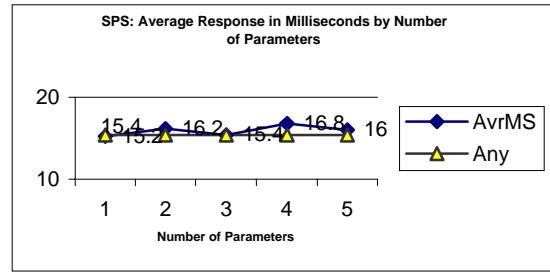


Figure 10. Average response in milliseconds by number of parameters

The performance observed is due to the pre-computation of the distance before the user enters the parameters and its storage in the combine parameter field. It is important to mention too that as every new case or template is added to the template database the distance is pre-computed and the result stored in the combine parameter field to facilitate future query or search.

## 4. Concluding Remarks

In the last section, we conclude our work with a summary, an analysis of some of our difficulties, and a brief description of possible future work.

### 4.1 Summary

We propose an expert system that combines the techniques of Decision Support System (DSS) and Case Base Reasoning System (CBRS) to experiment a simple and novel implementation called Study Planning System (SPS) for the aid of student advising. The solution collects past graduate records as templates of curriculum patterns that reveal advisable courses to enroll during a specific academic term. The templates help new students to determine the study pattern that best suits their needs. A new student using SPS is allowed to enter a set of constraints to be matched by the most similar templates.

The system requires low maintenance with the scalability of adding new templates to reflect new curriculum patterns, and thus, SPS can easily adopt new study plans without the need of considering changes in course prerequisites, graduation requirements, etc., that usually add complexity to the Rule-Based System (RBS) that is commonly used to build most student-advising systems.

SPS is a system based on statistical experiences and designed with the complete cycle of system development: research, design, prototype, testing, and documentation.

The research has comprised of the study and analysis of DSS, CBRS, RBS, and common student advising implementation. The design has included the concise definition of features, architecture, algorithms, and other techniques. The prototype has been implemented on a common computer workstation.

The nearest-neighbor indices used in SPS identify the similar cases or templates to his or her required study constraints. Due to the high cost of computing the nearest-neighbor indices at the time of each query, our implementation pre-builds the indices while adding templates with every needed parametric combination considered. A sorted list of template entries associated with each parametric combination stores the nearest-neighbor indices for that particular combination is thus generated. With the sorted lists, the computation time for every set of parameters entered at query can be reduced to a simple linear look-up process, making both the query process and the system-scaling process efficient. The scale-up process of building the probability matrix of the required first-order Markov chains is similar to that of building the indices. The query looks up the probability in the table at a row representing a course taken, to all the columns in that row, representing the probabilities of taking them as the next course. As a query option, the user can specify a probability threshold as the likelihood for only those over the threshold to appear as the next course [15, 16]. Without specifying any probability threshold, the system would display any existing course that occurs in the row (as all likelihood).

## 4.2 Analysis

Three issues occur regarding the weight of each unit in the selective parameters.

1. The heuristics we agree upon is that the default order of significance in parameters to be that the minimal GPA is more significant over the maximum unit number per academic term, and that the maximum unit number per academic term is more significant over the number of academic terms needed to graduate. Since these units lack normalization or conversion for similarity measurement, we advise the user to weigh the input parameters and determine their significance, and provide the order (if not by default).

2. The second issue is the number of parametric combinations grows in the order of factorial, the pre-sorting process when adding each new record can take lengthy time.

3. The reversal of the scalability has not been considered. As outdated graduate records in the system require deletion, the pre-sorted lists of nearest-neighbor

indices and the probability matrix need to have deletions and updates.

## 4.3 Future Work

SPS is supposed to be an online tool, however this implementation was only a prototype and a fully online version should be designed in order for the tool to be available and accessible to all students and advisors. SPS is first implemented as an experiment using graduate records collected within the Computer Science Undergraduate Program at Sacramento State, but the system can be enhanced to include other programs including graduate programs.

## 5. References

- [1] Weber, Rosina et al; A Textual Case-Based Reasoning Framework for Knowledge Management Applications; University of Wyoming at Laramie; 2001.
- [2] Carroll, Jan; Chappell, Gary; An Intelligent Universal Advisor; University of Texas at Houston; 1996.
- [3] Chi, Robert; Kiang, Melody; An Integrated Approach of Rule-Based and Case-Based Reasoning for Decision Support; University of Texas at Austin; 1991; ACM.
- [4] Kolodner, Janet; Case-Based Reasoning; Morgan Kaufmann Publishers, California 1993.
- [5] Leake, David B; Case-Based Reasoning: Experiences, Lessons, & Future Directions; MIT Press, Massachusetts 1996.
- [6] McKendree, Jean; Zaback, Jay; Planning for Advising; MCC Human Interface Laboratory, Austin Texas, 1988.
- [7] Murray, Scott W; Le Blanc, Louis A; A Decision Support System for Academic Advising; University of Arkansas at Little Rock; 1995.
- [8] Roussopoulos, Nick; Kelley, Stephen; Vincent, Frederic; Nearest Neighbor Queries; Department of Computer Science, University of Maryland; 1995.
- [9] Rousu, Juho; Case-based Planning; Helsinki University of Technology; Seminar on Knowledge Engineering; Fall 1997.
- [10] Sauter, Vicki L.; Decision Support Systems: An Applied Managerial Approach; John Wiley & Son, Inc.; 1997.
- [11] Siegfried, Robert M; Wittenstein, Adam M; Sharma, Tashi; An Automated Advising System for Course Selection and Scheduling; Adelphi University, New York; 2002.
- [12] Watson, Ian D.; Applying Case-Based Reasoning: Techniques for Enterprise Systems; Morgan Kaufmann Publishers, California 1997.
- [13] Watson, Ian; Marir, Farhi; Case-Based Reasoning: A review; University of Auckland, New Zealand; <http://www.ai.cbr.org/classroom/cbr-review.html>.

[14] Arya, Sunil; Mount, David M; Approximate Nearest Neighbor Queries in Fixed Dimensions; Department of Computer Science, University of Maryland, 1996.

[15] Leo Breiman. *Probability*. Original edition published by Addison-Wesley, 1968; reprinted by Society for Industrial and Applied Mathematics, 1992.

[16] J.L. Doob. *Stochastic Processes*. New York: John Wiley and Sons, 1953.